УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
май–июнь 2022 Том 22 № 3 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
May–June 2022 Vol. 22 No 3 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2022-22-3-538-546

Efficient incremental hash chain with probabilistic filter-based method to update blockchain light nodes

Maher A. Maalla[1]✉, Sergey V. Bezzateev[2]

[1,2] ITMO University, Saint Petersburg, 197101, Russian Federation
[2] Saint Petersburg State University of Aerospace Instrumentation, Saint Petersburg, 190000, Russian Federation

[1] maher.malla7@gmail.com✉, https://orcid.org/0000-0002-4806-8608
[2] bsv@aanet.ru, https://orcid.org/0000-0002-0924-6221

Abstract
In blockchain, ensuring integrity of data when updating distributed ledgers is a challenging and very fundamental process. Most of blockchain networks use Merkle tree to verify the authenticity of data received from other peers on the network. However, creating Merkle tree for each block in the network and composing Merkle branch for every transaction verification request are time-consuming process requiring heavy computations. Moreover, sending these data through the network generates a lot of traffic. Therefore, we proposed an updated mechanism that uses incremental hash chain with probabilistic filter to verify block data, provide a proof of data integrity and efficiently update blockchain light nodes. In this article, we prove that our model provides better performance and less required computations than Merkle tree while maintaining the same security level.

Keywords
Merkle tree, blockchain, hash chain, probabilistic filter, hash function, integrity

For citation: Maalla M.A., Bezzateev S.V. Efficient incremental hash chain with probabilistic filter-based method to update blockchain light nodes. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2022, vol. 22, no. 3, pp. 538–546. doi: 10.17586/2226-1494-2022-22-3-538-546

УДК 004.056.55

Эффективная инкрементная хеш-цепочка с вероятностным методом на основе фильтра для обновления легких узлов блокчейна

Махер Аднан Маалла[1]✉, Сергей Валентинович Беззатеев[2]

[1,2] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
[2] Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, 190000, Российская Федерация

[1] maher.malla7@gmail.com✉, https://orcid.org/0000-0002-4806-8608
[2] bsv@aanet.ru, https://orcid.org/0000-0002-0924-6221

Аннотация
В блокчейне обеспечение целостности данных при обновлении распределенных реестров является сложным и фундаментальным процессом. Большинство сетей блокчейнов для проверки подлинности данных, полученных от других одноранговых узлов в сети, используют дерево Меркла. Создание дерева Меркла для каждого блока в сети и составление ветви дерева для каждого запроса на проверку транзакции является трудоемким процессом, требующим больших вычислений. Кроме того, отправка этих данных по сети генерирует большой трафик. В работе предложен обновленный механизм, использующий инкрементную хеш-цепочку с вероятностным фильтром для проверки данных блока, предоставления доказательства целостности данных и эффективного обновления легких узлов блокчейна. Доказано, что предложенная модель обеспечивает более высокую производительность и требует меньше вычислений, чем дерево Меркла, сохраняя при этом тот же уровень безопасности.

Ключевые слова
дерево Меркла, блокчейн, хеш-цепочка, вероятностный фильтр, хеш-функция, целостность

Ссылка для цитирования: Маалла М.А., Беззатеев С.В. Эффективная инкрементная хеш-цепочка с вероятностным методом на основе фильтра для обновления легких узлов блокчейна // Научно-технический вестник информационных технологий, механики и оптики. 2022. Т. 22, № 3. С. 538–546 (на англ. яз.). doi: 10.17586/2226-1494-2022-22-3-538-546

Introduction

Blockchain is a peer-to-peer network that manages a distributed ledger and commits to some consensus protocol. It is considered as a promising and successful technology. In addition to its main importance in digital payment processing and money transfers, it has many applications as in supply chains, digital voting, immutable data backup, medical recordkeeping, and many other fields.

In its simple definition, blockchain is a way to encapsulate data in blocks where these blocks are linked through hash values to create an immutable chain of blocks which is very fundamental concept in blockchain to keep the data authentic without any possibility of updating or manipulating the already added blocks to the blockchain. Fig. 1 shows basic structure of blockchain, each block (n) in the blockchain consists of its hash (n) and the hash of the previous block ($n - 1$). In this way, every block hash is used in the next block header which leads to creating a chain of linked blocks.

