2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)

Merkle Tree: A Fundamental Component of Blockchains

Haojun Liu*, †
School of Cyber Security
Guangdong University of Foreign Studies
Guangzhou, China
*20181003158@gdufs.edu.cn

Xinbo Luo*, †
Department of Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
Suzhou, China
*xinbo.luo19@student.xjtlu.edu.cn

Hongrui Liu*, †
College of Computer Science and Engineering
Chongqing University of Technology
Chongqing, China
*shady_l@2018.cqut.edu.cn

Xubo Xia*, †
Canadian International School, Singapore
Singapore
*xuxi2024@cis.edu.sg
†These authors contributed equally.

2021 International Conference on Electronic Information Engineering and Computer Science (EIECS) | 978-1-6654-1674-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/EIECS53707.2021.9588047

***Abstract*—With the increasing popularity of blockchain technology, Merkle trees (or hash trees) are playing significant roles in verifying and retrieving data for the implementation of blockchain for allowing efficient and secure verification of the contents of large data structures. This article gives systematical insights into Merkle trees with respect to their principles, properties, advantages, and applications.**

Keywords—Merkle tree; Hash functions; Blockchain; Bitcoin

I. INTRODUCTION

Realizations, such as smart contracts and cryptocurrencies, are based on blockchain due to their high efficiency and the characteristic of decentralization. David Chaum first proposed a similar idea in 1979, and its original purpose is to deter timestamps from being tampered with illegally [1]. However, it was 2009 when blockchain commenced being recognized by the public, after the publication of Bitcoin, designed by Satoshi Nakamoto [2]. In the last ten years, Bitcoin has developed at an extremely fast speed and received much attention. Over ten million Bitcoins were circulated globally, and they were totally worth about 4.6 billion dollars in 2014 [3]. Since the amount of cryptocurrency in the market is significant and enormous, it is crucial to guarantee security and efficiency during the trade procedure over the Internet. Merkle tree or hash tree is a kind of tree that obtains the value of parents and the root by combining each of its children and computing the hash value of the combination [4]. This unique data structure is nowadays applied to blockchain technology, since it can provide stability and efficiency for verifying and storing the transactions in cryptocurrency.

Firstly, the binary search tree was invented by P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard in 1960 [5]. Secondly, in 1979, a student at Stanford University—Ralph C. Merkle—patented a new data structure based on binary search tree and named this structure after himself, known as Merkle tree. He published a paper to explain his invention later in 1988. This paper is considered a major bedrock to initiate Blockchain and Cryptocurrencies. Ralph C. Merkle is a computer scientist who participated in inventing public-key cryptography and inventing cryptographic hashing. Finally in 2008, Satoshi Nakamoto invented Bitcoin, which is a significant implementation of the Merkle tree. While the Merkle tree is used in digital currencies, it is also widely used in other fields. It is used in hash-based cryptography since the Merkle tree is hash based [6]. Such as ensuring the integrity of E-medical records [7]; E-commerce user information encryption [8]; money transferring based on the Merkle tree [9]. Improving security and reliability of online data authentication [10]; and so on. Meanwhile, it is implemented in file systems, such as Btrfs, which is used by Facebook [11]; ZFS, which is used on Linux. And IPFS, which Microsoft ION uses.

As mentioned above, blockchain receives popularity due to the great demand for cryptocurrency, and Merkle tree can help maintain security and stability. This paper is organized as follows. The hash function and verification of transactions are two significant methods of the Merkle tree. As the fundamental of the Merkle tree, the definition of hash function and the procedures of transaction verification will be given in Section II. Then in Section III, a comparison between different data structures and Merkle tree and computational complexity will be formulated. Meanwhile, the complexity of verification of transactions will also be discussed. In section IV, some applications and different types of Merkle tree, say, the n-array tree, will be formulated. Finally, in Section V, a conclusion of this paper will be provided in the last section of this paper.

