**Chapter Two**

# **REVIEW OF RELATED LITERATURE**

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

**Merkle Tree**

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

1. Set *leaf* = 0.

2. Output:

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

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

3. Refresh Auth Nodes:

For h such that 2h divides leaf + 1:

• Set authh be the sole node value in stackh.

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

• *stackh.initialize(startnode,h)*.

4. Build Stacks:

For all *h* ∈ [0,H − 1]:

• stackh.update(2).

5. Loop

• Set *leaf* = *leaf* + 1.

• If *leaf* < 2*H* go to Step 2

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

1. Set *leaf* = 0.

2. Output:

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

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

3. Refresh Auth Nodes:

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

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

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

• *stackh.initialize(startnode,h).*

4. Build Stacks:

Repeat the following 2*H* − 1 times:

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

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

• *Stackfocus.update*(1).

5. Loop

• Set *leaf* = *leaf* + 1.

• If *leaf* < 2*H* go to Step 2.

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

1. Set *leaf* = 0.

2. Output:

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

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

3. Next Subtree:

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

• Set *Existi* = *Desirei*.

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

4. Grow Subtrees

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

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

**Distributed Hash Tables**

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

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

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

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

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

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

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

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

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

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

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

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

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

• No global knowledge

• Absence of single point of failures

**Blockchain**

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

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

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

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

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

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

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

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

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

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

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

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

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

**Proof-of-Authority**

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

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

**Asymmetric Encryption**

Asymmetric encryption systems are typically employed for discreetly delivering a symmetric encryption scheme's session key for message encryption. In fact, asymmetric and symmetric encryption techniques are frequently used in practice. (Fujisaki and Okamoto, 2011).

Goldwasser and Micali (1984) discussed the symmetric (aka private-key) encryption scheme as follows. Given by a pair of algorithms, *Π* = (*E,D*), where for every sufficiently large *k* ∈ *N*,

• *E*, the encryption algorithm, is a probabilistic polynomial-time (in *k*) algorithm that takes secret key *a* ∈ KSP and message *x* ∈ MSP, draws coins *r* uniformly from coin space COIN, and produces ciphertext *y := Ea(x;r).* This experiment is written as *y* ← *Ea(x).* The key, message, and coin spaces, KSP, MSP and COIN, are uniquely determined by *k*.

• *D*, the decryption algorithm, is a deterministic polynomial-time (in *k*) algorithm that takes secret key *a* ∈ KSP and ciphertext *y* ∈ {0, 1}∗, and outputs message *x := Da(y).*

We require that a symmetric encryption scheme should satisfy the correctness condition: For every sufficiently large *k* ∈ *N*, every *a* ∈ KSP and every *x* ∈ MSP, we always have *Da(Ea(x)) = x*.

Bellare et al. (1998)detailed the asymmetric (aka public-key) encryption scheme. Given by a triple of algorithms, *Π = (K, E,D)*, where for every sufficiently large *k* ∈ N:

• *K*, the key-generation algorithm, is a probabilistic polynomial-time (in *k*) algorithm which on input *1k*outputs a pair of strings, (*pk,sk*), called the public and

secret keys, respectively. This experiment is written as (*pk,sk*) ← *K*(1*k*).

• *E*, the encryption algorithm, is a probabilistic polynomial-time (in *k*) algorithm

that takes public key pk and message *x* ∈ *MSP*, draws coins *r* uniformly from coin

space COIN, and produces ciphertext *y := Epk(x;r).* This experiment is written as *y ← Epk(x).* The message and coin spaces, MSP and COIN, are uniquely determined by *pk*.

• *D*, the decryption algorithm, is a deterministic polynomial-time (in *k*) algorithm that takes secret key *sk* and ciphertext *y* ∈ {0, 1}∗, and returns message *x := Dsk(y).*

We require that an asymmetric encryption scheme should satisfy the following correctness condition: For every sufficiently large *k* ∈ *N*, every (*pk,sk*) generated by *K*(1*k*)

and every *x* ∈ *MSP*, we always have *Dsk(Epk(x)) = x*.

**Decentralized Storage, Blockchain and Medical Records**

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

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

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

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

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