This chain of hashes gives a way to verify the integrity of the blockchain by recalculating the hash for a block and comparing with the hash value in the block, then using this value to validate the next block. Therefore, any change in any block will lead to wrong hash value which in turn causes the verification process to fail for all the following blocks. This model maintains the integrity of the blockchain and provides an immutable distributed ledger system. In order to verify the data of blocks, which are basically transactions, blockchain relies on Merkle tree to verify all transactions existed in the block body using Merkle root hash which is stored in the block header [1].

The success of the blockchain concept is connected with the financial success of Bitcoin. Therefore, we will consider Bitcoin as our case study to explain blockchain structure in more detail and understand why Merkle tree is fundamental in blockchain. Fig. 2 demonstrates Bitcoin's detailed structure and how Merkle tree is constructed from all transactions in a block to produce Merkle root which is used to fully verify these transactions.

In Fig. 2, T is timestamp of the block which indicates the exact moment in which the block has been mined and validated by blockchain network; N is the nonce of the block which is a number added by miners to meet difficulty level restriction; V is the blockchain version number; *diff* is the difficulty target for the block; and H_x is the hash value of transaction x.

We will study Merkle tree model in blockchain and the process of verifying blocks transactions, and we will introduce our proposed model which replaces Merkle tree and provides a more efficient mechanism that requires less time and fewer computations for the verification process.

Hash chain

The idea of hash chains was first proposed by Lamport to facilitate safeguarding of one time password schemes (OTPs) when the attacker is able to eavesdrop on communications. Since then it has been employed in a wide range of applications, mainly in blockchain. Fig. 3 shows its concept.

We generate a chain of hashes s^j (with j natural number ≥ 0) by using a hash function h. Every element s^k from the hash chain is computed by applying hash function h on the previous element s^{k-1}; then we use these outputs in inverse way [2].

One of the most famous and used type of hash chain is binary hash chain or Merkle tree [3]. It is used in many applications including blockchain [4–6], health care [7, 8], financial transactions [9–11], cloud computing [12, 13] and smart grid [14]. Many modified versions of Merkle tree have been introduced to provide better performance. Modified Merkle tree versions have been proposed to provide better time and space [12, 13, 15–17]. All these

