ISSN: 2664-9640

Research on Pow Scheme and Blockchain Security Technology Based on Merkel Tree

Tan Tao

School of Arts and Sciences, Yangtze University, Jingzhou, China

Abstract

Blockchain technology solves the problem that traditional centralized-based systems are vulnerable to attacks by attackers through decentralized solutions. Based on the traditional Merkel tree, this paper proposes a proof-of-work mining algorithm based on the Merkel tree. First, a set of random numbers is generated according to certain rules and the Merkel tree is generated, and then a node is arbitrarily selected at each layer of the Merkel tree. The entire algorithm process requires a large amount of memory resources, which is in line with the mining algorithm in the blockchain. Need to occupy computer resources. Moreover, because the mining algorithm occupies a lot of system memory, it can increase the design difficulty of ASIC mining machines to a certain extent and alleviate the centralization problem of mining computing power.

Keywords

Merkel tree; proof of work; hash function; blockchain.

1. Introduction

With the rapid development of information technology, issues such as network security and privacy protection have attracted more and more attention. The traditional way to solve these problems is to design complex cryptographic protocols to strengthen security, but traditional solutions are based on centralized systems. The stronger the control of the system's central organization, the more vulnerable it is to attackers. However, the blockchain technology that has emerged in recent years has revolutionized the use of decentralized solutions. Bitcoin is the earliest application of blockchain technology. The founder Satoshi Nakamoto designed a proof-of-work mechanism to ensure the security of asset transfer in a decentralized system. In 1993, Cynthia Dwork and Moni NaorNaor first proposed the use of cost functions to combat spam attacks [1]. In 1997, Adam Back first proposed Hashcash, which uses a hash function as a cost function to solve spam and DOS attacks [2]. Although Hashcash could not completely block the attacker's attack on the mail system, but greatly increased the attacker's cost, it gave Satoshi Nakamoto an important inspiration and became one of the core technologies of Bitcoin POW afterwards. In 1999, Markus Jakobsson and Ari Juels formally proposed the term "Proof of Work" [3]. In 2008, with the release of the Bitcoin white paper, people re-recognized that POW is of great significance in solving the problem of decentralized consensus [4].

The mining algorithm used by Bitcoin is the SHA256 hash function, which has better hardware real-time performance [5]. However, because the SHA256 mining algorithm can be optimized to a certain extent, it may leave certain security risks [6]. In 2011, in order to improve the short confirmation time of Bitcoin, Litecoin changed the mining interval and other parameters of Bitcoin and issued Litecoin[7]. In order to prevent possible 51% attacks, Litecoin's mining algorithm uses a completely different Scrypt algorithm from Bitcoin [8]. The Scrypt algorithm needs to generate a 128KB cache during the mining process, which forces the miner to have a larger cache space, which can reduce the optimization effect of the dedicated miner to a certain extent. In 2015, Tromp proposed a graph-based POW algorithm. This algorithm first builds a

ISSN: 2664-9640

huge random graph data structure, and then finds a minimum loop in this graph to complete the proof of work [9]. The mining process of the blockchain actually produces a huge waste of social resources, and a large amount of electricity resources need to be wasted every year for mining [10].

Merkel tree is a data structure proposed by Ralph Merkle to verify data integrity [11]. The leaf nodes of the Merkel tree usually store the hash value of a data block or a single file, while the non-leaf nodes store the hash value of the connection of its child nodes. The appearance of Merkel tree provides a new solution for data integrity verification. At the same time, the Merkel tree is also applied to the HTTP response and verification in the Web server [12]. In 2007, Piotr Berman proposed a new algorithm that can optimize Merkel tree traversal, which can effectively reduce the time complexity and space complexity in the traversal process [13]. In 2008, Coelho proposed a Merkel tree-based difficulty verification algorithm, which takes a relatively stable time to solve specific problems and has no input parameters [14].

This paper uses the characteristics of the Merkel tree construction process to propose a POW mining algorithm based on the Merkel tree. This algorithm uses the structural characteristics of the Merkel tree to design a mechanism for randomly accessing the nodes of each layer of the Merkel tree. For the reliability of the mining algorithm, the value range of the depth of the Merkel tree is analyzed.

