**Chapter Three**

**THEORETICAL FRAMEWORK**

**Present State of Covid-19 Tests and Vaccine Certificate Storage**

Documents and certificates given out by various units (private and public) for Covid-19 related tests such as rRt PCR and Cartridge – Based PCR 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’s 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 3.1 – 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.

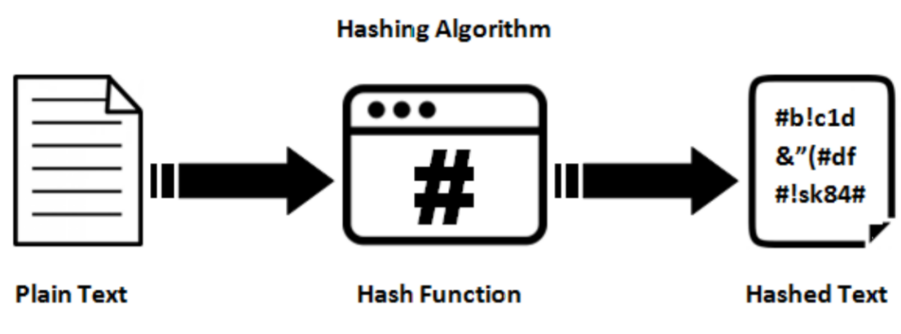
The next sections will discuss the different algorithms and frameworks to be used in order to achieve the proposed solution.

**Cryptographic Hash Functions**

A cryptographic hash function is an algorithm which maps data of any size (often called the "message") to a bit array of a fixed size ("hash value", "hash", or "message digest"). It is a one-way function, that is, a function which is practically infeasible to invert or reverse the computation. Ideally, the only way to find a message that produces a given hash is to attempt a brute-force search of possible inputs to see if they produce a match, or use a rainbow table of matched hashes. Cryptographic hash functions are a basic tool of modern cryptography.

The ideal cryptographic hash function has the following main properties:

* 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 infeasible to generate a message that yields a given hash value (i.e. to reverse the process that generated the given hash value)
* it is infeasible to find two different messages with the same hash value
* a small change to a message should change the hash value so extensively that a new hash value appears uncorrelated with the old hash value



**Figure 3.2 – Hashing Process**

Most cryptographic hash functions are designed to take a string of any length as input and produce a fixed-length hash value.

A cryptographic hash function must be able to withstand all known types of cryptanalytic attack. In theoretical cryptography, the security level of a cryptographic hash function has been defined using the following properties:

* **Pre-image resistance**

Given a hash value h, it should be difficult to find any message m such that h = hash(m). This concept is related to that of a one-way function. Functions that lack this property are vulnerable to preimage attacks.

* **Second pre-image resistance**

Given an input m1, it should be difficult to find a different input m2 such that hash(m1) = hash(m2). This property is sometimes referred to as weak collision resistance. Functions that lack this property are vulnerable to second-preimage attacks.

* **Collision resistance**

It should be difficult to find two different messages m1 and m2 such that hash(m1) = hash(m2). Such a pair is called a cryptographic hash collision. This property is sometimes referred to as strong collision resistance. It requires a hash value at least twice as long as that required for pre-image resistance; otherwise collisions may be found 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:

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.

Block Data

* Actual data (text, files)



**Figure 3.3 – Generic Blockchain Transactions**

Figure 3.3 shows how blockchain works given we have a simple data of text. The first block is called 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 initial state of the system. This is recorded 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.

The following properties of a consensus model are:

* The initial state of the system is agreed upon (e.g., the genesis block).
* Users agree to the consensus model by which blocks are added to the system.
* Every block is linked to the previous block by including the previous block header’s hash digest (except for the first ‘genesis’ block, which has no previous block and for which the hash of the previous block header is usually set to all zeros).
* Users can verify every block independently

**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 comparable in nature to Kademlia, the peer-to-peer distributed hash table (DHT) that is widely known for its use in the BitTorrent protocol.

IPFS is essentially a peer-to-peer system for retrieving and sharing IPFS objects. An IPFS object is a data structure with 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 the transactions in a block by producing a digital fingerprint of the entire set of transactions, thereby enabling a user to verify whether or not a transaction is included in a block. Merkle trees are created by repeatedly hashing pairs of nodes until there is only one hash left (this hash is called the Root Hash, or the Merkle Root). They are constructed from the bottom up, from hashes of individual transactions (known as Transaction IDs). Each leaf node is a hash of transactional data, and each non-leaf node is a hash of its previous hashes. Merkle trees are binary and therefore require an even number of leaf nodes. If the number of transactions is odd, the last hash will be duplicated once to create an even number of leaf nodes.