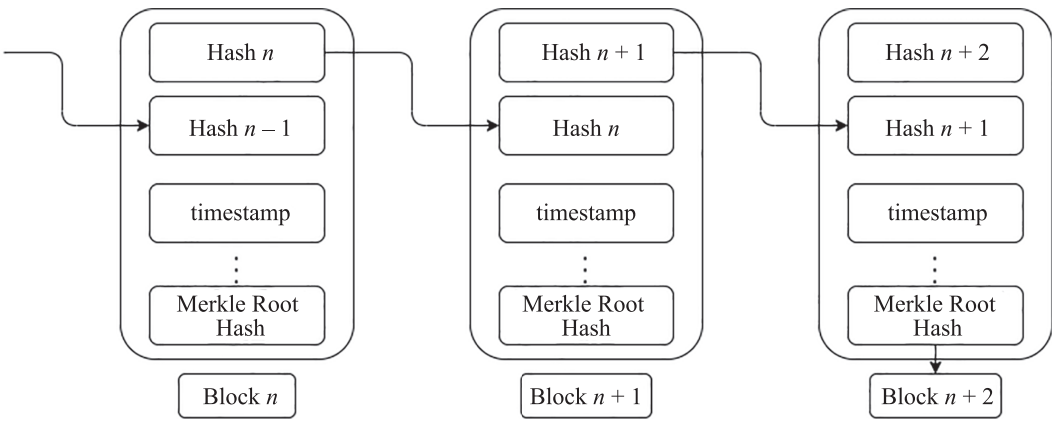

Fig. 1. Blockchain basic structure

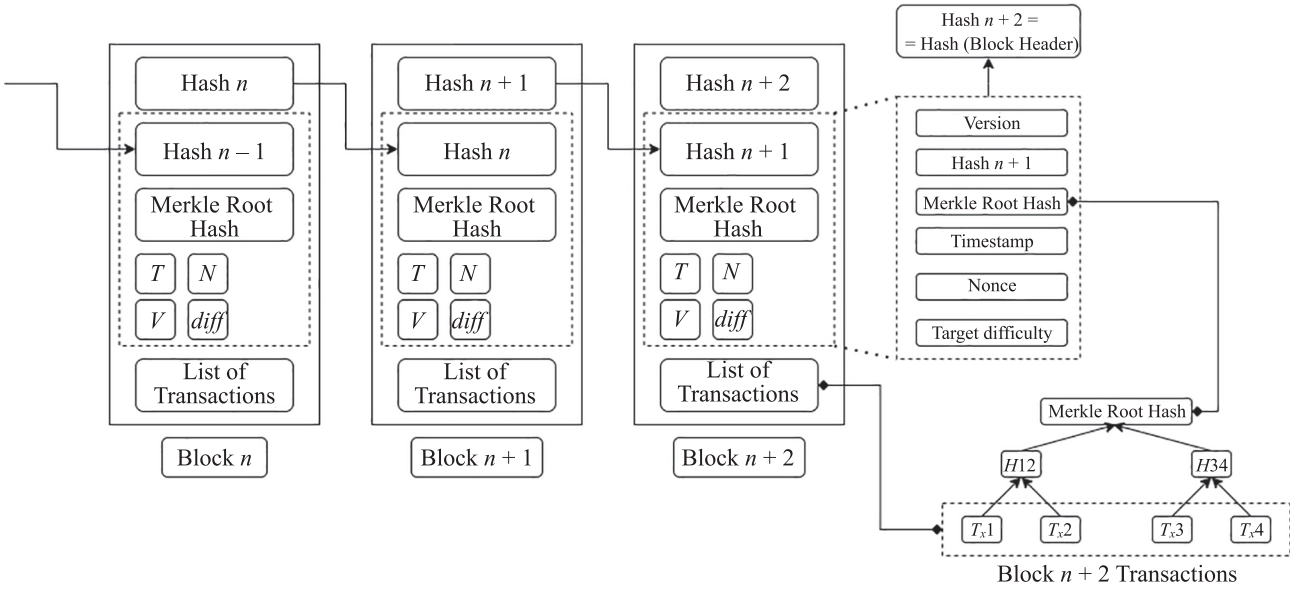

Fig. 2. Bitcoin's detailed structure

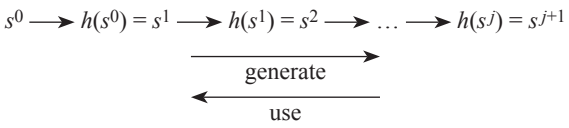

Fig. 3. Hash chain basic concept

researches show that Merkle tree plays key role in many applications, and many modifications has been made to match the variety of these applications.

Merkle tree construction

In blockchain, every block contains a list of transactions. Merkle tree is constructed over these transactions providing a model to verify integrity of these transactions and that they are in correct order. Therefore, every block has its own Merkle tree presenting all transactions and identifying by Merkle root.

Fig. 4 shows how to construct Merkle tree for a certain block. Initially, all transactions inside the block are hashed using cryptographic hash function, for instance SHA256 as in Bitcoin [1]. After that, every two consecutive hash values are concatenated and then hashed using the same hash function forming their parents.

The same process is performed to the next consecutive hashes and repeated until it becomes a single hash value which is called Merkle root. (If there is an odd number of transactions, last transaction is doubled and its hash is concatenated with itself). Finally, the Merkle root is stored in the block header and distributed to all peers in blockchain network.