II. PRINCIPLES

A. *Definition of Hash Function*

The hash function is one of those important cryptographic primitives and fundamental elements in constructing the Merkle tree. It outputs a fixed-size digest of a given message of arbitrary length, usually called a hash value, uniquely representing the origin message [12]. In Merkle trees, hash function like SHA256 is used in the construction period, transforming numerous records of transactions into a compact form [13]. Meanwhile, designers apply the hash function to mining mechanisms for cryptocurrencies, say, Proof of Work in Bitcoin. As hash function is an essential part of Merkle trees and blockchain, it ought to provide security for regular mechanisms in the Merkle tree. It turns out that the hash function should provide at least three primary properties to assure the security and stability of applications.

B. *Verification of transactions*

Verification of a transaction is a distinctive and straightforward implementation of the Merkle tree in Bitcoin. For further investigation in verification, the mechanism of the Merkle tree needs to be understood first. Assume a Merkle tree is given and shown in Figure 4, the set of data on the bottom, which is called data blocks, contains information of transactions. The first step is to calculate hash values of these data blocks represented as $Node0i = hash(Data0i), i = 1,2,\dots,8$. After calculation, the results are saved in the Merkle tree, which is called a node. This set of nodes are also considered as leaf nodes. Two adjacent leaf nodes can then be added, and the hash value of it will be saved again. $Node1\left(\frac{(i+1)}{2}\right) = hash(Node0i + Node0(i+1), i = 1,3,5,7$. For some special situations where the number of nodes is odd, the one remaining node needs to be duplicated first and then put into the hash function. $Node1((i+1)/2) = hash(Node0i + Node0i), i = 2n+1, n = 1,2,3 \dots$ This procedure can be performed multiple times to obtain a final result. This result is considered a root node containing all the information of transactions in a block.

Therefore, an ideal binary Merkle tree can have the following properties:

- The number of leaves is $2^n, n = 1,2,3, \dots$
- Each node has 0 or 2 child nodes
- All leaf nodes should be on the same level

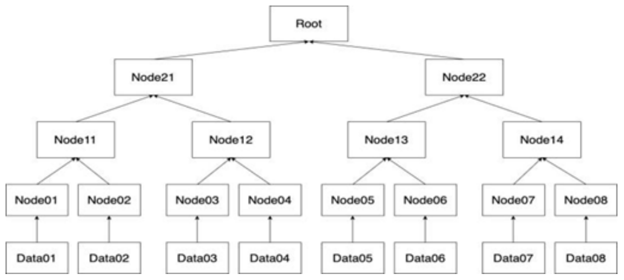

Fig.1. The basic structure of Merkle trees

The root node is stored in the block header, and the block header is a part of the blockchain which fetches hash values during the process of mining. The hash values of the last block, a random number, and information of all the transactions in the current block are included in a root node. Thus, having the root node in the block header makes the transaction tamper-proof because it contains the hash values of all the transactions within a block.

To verify if a transaction exists or untampered with, the procedures can be taken reversely. An efficient way to achieve this is to build another Merkle tree according to given information of the transaction. Firstly, hash values of nodes are required to locate the data block needed to be verified. Some child nodes that are not relevant to the branch are supposed to be given due to the structure of Merkle trees. After calculating using these values, a new root node can be generated. The data block can then be proved to be existing if the new root node is the same as the original one by comparing and vice versa.