**Figure 3.4 – Merkle Tree Implementation using hashes**

A directed acyclic graph (DAG) is a conceptual representation of a series of activities. The order of the activities is depicted by a graph, which is visually presented as a set of circles, each one representing an activity, some of which are connected by lines, which represent the flow from one activity to another. Each circle is known as a “vertex” and each line is known as an “edge.” “Directed” means that each edge has a defined direction, so each edge necessarily represents a single directional flow from one vertex to another. “Acyclic” means that there are no loops (i.e., “cycles”) in the graph, so that for any given vertex, if you follow an edge that connects that vertex to another, there is no path in the graph to get back to that initial vertex.



**Figure 3.5 – DAG Illustration**

A Merkle DAG is a DAG where each node has an identifier, and this is the result of hashing the node's contents — any opaque payload carried by the node and the list of identifiers of its children — using a cryptographic hash function like SHA256. This brings some important considerations:

* Merkle DAGs can only be constructed from the leaves, that is, from nodes without children. Parents are added after children because the children's identifiers must be computed in advance to be able to link them. Every node in a Merkle DAG is the root of a (sub)Merkle DAG itself, and this subgraph is contained in the parent DAG.
* Merkle DAG nodes are immutable. Any change in a node would alter its identifier and thus affect all the ascendants in the DAG, essentially creating a different DAG. Take a look at this helpful illustration using bananas (opens new window)from our friends at Consensys.
* Merkle DAGs are similar to Merkle trees, but there are no balance requirements, and every node can carry a payload. In DAGs, several branches can re-converge or, in other words, a node can have several parents.

Identifying a data object (like a Merkle DAG node) by the value of its hash is referred to as content addressing. Thus, we name the node identifier as Content Identifier, or CID.



**Figure 3.6 – Merkle DAG implemented on a file system**

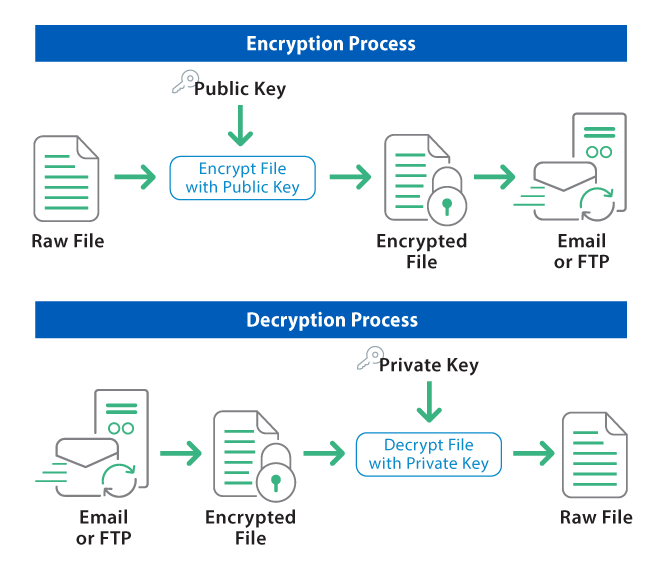
**Open Pretty Good Privacy**

OpenPGP is a key-based encryption method used to encrypt files so that only their intended recipient can receive and decrypt them. OpenPGP is used widely to secure e-mail communications, but its technology can also be applied to FTP.

OpenPGP works by using two cryptographic keys to secure files. A Public Key is used to encrypt the file so that only its corresponding Private Key can decrypt it.

The following is a step-by-step process of how OpenPGP Mode works with FTP.

1. The file to be uploaded is encrypted using a Public Key that the file's intended recipient has previously provided.
2. The encrypted file is uploaded to the FTP server.
3. The intended recipient retrieves the file from the FTP server.
4. Using the Private Key (which together with the Public Key used to encrypt the file initially comprises the Key Pair), the intended recipient decrypts the file and accesses its contents.