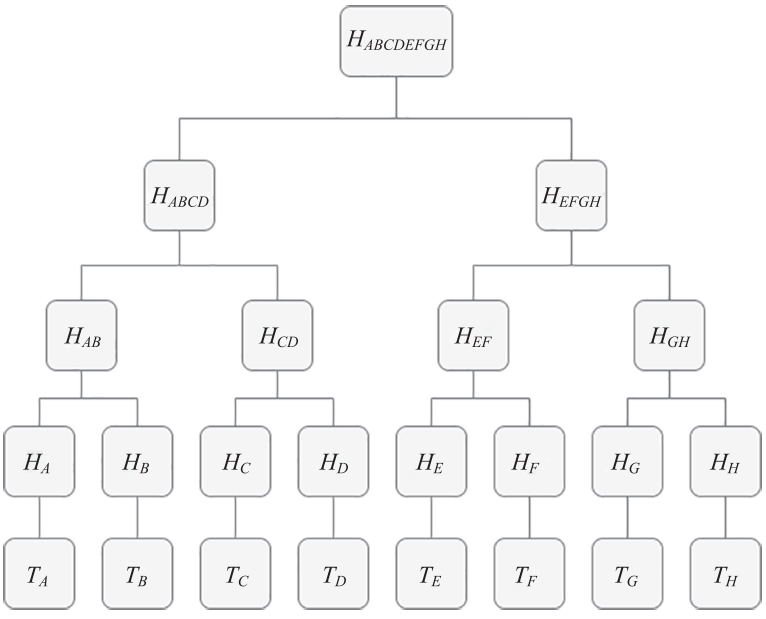

Fig. 4. Construction of binary Merkle tree

Merkle tree verification

In blockchain networks that used the concept of light nodes which don't store the whole blockchain locally, they just store headers of the blocks. Therefore, they can't verify the authenticity of some transactions by themselves; they have to rely on other trusted full nodes to provide the required data to the verification process.

Fig. 5 shows the process of verification on a certain transaction in a block. Merkle tree allows peers to verify a specific transaction without downloading the whole block data. When a user wants to verify a transaction, he calculates its hash (the black box in Fig. 5), and then he doesn't require the whole Merkle tree, it requires only some hash values from the tree (Merkle branch) in order to be able to verify the integrity of the transaction, as shown in the blue boxes in Fig. 5 for transaction T_D.

Merkle tree analysis

In our analysis we study time complexity and space requirements for binary Merkle tree to demonstrate the cost of computing the Merkle tree and the required operations to verify block data through Merkle branch.

Let's consider a block with T transactions. Binary Merkle tree for this block has the following attributes:

— The number of leaves is T.
— The number of internal Leaves is $T - 1$.
— The total number of nodes is $n = 2T - 1$.
— The height for n nodes is $h = \log_2(n)$.

Consider Merkle tree performance and issue analysis:

1. Construction time complexity

   To construct Merkle tree for n nodes, we simply need to calculate n hash values. Therefore, the construction time complexity is $O(n)$.

2. Adding new node time complexity

   To add a new node Merkle tree for n node, we simply have to recalculate hashes from leaves to root on the right side of tree since the insertion process appends new node to the last node of the tree; so we just need h hash value to be recalculated since h is the height of the tree. Therefore, the construction time complexity is $\log_2(n)$.

3. Verification time complexity

   Verification of the block integrity using Merkle tree requires rebuilding the whole tree again by computing the hash of every transaction in the block and constructing Merkle tree to reach Merkle root; then comparing the computed value with the Merkle root value in the block header (which is considered as reference value). If two values are equal, then the block data is valid, otherwise, it's not valid. This process requires n hash calls. Therefore, the full verification's complexity is $O(n)$.

   We don't need the whole Merkle tree to verify the integrity of a certain transaction in a block; we just need several hashes to do so. It's enough to have number of hashes equal to the tree height. Therefore, the verification time complexity is $O(h)$ which is $O(\log_2(n))$.

4. Space complexity

   To store Merkle tree for n nodes, we simply need to store n hash values. Therefore, the space complexity is $O(n)$, which is considered a large space.

5. Merkle tree issues

   There are some disadvantages in Merkle tree which make it not optimal simulation for blockchain:

— The cost of Merkle tree construction is high, especially when we have a lot of transactions in block (in average, 1626 transactions per block in Bitcoin process during October 2021)[1].
— Blockchain networks don't store Merkle tree. Therefore, full nodes have to construct it and provide Merkle branch every time transaction verification is requested.