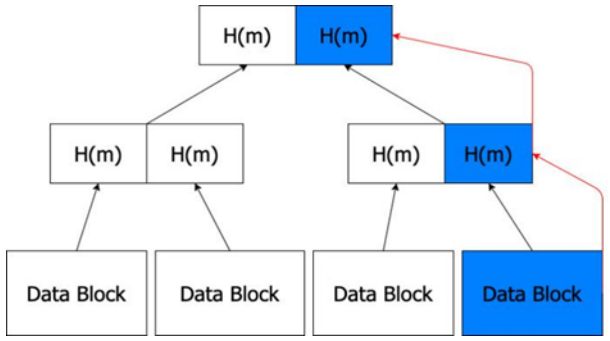

Fig.2. Process of verification

For example, consider a scenario that a client obtained the root node, and the Merkle tree can be a trusted source. Suppose another client wants to show that Data03 exists or not tampered with, the information required is the irrelevant hash values which are Node04, Node11, Node22 to build, a new Merkle tree based on the rules of its construction. Similarly, this scheme can be applied to digital signatures. The data blocks in the figure above can be replaced with digital signatures. In this scheme, fewer public keys are needed, and it is easier to manage all the public keys.

III. PROPERTIES

A. *Properties of Hash Function in Merkle tree*

Preimage Resistance. Preimage resistance, or one-wayness, means that for a given fixed length message digest y, it is computationally infeasible to figure out any specific input x that has a hash value of y [14]. For people who wants to break the hash function, it is difficult to retrieve the original message from the fingerprint. In other words, hash functions obscure the information of the original message, but it still maps the message with the fingerprint. Figure 3 below shows an explicit explanation.

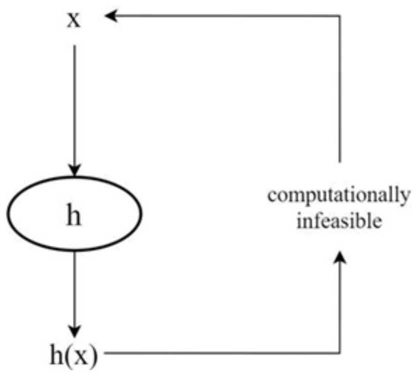

Fig.3. Preimage Resistance

Second-preimage Resistance. Second-preimage resistance is another crucial feature of hash functions. It can avoid finding out another specific message that has the same hash value as the original one. Such a feature is determinant in the Merkle tree and blockchain technology because it can deter malevolence forgeries and cheats. Since the original message is fixed, it is more challenging compared to the situation of preimage resistance. Figure 4 illustrates the definition of this property.

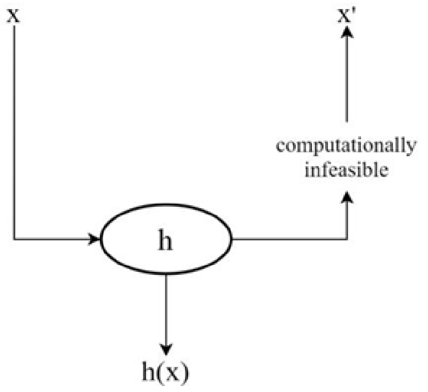

Fig.4. Second-preimage Resistance

Collision Resistance. For hash function, when it is computationally infeasible to find out two different original message $x1, x2$ such that $h(x1) = h(x2)$, we can call it collision resistance. An explanation is demonstrated in Figure 5.

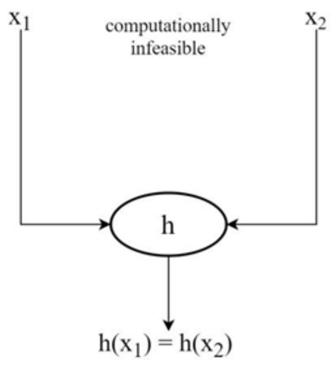

Fig.5. Collision Resistance

B. *Merkle Proof*

In the Bitcoin blockchain, Merkle proofs are used to validate the existence of the data in the Merkle tree. Merkle trees make extensive use of one-way hashing. Merkle proofs are established by hashing a hash's corresponding hash together and climbing up the tree until the root hash, which is or can be publicly known is obtained. Given that one-way hashes are intended to be collision free and deterministic algorithms, no two plaintext hash value should be the same. Merkle proofs are better explained with the following example in Figure 6.