2. Typical POW mining process

2.1. Features of POW mining

The POW system can have a 50% fault tolerance rate, which is the highest fault tolerance rate among all blockchain consensus mechanisms. The high fault tolerance rate allows it to be applied in many public chains. After Bitcoin, a lot of digital currencies appeared, but in order to prevent 51% attacks, the specific mining algorithms were basically changed. The earliest application of POW on the blockchain was on Bitcoin. After that, many digital currencies used the POW mechanism. Although they use different specific mining algorithms, they are all based on the POW consensus mechanism and meet the typical characteristics of POW: (1) For the sender, a specific amount of work needs to be completed, which is generally to solve a mathematical problem. Get a solution to a difficult problem. Generally, this process takes a long time. Then send the solution you get to the receiver. (2) After receiving the sender's solution, the receiver can simply and quickly verify whether the solution is correct. If it is correct, it is considered that the request made by the sender is legitimate, and then the next data interaction process is carried out with it. If it is wrong, the sender's request is considered illegal, and data interaction with it is refused.

Generally speaking, mathematical puzzles that meet POW requirements should have the following characteristics:

It can be publicly verified. In a decentralized system, public verification is a must. (2) The single calculation cost is fixed. The resources consumed by one calculation process should be nearly the same. Under the same conditions, the cost of multiple calculations by the user should be linearly positively related to the number of times. For example, the cost of two calculations should be twice the cost of a single calculation. In this way, normal users can estimate the cost of using the system. (3) The single calculation cost is lower. This can minimize the impact on normal users, so that the system can be used at a lower cost. And the total cost=single cost*the number of times of accessing the system. For ordinary users with fewer visits, the total cost can be very low. Conversely, for an attacker who wants to send a large amount of spam, although the single cost is very low, the total cost must be high because of the many times of accessing the system. (4) One-way function. This is the core feature. A one-way function is essentially a function, and it must satisfy the fundamental characteristics of the function: input and output

ISSN: 2664-9640

have a fixed mapping relationship. Since there is a mapping relationship, if you use brute force cracking, that is, traverse all the mapping relationships, you can definitely find the corresponding input for a specific output. Then, this also indicates that the time complexity from input to output is 1), and the time complexity from output to input is. One more characteristic of one-way functions in cryptography is that the output result has a fixed length.

2.2. POW mining process

The typical POW mining process is actually a process of constantly solving mathematical problems. The mathematical problem used in the process of blockchain mining is generally to find a random number that meets certain conditions. Since the one-way hash function occupies less computer resources, it is generally selected as the mining algorithm. The mining process of the blockchain first links the non-random number block header data headerØ and the random number nonce into a complete block header data header to the mining algorithm. Through the operation of the hash function H, the fixed-length output result is hashed It is converted into a large integer by BigNumber function and compared with the prescribed difficulty value Difficulty. If it is less than the difficulty value, the mining is successful, otherwise the difficulty value must be changed and the new calculation process will be tried again.

The first step is to initialize the random number nonce first. Because the random number nonce needs to be traversed one by one on the mining nodes, for a single mining node that has not joined the mining pool, the random number nonce should start from 0.

The second step is to connect the non-nonce block header headerØand the random number nonce to form a complete block header. However, in general, this block header needs to be changed continuously. Every time the random number nonce is changed, the block header is also automatically changed once. Therefore, the block header at this time is a "temporary" state, and its final state is determined based on the mining results.

In the third step, the block header is used as the input of the hash function H, and the hash value hash is calculated.

Steps 4 to 6, convert the hash value hash to a large integer. If the large integer is greater than the difficulty value specified by the block, then the currently selected random number nonce is incorrect, and the nonce should be increased by 1, and then returned to Step 2.

Steps 7 to 8, convert the hash value hash into a large integer. If the large integer is less than the difficulty value specified by the block, then the currently selected random number nonce is correct, and the function outputs the nonce and the block header hash value hash.

3. POW scheme based on Merkel tree

3.1. The construction and characteristics of the Merkel tree

The Merkel tree is a binary tree with a depth of d, the total number of nodes is $2^{d+1}-1$, and the number of leaf nodes is 2^d. In order to obtain the root hash n0, first store the data cache to be verified in the leaf nodes of the Merkel tree in turn, and then select a one-way hash function H to calculate the hash value $n_{2d-1},....n_{2d+1-2}$. Then the internal nodes perform the pairwise combination operation $H(n_{2i+1}||n_{2i+2})$to get n_i, and finally get a tree root n_0, this tree root is called the root hash. The two child nodes of the i-th non-leaf node are n_{2i+1} and n_{2i+2} respectively. Merkel tree as shown.