[1] Blockchain Explorer — Search the Blockchain | BTC | ETH | BCH, October 2021. https://www.blockchain.com/charts/n-transactions-per-block (accessed: 01.11.2021).

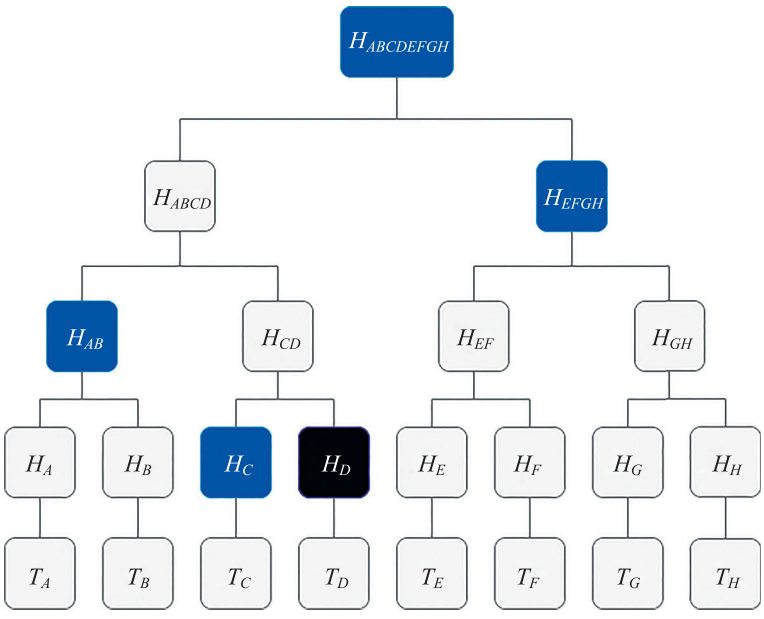

Fig. 5. Verification on transaction "T_D"

Moreover, it's time consuming to rebuild it again for every request.

— It creates unnecessary network traffic because it's required to transfer Merkle branch to verify one transaction. L network accesses are required to verify one block transactions. If we assume that m users are requesting the verification the same block, then it causes enormous number of network access. Therefore, the complexity of verifying one block (one Merkle tree) is $O(mL\log_2 n)$ that affects network performance.

As a result, Merkle tree has many significant features that make it fundamental for blockchain. However, it has costly data structure and it needs a lot of resources to be constructed. It doesn't require the whole hash tree to be downloaded in order to verify a block of data, instead it requires just a few hash values (Merkle branch) to be able to verify data authenticity, but still, it takes time and resources and needs computing these hashes and also cause unnecessary network traffic. Therefore, we need more efficient mechanism to verify data integrity and authenticity which require less computing and time complexity.

Probabilistic filter

Probabilistic filters are high-speed, space-efficient data structures that support approximate membership tests with a one-sided error. These filters can claim that a given entry is definitely not represented in a set of entries or might be represented in the set. Therefore, negative responses are conclusive, whereas positive responses incur a small false-positive probability (it might sometimes indicate that an entry is a member of the represented set although it is not). One of the most famous filters is **Bloom filter** which is represented with m bit array.

Some of Bloom filter properties are:

— It allows for membership check in constant space and time.

— Very infrequently it will give a false-positive answer, implies it will say YES if the answer is NO (probably in the set).

— It will never give false-negative answer, implies it will never say NO if the answer is YES (definitely not in the set).

— Basic Bloom filter supports two operations: test and add.

— It has constant time complexity for both adding items and asking whether a key is present or not.

— You can't remove an item from a Bloom filter.

— It also requires much less space compared to the number of items you need to store and check.

We can insert an element in this filter by inserting 1 into k slots in the bit array by k hash functions. When we want to check if a certain element is member in this filter or not, we must check all k slots if they have 1 value; if not, it means that this element is definitely not a member of the filter.

Fig. 6 shows an example of a Bloom filter with $m = 18$ bit array and $k = 3$ hash functions, representing the set $\{x, y, z\}$. The colored arrows show the positions in the bit array that each set element is mapped to. The element w is not in the set $\{x, y, z\}$ because it hashes to one bit-array position containing 0.