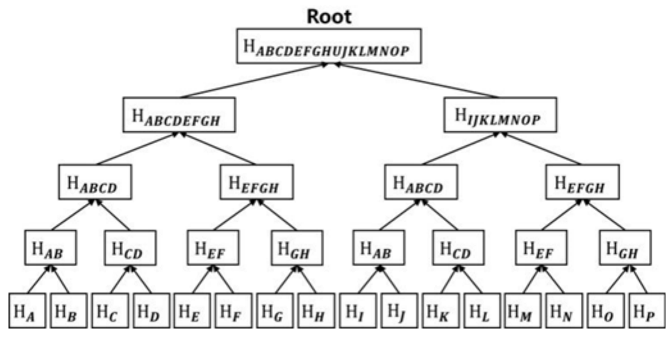

Fig.6. Merkle Proofs

In order to prove the existence of a leaf node, we can traverse from the bottom to the root. two hashes can be combined to generate a new one and finally the root value can be derived. According to the root value, we can validate the node.

IV. ADVANTAGES

A. *Comparison*

The data structure of the blockchain is discrete, and a block in Bitcoin is stored in a file. Thus, to know the verification path of a transaction in traditional cases, the blockchain where the transaction is located must be obtained. This is a replicated query process, and the client may need to go through all the blocks to find it. However, the Merkle tree enables transactions to be verified without people having the entire data in the block.

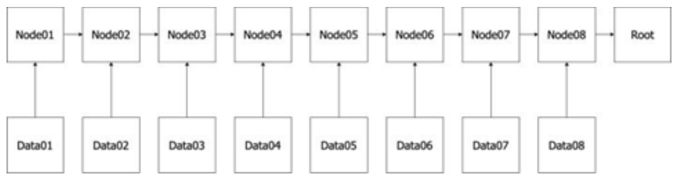

Fig.7. Pipeline data structure

A scenario where the structure of the data that a user received is a straight line of data blocks (shown in Figure 7) and is provided with hash values of all nodes to prove Data01 exists. The only way to complete this data structure is to hash all the values, combine all the hash values to get a string, and hash this string to get a root node. In this case, data can only be processed in one direction under this structure. In other words, all data from Node01 must be passed to and processed by Node02 and perform this procedure repeatedly until the last node is approached. Going back to the scenario mentioned previously if the receiver wants to check the existence of a data block in this model, all the data blocks that appear after that one should be checked before finishing the proof.

B. *The computational complexity of verification in Merkle trees*

As is discussed above, it consumes plenty of time to verify a file at a time, especially when the file is divided into several parts, stored in distinct places. Instead of transmitting the entire Merkle tree, the server merely sends the block headers and adds some relevant nodes in Merkle trees in the process of verification or authentication. As a result of that, the efficiency and superiority of Merkle trees ensure its application in verification, like Simple Verification Payment in Bitcoin. Since Merkle trees are typically implemented as binary trees, only the computational complexity of binary form Merkle trees will be included in this section.

Tampering Detection. On the occasion that we only need to detect whether the file has been tampered with illegally, the solution is to construct a new Merkle tree based on the file to be detected and compare the new Merkle root with the original one. Therefore, the calculation expenditure mostly depends on how many times the client needs to hash during the verification procedure. Generally speaking, it requires approximately $2n$ times hash operations if there exist n data blocks. Figure 8 compares the time consumption of different amounts of data blocks.

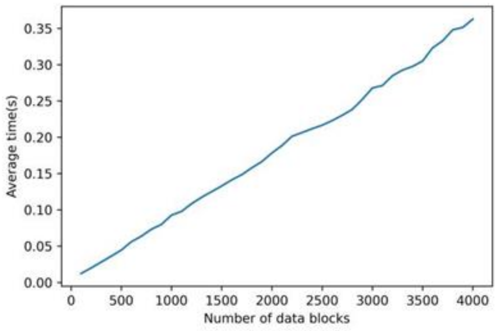

Fig.8. The plot of average time on computing Merkle root

The complexity of constructing a new Merkle tree and computing its Merkle root is $O(n)$, where the amount of data blocks is denoted by n.