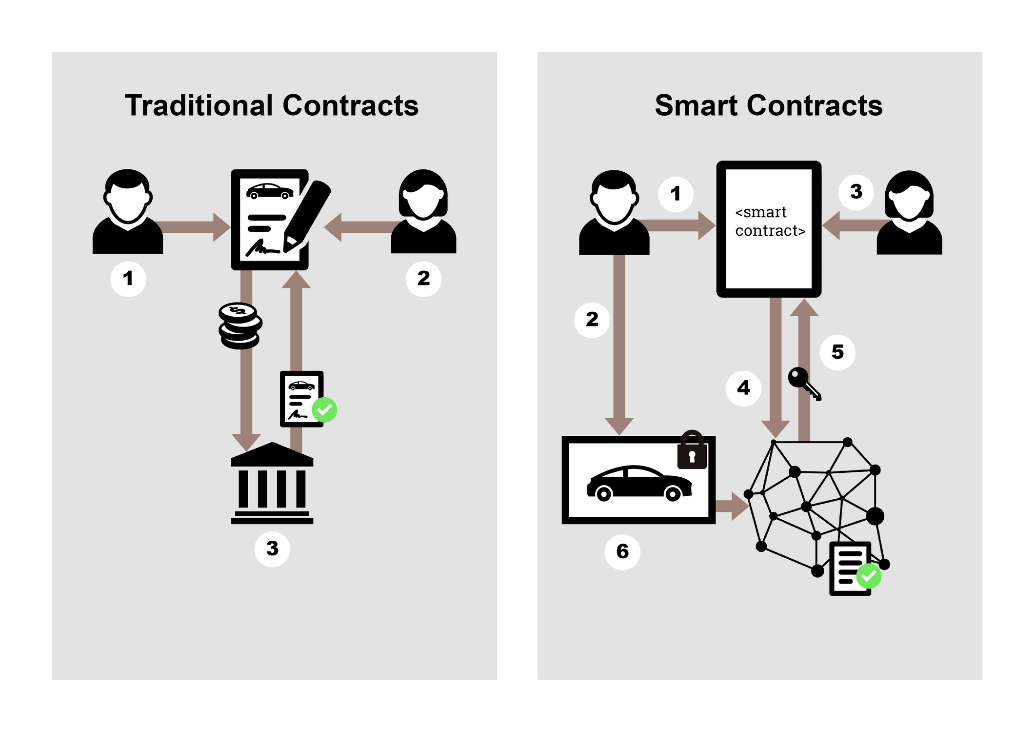


**Figure 3.7 – OpenPGP Two-way process**

**Smart Contract**

Smart Contract is a term used to describe computer code that automatically executes all or parts of an agreement and is stored on a blockchain-based platform. The code itself is replicated across multiple nodes of a blockchain and, therefore, benefits from the security, permanence and immutability that a blockchain offers. That replication also means that as each new block is added to the blockchain, the code is, in effect, executed. If the parties have indicated, by initiating a transaction, that certain parameters have been met, the code will execute the step triggered by those parameters. If no such transaction has been initiated, the code will not take any steps. Most smart contracts are written in one of the programming languages directly suited for such computer programs, such as Solidity.

If two individuals, Alice and Bob, do not know each other they also do not trust each other. When they want to make an agreement, they usually need a trusted third party that acts as intermediary. This intermediary verifies the transaction and can also enforce an action that was written into the agreement. With a smart contract in a blockchain, there is no need for a trusted intermediary because the clearing and settlement is automatically executed and enforced via blockchain technology.



**Figure 3.8 – Smart Contract comparison**

The following example in Figure shows the process of selling and buying a car between Alice and Bob. It also indicates the difference between a traditional contract and a smart contract. The comparison was originally created by BlockchainHub, this Figure was adopted by Braincept AG.

Traditional Contracts

1. Bob would like to sell his car.
2. Alice would like to buy a car.
3. A third party (intermediary) enables the trust that is needed in order to transfer the ownership of the car. Mostly different intermediaries are needed: motor vehicle registration authority, notary, insurance company. All middlemen take fees.

Smart Contracts

1. Bob would like to sell his car. He defines in a smart contract the conditions by which he will sell the car and signs the contract with his private key.
2. Bob leaves his car locked with a smart lock in his garage. The car has its own blockchain address and the smart lock is controlled by a smart contract.
3. Alice would like to buy a car. She finds Bobs car on an internet platform and signs the smart Bob’s contract with her private key. She adds the agreed amount from her blockchain address to Bob’s blockchain address.
4. As soon as the smart contract is executed the whole blockchain network will check if Bob is the real owner of the car and if Alice has enough money to buy the car.
5. If all peers in the blockchain network agree on the same state, it means that all conditions in the smart contract are met. The access code for the smart lock will be transferred to Alice and the blockchain address of the car will be registered to Alice. Bob will get the defined amount of money in his blockchain address.
6. Alice will be able to open the smart lock with her private key.

**Conceptual Framework**

This section aims to demonstrate the overview of the final product of this research. It identifies relevant variables, inputs, mappings and other components and how they will interact with each other. This includes all the underlying concepts and their associated mappings based on the system’s use.



**Figure 3.9 – 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 Test Result or 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.