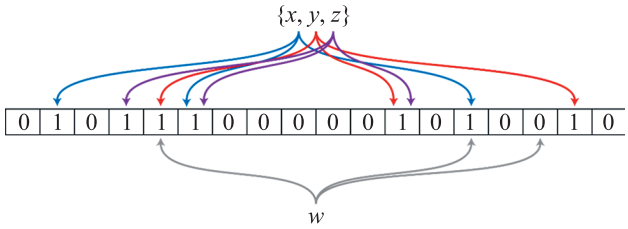

Fig. 6. Bloom filter presentation

Let f is the false-positive rate; n is the number of inserted items; k is the number of hash functions; and m is the number of bits in the filter. The following equation determines the false-positive rate as a function of other three parameters [18].

$$f \approx \left(1 - e^{-\frac{nk}{m}}\right)^k. \quad (1)$$

The following equation determines the optimal number of hash functions:

$$k_{opt} = \frac{m}{n}\ln 2. \quad (2)$$

The following equation determines the array size in bits for a given input (n) and desired false-positive rate:

$$m = -\frac{n\ln f}{(\ln 2)^2}. \quad (3)$$

We should carefully choose these parameters to optimize the desired Bloom filter. We can get a very low false-positive rate that can be neglected.

1. Construction

   Building cost for a Bloom filter for a block with L transactions is $O(Lk)$, we can consider $O(k) \approx O(1)$ because k is nearly constant.

2. Verification

   Verifying cost for a Bloom filter for a transaction in a block is $O(k)$, because we only need to check k bits in an array.

3. Space

   We just need m bits to store a Bloom filter.

Incremental hash chain

We proposed to use incremental hash chain to replace Merkle tree by providing more efficient approach to verify integrity of data which consume less time and computation power than Merkle tree.

Construction

Block is consisted of many transactions, and in order to construct an incremental hash chain for these transactions in order to verify that these transactions are correct and in the same order, we will start incrementally building our hash chain from the first transaction until the last one.

Let a block has L transactions; T_i presents the transaction with index i; $H(T_i)$ is the hash value of transaction T_i. To calculate the hash value for T_i, we need the transaction concatenated with the hash of previous transaction (except the first transaction), and it's given by equation

$$H(T_i) = \begin{cases} H(T_1);\ i = 1 \\ H(H(T_{i-1})\|T_i);\ 1 < i \le L \end{cases}. \quad (4)$$

The last hash value of equation (4) is the root of the incremental hash value; we call it Incremental Hash Root (INR). This value will be stored in the block header and transmitted to all nodes to be used in the verification process.

Definition 1. IHR for a certain block contains L transactions in the incremental hash value of the last transaction of that block, and it's given by the following equation

$$IHR = H(T_n).$$

Equation (4) demonstrates that all transactions are linked together incrementally, and any change to the data or order of transactions will lead to a wrong IHR.

Total hash function calls in a block contain L transactions, i.e. exactly L calls. Therefore, construction time complexity is $O(L)$, while we need $2L–1$ calls in binary Merkle tree. Our approach downsizes computation time to half, so it uses more efficient mechanism and saves power and time.

Verification

In our proposed approach we can verify the block data by recomputing IHR again. This process provides overall verification for block data. But when it comes to verify a single transaction in a block, we cannot use our approach to achieve that. Our approach is missing one significant feature that Merkle tree provides, it has partial verification. Therefore, we integrate incremental hash with Bloom filter to achieve this feature.

Incremental Hash Chain Bloom Filter-based

Merkle tree is costly data structure, and it requires a lot of computations when a blockchain requests a transaction in a block from a full node. The full node should build a Merkle tree for this block and send the transaction with the Merkle branch to the light node. Moreover, this process will be repeated for every single transaction in a block that required a lot of computations and network traffic. We proposed a new mechanism using Incremental Hash Chain Bloom filter-based to verify transactions.