Specific Tampering Detection. On this occasion, the client needs to figure out whether a data block has been distorted, given a specific data block ID. There is no need to construct another Merkle tree under this circumstance. For a more accurate result and more efficient computations, it usually involves the trusted source, which can provide the information of some original relevant nodes on the authentication path for the client to compute a new Merkle root [15]. By comparing the value of the new Merkle root with that of the original Merkle root, it can be judged that whether the specific data block has been tampered with. Obviously, in this occasion, the computational complexity is mainly determined by the depth of the Merkle tree, which is $O(\log_2 n)$, where the amount of data blocks is denoted by n.

Tampering Locating. Instead of discriminating whether a file has been tampered with or checking out whether a specific data block has been tampered with, tampering locating is more demanding and challenging than these previous situations. The definition requires a detailed location or ID of the tampered data blocks in the Merkle tree, if there are tampered data blocks. Based on this problem, it is necessary to construct a Merkle tree and then compare each node in the Merkle tree from top to bottom. Figure 9 illustrates an example of locating a tampered data block in a Merkle tree of depth 3. The process in Round 1 is very similar to Tampering Detection, since the file must be distorted if the Merkle root has changed. Next, both of the child nodes of the Merkle root are compared in Round 2 to show which one has been different. And then, similarly, we compare the child nodes of the changed node in the previous round and repeat the procedure until we reach the leaf node. Finally, the tampered data block can be found according to the leaf node discovered in the last round. For the client, construction of the Merkle tree and comparison with relevant child nodes from the trusted source, like a server, is necessary for the whole procedure. Therefore, the complexity of computation is mainly up to the time spent on construction and comparison.

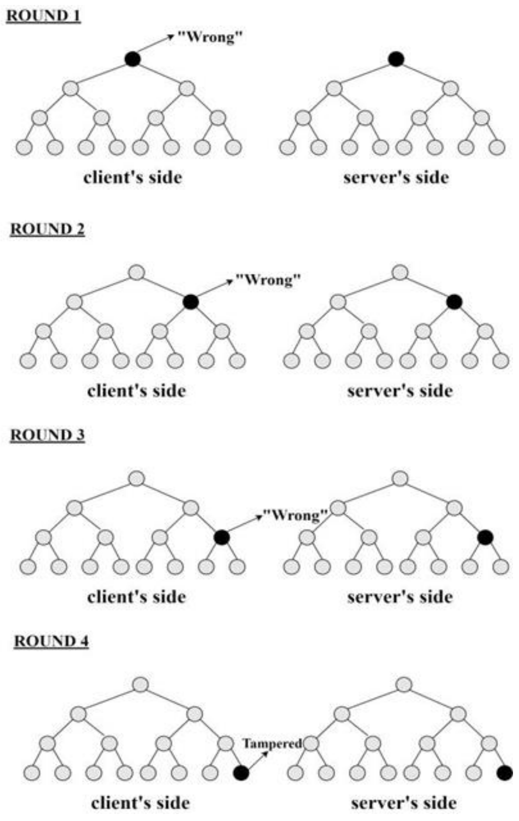

Fig.9. Explanation of the procedure of tampering locating

V. Application

A. *Sparse Tree in Bitcoin and Ethereum*

A Sparse Merkle Tree contains indexed data and each data point is placed at the leaf that corresponds to that data point's index. Due to this particular characteristic, Sparse Merkle trees have achieved a lot of application. Plasma Cash makes use of sparse Merkle trees to store information about deposited assets. Nowadays, Ethereum researchers are looking into Sparse Merkle trees as a replacement for the Merkle Patricia tries currently used to store Ethereum.