ISSN: 2664-9640

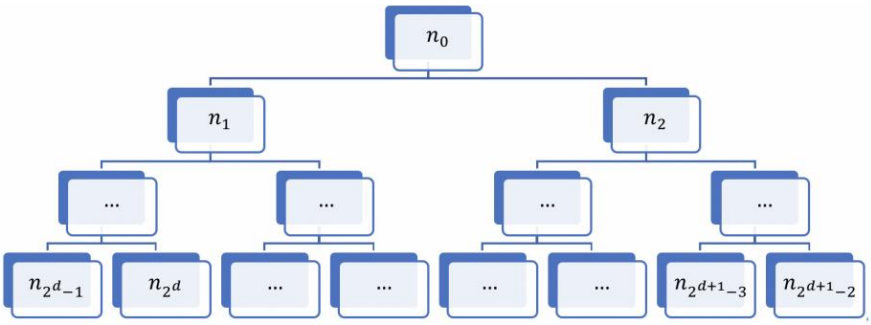

Figure1.Merkle Tree Construction

Table1.Merkle Tree Construction

Algorithm: *GetMerkleHash()*	
Input: *cache*	
Output: $n_0, n_1, \ldots, n_{2^{d+1}-3}, n_{2^{d+1}-2}$	
1	*for* $i = 0;\ i < 2^d;\ i = i + 1$:
2	$n_{2^d-1+i} = H(cache_i)$
3	*for* $i = 2^d - 2;\ i \geq 0;\ i = i - 1$:
4	$n_i = H(n_{2i+1} \parallel n_{2i+2})$
5	*return* $n_0, n_1, \ldots, n_{2^{d+1}-3}, n_{2^{d+1}-2}$

The verification process of the Merkel tree has already been mentioned, comparing the data difference of different trees can get the result with very little memory. Comparing the data integrity of N Bytes, the required memory space is only $\log_2 N * l$ Bits and the time complexity is O(logN). In the traditional one-by-one comparison, the required memory space is N Bytes, and the time complexity is O(N).

In view of the huge efficiency improvement of the Merkel tree, the extra space required to construct the Merkel tree is only $2N * l$ Bits. It can be seen from the above analysis and examples that the Merkel tree can effectively reduce the bandwidth required by the network in the process of verifying data integrity. In practical applications, using this Merkel tree to find different data blocks can not completely download all the data. In scenarios with low data access bandwidth, such as P2P network downloads, the missing parts of the data can be compared quickly and the amount of repeated downloads can be reduced. Therefore, in some scenarios where the network transmission rate is not high, the application of the Merkel tree can effectively improve the data interaction efficiency of the system.

3.2. Data preparation of Merkel tree mining algorithm

The input value of initializing the Merkel tree is the height of the block height, and the output value is all nodes of the Merkel tree with a depth of d. The process of initializing the Merkel tree function is shown in the table.

The obtained value is stored in the memory, so far, all the preparations for the mining data have been completed. The mining node searches for a suitable random number through the process described later to complete the mining process.

ISSN: 2664-9640

Table 2.Initial Merkle Tree

$Algorithm$: $InitialMerkle()$	
$Input$: $height$	
$Output$: $n_0, n_1, \dots, n_{2^{d+1}-3}, n_{2^{d+1}-2}$	
1	$section = \lfloor height/epoch \rfloor$
2	$if\ section = 0\ , then\ seed_0 = H(\{0\}^{256})$
3	$for\ i = 1; i \le section; i = i + 1:$
4	$seed_i = H(seed_{i-1})$
5	$data_{section} = R(seed_{section})$
6	$n_0, n_1, \dots, n_{2^{d+1}-3}, n_{2^{d+1}-2} = GetMerkleHash(data_{section})$

3.3. The process of calculating random numbers in the Merkel tree mining algorithm

Following the typical POW mining model, in order to obtain a random number that meets the requirements of the protocol, the input value of the function is the block header part without random number headerØand Merkle tree nodes n0, n1... The output result is the random number nonce and zone found The hash value of the block header hash. The specific process of the POW mining function MerkleMining() based on the Merkle tree is shown in the table.