Miner node side

Mining nodes are only responsible for creating blocks to add to the blockchain, let's consider a miner builds a new block containing L transactions. He starts with computing hash values for the transactions using equation (4) to generate IHR to be added to the block header. In our approach, it's required to build the Bloom filter for this block by inserting the hash values of the transactions into the filter array, then to compute the hash value of the constructed Bloom filter (let's call this value **BFH**) and to add the resulting hash value to the block header to be used later by light nodes in the verification process for checking the integrity of received Bloom filter from a full node.

Full node side

In our approach, block header contains IHR and BFH. When a full node receives a new block from a miner, it will recompute the hash values for all transactions in the block to verify IHR value in the block header; then it will construct Bloom filter for the block and compute hash value for it and compare it with BFH value existing in the block header; and at last recompute the block hash to verify the block data.

When a light node requests a transaction, the full node sends the transaction data, the previous transaction hash, and the Bloom filter.

Light node side

A light node requests a transaction and receives the response from a full node. The response contains: transaction data, previous transaction hash and Bloom filter. The light node computes the hash for the Bloom filter and compares it with the BFH in the block header. If they don't match, the node rejects the transaction; if they match, then the node computes the hash of the transaction using equation (4) and gets the transaction hash. After that it checks if this hash value exists in the received filter: if it exists – the transaction is valid and verified, else the light node rejects the transaction.

Considerations

The main problem of our approach is that we rely on Probabilistic filter, which doesn't give us a deterministic result; but if we build our filter carefully with choosing the perfect parameters for the filter, we can get a very small false-positive probability which we can neglect.

By using the equations (1), (2) and (3), let us consider Bitcoin where in October 2021[1], in average, there were 1626 transactions per block. As for Ethereum, there were about 74 transactions in a block (during 2021). Let's say we want to present 2000 transaction in a Bloom filter (m = 2000), and we choose the size of the filter as 10 Kbit (n = 10 Kbit) and, applying the previous equations, we can find the results as shown in Fig. 7, *a*.

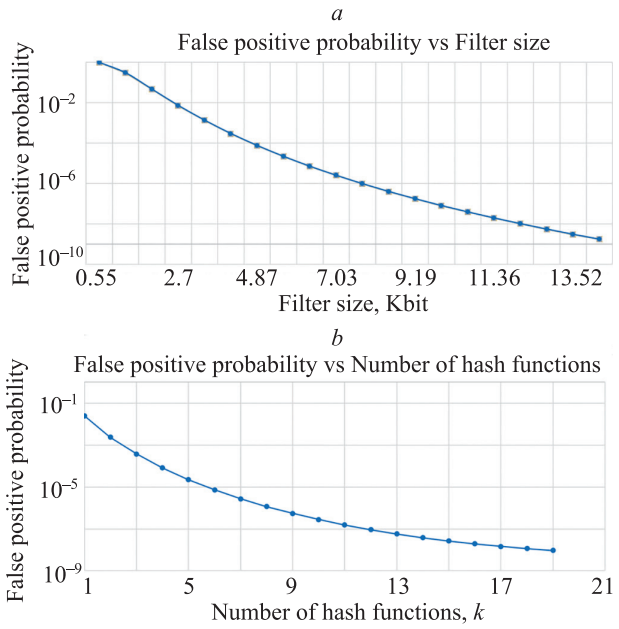

Fig. 7. Probability of false alarm depending on: filter size (*a*); the number of hash functions (*b*)

[1] Blockchain Explorer — Search the Blockchain | BTC | ETH | BCH, October 2021. https://www.blockchain.com/charts/n-transactions-per-block (accessed: 01.11.2021).

Table. Comparison between binary Merkle tree and our proposed incremental hash chain model

Feature	Binary Merkle tree	Our approach
Construction cost	$O(n)$	$O(L)$ for IHR $O(KL)$ for Bloom filter
Verification of a transaction	$O(\log_2 n)$	$O(K)$
Verification of all transactions	$O(n\log_2 n)$	$O(L)$
Space cost	$O(n)$	$O(L)$

The false-positive probability is $p = 0.000000982$ using 8 hash functions ($k = 8$), which is considered as neglected value.