B. *Simplified Payment Verification (SPV)*

Compared with this data structure, the Merkle tree can be more efficient and a better solution to transaction verification. The sophisticated structure of Merkle trees also allows them to achieve SPV. SPV makes it unnecessary for a client to download the full blockchain but can still know if a transaction exists in that blockchain by confirming that the block headers are included and validating the relevant parts of the Merkle tree to confirm the transaction exists in the block [16]. A user or user's Bitcoin SPV wallet only needs a copy of the block headers of the longest chain, which are available by querying network nodes until it is apparent that the longest chain has been obtained. Then, the wallet uses the SPV client to obtain the Merkle branch linking the transaction to its block. Linking the transaction to a place in the active chain demonstrates that a network node has accepted it, and blocks added after it further establishes the confirmation. Therefore, implementing the Merkle tree in blockchain simplifies the process of verification and requires less information.

C. Patricia Tree in Ethereum

Currently, People are "investing" a lot of money into "assets" with no history of earning money, only to see their prices rise because other people are also putting money into them [17]. Essentially, a Merkle tree belongs to the group of hash trees. Under normal circumstances, Merkle trees are binary trees. For example, just binary Merkle trees are applied in the Bitcoin blockchain. However, Merkle trees are not just binary trees, giving some Merkle trees are actually ternary or even n-ary. Compared to binary Merkle trees in Bitcoin, the n-ary Merkle trees are applied in Ethereum. In a Merkle Patricia tree, a variety of nodes are included (empty nodes, leaf nodes, branch nodes, extended nodes). Ethereum is a project that is trying to build a universal technology. A technique to build all transaction-based state machine concepts.

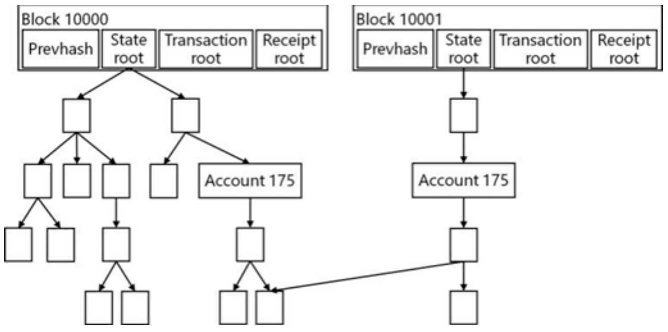

Fig.10. Tree structures for Ethereum

The transaction tree handles the first; the state tree handles the third and fourth, and the receipt tree handles the second. Calculating the first four query tasks is quite simple. The server simply finds the object, obtains the Merkle branch, and responds to the light client through the branch. The state tree also handles the fifth query task. Under this structure, transactions can be checked by the transaction root which totally differs from Bitcoin.

VI. Conclusion

The increasingly wide application of Merkle trees contributes to a promising prospect for blockchain. In summary, a Merkle tree is a hash tree measured with the SHA-256 algorithm. This unique characteristic leads to a more flexible manipulation for developers to create the blockchain. In the blockchain, characterized by decentralization, transactions appear to be transparent from the users. Therefore, the verification of transactions would pose a challenge. To resolve this problem, the blockchain only stores the block header for each blockchain other than its whole. In this case, clients may not trace and follow the transactions unless they check the public ledger. Moreover, Merkle trees are not all binary trees. Ternary or even n-ary trees constitute binary trees. e.g., Ethereum is not a binary tree. Now, the overall investigative direction of Merkle trees is not just blockchain. It is also widely applied in the P2P network, credential authentication, and verification. And further investigations are as follow: what are the advantages and disadvantages of Merkle trees. So far, n-ary Merkle tree such as the Ethereum can entail more essential information than that of binary trees, making it easier to check the transactions. In contrast, the drawback may lie in the fact that a more complex structure may require a larger capacity which may increase the workload and overhead for the servers. In the future, blockchain educational technology has provided evaluations and control tools for permanent learner credentials. Now, most EU nations are investigating educational blockchain. The technology creates an infrastructure to document, store and manage credentials and offer a durable record of achievements. And it can also be conducive to universities by cutting off administrative expenses and bureaucracy.