Table 3.merkle Tree Mining Process

$Algorithm$: $MerkleMining()$	
$Input$: $header_\emptyset, n_0, n_1, \dots, n_{2^{d+1}-3}, n_{2^{d+1}-2}$	
$Output$: $nonce$, $hash$	
1	$Initial:\ nonce = 0$
2	$header = header_\emptyset \parallel nonce$
3	$h = H(header)$
4	$h_0 = H(h \parallel n_0)$
5	$for\ i = 1;\ i \le d;\ i = i + 1:$
6	$k = BigNumber(h_{i-1})\ mod\ 2^i$
7	$c = 2^i - 1$
8	$h_i = H(h_{i-1} \parallel n_{c+k})$
9	$hash = H(h_0 \parallel h_1 \parallel \cdots \parallel h_{d-1} \parallel h_d)$
10	$if\ BigNumber(hash) \ge Difficulty:$
11	$nonce = nonce + 1$
12	$return\ step\ 2$
13	$if\ BigNumber(hash) < Difficulty:$
14	$return\ nonce$, $hash$

3.4. Workload verification of Merkel tree mining algorithm

After a mining node successfully digs a block, it needs to send a new block to neighboring nodes. Neighboring nodes need to verify the validity of the new block. Only blocks that are valid and meet the workload requirements can be added to the chain. For a general full node, verifying the validity of a block only needs to bring the hash value and corresponding height value of the block into the mining function, and perform a mining operation to get the validity result.

The input value of the verification function is the height of the block and the header of the block header, and the output value is the result of the verification, which is of Boolean type. The function VerifyMiningMerkle() process to verify the validity of the block is shown in the table.

ISSN: 2664-9640

Table 4.Merkle Tree Verify Process

$Algorithm: VerifyMiningMerkle()$
$Input: height, header$
$Output: true\ or\ false$
$n_0, n_1, \dots, n_{2^{d+1}-3}, n_{2^{d+1}-2} = InitialMerkle(height)$
$h = H(header)$
$h_0 = H(h \parallel n_0)$
$for\ i = 1;\ i \le d;\ i = i + 1:$
$\quad k = BigNumber(h_{i-1})\ mod\ 2^i$
$\quad c = 2^i - 1$
$\quad h_i = H(h_{i-1} \parallel n_{c+k})$
$hash = H(h_0 \parallel h_1 \parallel \cdots \parallel h_{d-1} \parallel h_d)$
$if\ BigNumber(hash) \ge Difficulty:$
$\quad return\ false$
$if\ BigNumber(hash) < Difficulty:$
$\quad return\ true$

$Algorithm: VerifyMiningLight()$
$Input: height, header$
$Output: true\ or\ false$
$h = H(header)$
$n_0 = Request(0, height)$
$h_0 = H(h \parallel n_0)$
$for\ i = 1;\ i \le d;\ i = i + 1:$
$\quad k = BigNumber(h_{i-1})\ mod\ 2^i$
$\quad c = 2^i - 1$
$\quad n_{c+k} = Request(c + k, height)$
$\quad h_i = H(h_{i-1} \parallel n_{c+k})$
$hash = H(h_0 \parallel h_1 \parallel \cdots \parallel h_{d-1} \parallel h_d)$
$if\ BigNumber(hash) \ge Difficulty:$
$\quad return\ false$
$if\ BigNumber(hash) < Difficulty:$
$\quad return\ true$

Table 5.Light Node Verify Process

3.5. Light node verification Merkel tree mining algorithm results

In the blockchain, not all nodes mine or store complete block data. Some nodes that only undertake simple payment functions or only verify the validity of the block are light nodes. When light nodes verify the validity of transactions or blocks, they often use Merkel trees to reduce the burden of data processing on light nodes, and at the same time, it can also reduce the scale of data requests to nearby nodes.

Applied to the Merkel tree on the proof of work, if a light node wants to verify the validity of a block, it does not need to spend time to generate a huge Merkel tree. It only needs to randomly request a specific Merkel from neighboring nodes. The tree node will do. Here you need to define a function Request():ni=Request(i,height).The input value of the Request function is the requested Merkel tree node number i and the block height Height, and the return value is the specific Merkel tree node n_i. This function is mainly used for Merkel tree nodes that request specific block numbers from neighboring nodes.

The input value of the light node verification function is the height of the block and the header of the block, and the output value is the result of the verification, which is of Boolean type. The process of the function VerifyMining Light to verify the validity of the block is shown in the table.

ISSN: 2664-9640

3.6. Time complexity analysis of Merkel tree mining algorithm

Merkle tree has two cycles in the constructor GetMerkleHash(). The calculation of the first cycle process first needs to calculate the leaf nodes 2^d times, and the second cycle to calculate the non-leaf nodes needs to calculate $2^d - 1$times, and the total calculation times are $2^{d+1} - 1$. And because the data size to be constructed is $N= 2^d$, the total number of calculations is $2N-1$. It can be seen that the time complexity of constructing a Merkel tree is O(N).