References

[1] Sherman, "On the Origins and Variations of Blockchain Technologies," *IEEE Security Privacy,* vol. 17, no. 1, pp. 72–77, 2019.

[2] P. K. Kaushal, A. Bagga and R. Sobti, "Evolution of bitcoin and security risk in bitcoin wallets," in *Proceedings of 2017 International Conference on Computer, Communications and Electronics,* Jaipur, India, 2017, pp. 172-177.

[3] A. Yelowitz, "Characteristics of Bitcoin Users: An Analysis of Google Search Data," *Applied Economics Letters,* vol. 22, no. 13, pp. 1-7, 2015.

[4] I. Chang and T.Liao "A survey of blockchain security issues and challenges," *International Journal of Network Security,* vol. 19, no. 5, pp. 653-659, 2017.

[5] T. Cormen, C. Charles, R. Rivest, C. Stein, Introduction to Algorithms (3rd ed.). USA: MIT Press and McGraw-Hill, 1990.

[6] M. Bosamia, D. Patel, "Current Trends and Future Implementation Possibilities of the Merkel Tree," *International Journal of Computer Sciences and Engineering,* vol. 6, no. 8, pp. 294-301, 2018.

[7] D. Koo, Y. Shin, J. Yun, J. Hur, "Improving Security and Reliability in Merkle tree-based Online Data Authentication with Leakage Resilience," *Applied Sciences,* vol. *8*, *no.*12, pp. 2532, 2018.

[8] B. Sharma, C. N. Sekharan, F. Zuo, "Merkle-tree based approach for ensuring integrity of electronic medical records," in *Proceedings of 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON),* New York, USA, 2018， pp. 983-987.

[9] Q. Li, "Research on E-Commerce User Information Encryption Technology Based on Merkle Hash Tree," in *Proceedings of 2019 International Conference on Robots & Intelligent System (ICRIS),* Haikou, China, 2019, pp. 365-369.

[10] S. Dhumwad, M. Sukhadeve, C. Naik, K. N. Manjunath, S. Prabhu, "A peer to peer money transfer using SHA256 and Merkle tree," in *Proceedings of 2017 23RD Annual International Conference in Advanced Computing and Communications (ADCOM),* Bangalore, India, 2017, pp. 40-43.

[11] O. Rodeh, J. Bacik, C. Mason, "BTRFS: The Linux B-tree filesystem," *ACM Transactions on Storage (TOS),* vol. *9, no.* 3, pp. 1-32, 2013.

[12] M. Singh and D. Garg, "Choosing Best Hashing Strategies and Hash Functions," in *Proceedings of 2009 IEEE International Advance Computing Conference,* Patiala, India, 2009, pp. 50-55.

[13] I. Giechaskiel, C. Cremers and K. B. Rasmussen, "When the Crypto in Cryptocurrencies Breaks: Bitcoin Security under Broken Primitives," *IEEE Security & Privacy,* vol. 16, no. 4, pp. 46-56, 2018.

[14] M. Wang and Y. Li, "Hash Function with Variable Output Length," in *Proceedings of 2015 International Conference on Network and Information Systems for Computers,* Wuhan, China, 2015, pp. 190-193.

[15] D. Berbecaru and L. Albertalli, "On the Performance and Use of a Space-Efficient Merkle Tree Traversal Algorithm in Real-Time Applications for

Wireless and Sensor Networks," in *Proceedings of 2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications,* Avignon, France, 2008, pp. 234-240.

[16] P. Ray, N. Kumar and D. Dash, “BLWN: Blockchain-based lightweight simplified payment verification in IoT-assisted e-healthcare,” *IEEE Systems Journal,* vol. 15, no. 1, pp. 134-145, 2020.

[17] Y. Sovbetov, "Factors influencing cryptocurrency prices: Evidence from bitcoin, ethereum, dash, litcoin, and monero." *Journal of Economics and Financial Analysis 2,* no. 2, pp 1-27, 2018.