If the Merkel tree has been constructed in advance, in the process of calculating the random number MerkleMining(), d calculations are required to obtain d hash values, and d is the depth of the Merkle tree, which is similar to Merkle Mining. The number of leaves of the tree $N= 2^d$, that is $d=\log_2 N$. It can be seen that the time complexity of calculating random numbers is O(logN). For nodes that have not constructed the Merkel tree in advance, because a node needs to be obtained at each depth, and from the above analysis, it can be seen that the time responsibility for calculating the Merkel tree node is O(N), and the comprehensive time is complicated The degree is O(NlogN).

For full nodes and light nodes, the time required to verify the validity of the block is the same as the time required for a single calculation of the random number nonce, so the time complexity is O(logN).

3.7. Space complexity analysis of Merkel tree mining algorithm

In the mining process, the main memory space occupied is the structure and mining of the Merkel tree. For a Merkel tree with a depth of d, the memory space occupied by it is proportional to 2d+1. After the Merkel tree is constructed, the memory space consumed by the random number can be estimated according to the size of the depth d. Because the calculation of random numbers requires selecting a node for each depth, the memory space consumed in the process of calculating random numbers is positively correlated with the depth d. And because the relationship between depth d and data size N is $N= 2^d$, the process of comprehensively constructing Merkel trees and calculating random numbers has a space complexity of O(N+logN), that is, O(N).

In the process of verifying the validity of a block, for a full node, the space complexity of constructing a Merkel tree of a specific block is O(N). To verify the validity of the block, d hashes need to be calculated, and the space complexity is O(logN). In summary, the space complexity of verification workload is O(N).

For light nodes, because there is no need to construct a complete Merkel tree, it only needs to request specific Merkel tree nodes from neighboring nodes. The number of requested nodes is proportional to the Merkel tree depth d. Then the space complexity of the light node verification workload is O(logN).

4. Conclusion

This paper uses the characteristics of the Merkel tree construction process to propose a POW mining algorithm based on the Merkel tree. This algorithm uses the structural characteristics of the Merkel tree to design a mechanism for randomly accessing the nodes of each layer of the Merkel tree. For the reliability of the mining algorithm, the value range of the depth of the Merkel tree is analyzed.

References

[1] Dwork C, Naor M. Pricing via processing or combatting junk mail[C]//Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 1992: 139-147.

[2] Back A. Hashcash-a denial of service counter-measure[J/OL]. 2002. ftp://sunsite. icm.edu. pl/site/replay.old/programs/hashcash.

ISSN: 2664-9640

[3] Jakobsson M, Juels A. Proofs of work and bread pudding protocols[M]//Secure Information Networks. Springer, Boston, MA, 1999: 258-272.
[4] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J/OL]. 2008. http://www. academia. edu/download/32413652/BitCoin_P2P_electronic_cash_system.pdf
[5] M'Raihi D, Machani S, Pei M, et al. Totp: Time-based one-time password algorithm[R]. 2011.
[6] Courtois N T, Grajek M, Naik R. Optimizing sha256 in bitcoin mining[C]//International Conference on Cryptography and Security Systems. Springer, Berlin, Heidelberg, 2014: 131-144.
[7] Lee C. Litecoin[EB/OL]. 2011. https://bitcointalk.org/index.php?topic=47417.0
[8] Percival C, Josefsson S. The scrypt password-based key derivation function[R]. 2016.
[9]Tromp J. Cuckoo cycle: a memory bound graph-theoretic proof-ofwork[C]//International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2015: 49-62.
[10]O'Dwyer K J, Malone D. Bitcoin mining and its energy footprint[C]//25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)
[11]Merkle R C. A digital signature based on a conventional encryption function[C]//Conference on the Theory and Application of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1987: 369-378
[12]Bayardo R J, Sorensen J. Merkle tree authentication of HTTP responses[C]//Special interest tracks and posters of the 14th international conference on World Wide Web. ACM, 2005: 1182-1183
[13]Berman P, Karpinski M, Nekrich Y. Optimal trade-off for Merkle tree traversal[J]. Theoretical Computer Science, 2007, 372(1): 26-36.
[14]Coelho F. An (almost) constant-effort solution-verification proof-of-work protocol based on merkle trees[C]//International Conference on Cryptology in Africa. Springer, Berlin, Heidelberg, 2008: 80-93.