We can see from (Fig. 7, *a*, *b*) that we can tune the number of hash functions used in Bloom filter and tune the size of the filter according to the size of the input, and we can get very small value for p which is neglected; also we can have dependence on the Bloom filter to verify the partial verification for a transaction in a block.

Analysis

Our proposed solution is much faster than binary Merkle tree as demonstrated in Table, it downsizes required computation resources to half. We need $2L–1$ hash calls to construct binary Merkle tree while we only need L calls in our solution; it saves more time and computing power. When it comes to verification process, our approach provides better performance in overall verification when we want to verify block data integrity. Moreover, when it comes to partial verification, our approach gives better performance using $O(K)$ calls to verify a single transaction using Bloom filter.

The big difference in our approach is that we need to build Bloom filter and send it to the light nodes which require transaction verification from this block. But this difficulty replaces the need to compute Merkle branch for every transaction request which is a very cost operation to do. But in our approach we just need to build the filter once and use it with all light nodes which gives better performance and computing size when we are dealing with blocks having big number of transactions (thousands), like Bitcoin does.

Collision resistance probability

One of the most import properties in any cryptographic hash mechanism is its resistance to collision which means getting the same hash value for two different inputs. When we want to choose a suitable and secure hash function to be used in our solution, we should consider its collision resistance. Let H is a hash function with m bits output. According to Birthday Paradox [19, 20], the expected hash collision with 50 % probability is accrued when we reach $\sqrt{2^m}$ outputs, for example. If we consider SHA256, since it is used in Bitcoin, we face collision with 50 % probability after getting 2^{128} outputs. This means that after getting 2^{128} hash values, we would start to get collisions in our system with 50 % probability. Therefore, we should consider the blockchain hash usage rate to be able to predict the required m-bit output hash function.

By studying Bitcoin as the first and the most used blockchain network, we find that in average Bitcoin minors generate $\approx 2^{91}$ hashes per year[1], so we need 2^{37} years with the same computing platform and power consumption. Therefore, we can safely consider that SHA256 is suitable to our system.

Disadvantage

In case there is a false-positive value in the Bloom filter, we cannot figure out where is the problem. In that case we have to request the whole block data (all transactions) from a full node, recompute the IHR and compare it with the IHR in the block header to check the integrity of the whole block data. All that requires a lot of computations and gives a bad performance. However, many recent researches prove that the false-positive probability is negligible [21], and a Bloom filter research with free zones is required [22]. Therefore, we can say that the probability is negligible.

Conclusion

In this article, we introduce a new approach to replace binary Merkle tree in blockchain by proposing more efficient model using incremental hash chain Bloom filter-based. Our solution consumes less computing power than binary Merkle tree. Moreover, it needs less time and space to construct the model and verify the integrity of blockchain data. We compare our mode with binary Merkle tree and prove that our model is more efficient in every aspect.

[1] Blockchain Explorer — Search the Blockchain | BTC | ETH | BCH, October 2021. https://www.blockchain.com/charts/n-transactions-per-block (accessed: 01.11.2021).

References

1. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008, pp. 21260.
2. Lamport L. Password authentication with insecure communication. *Communications of the ACM*, 1981, vol. 24, no. 11, pp. 770–772. https://doi.org/10.1145/358790.358797
3. Merkle R.C. A digital signature based on a conventional encryption function. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1988, vol. 293, pp. 369–378. https://doi.org/10.1007/3-540-48184-2_32
4. Wang S., Ouyang L., Yuan Y., Ni X., Han X., Wang F.-Y. Blockchain-enabled smart contracts: architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019, vol. 49, no. 11, pp. 2266–2277. https://doi.org/10.1109/TSMC.2019.2895123
5. Das K., Bera B., Saha S., Kumar N., You I., Chao H.-C. AI-envisioned blockchain-enabled signature-based key management scheme for industrial cyber-physical systems. *IEEE Internet of Things Journal*, 2022, vol. 9, no. 9, pp. 6374–6388. https://doi.org/10.1109/JIOT.2021.3109314
6. Sharma P., Jindal R., Borah M.D. Blockchain technology for cloud storage: A systematic literature review. *ACM Computing Surveys*, 2020, vol. 53, no. 4, pp. 3403954. https://doi.org/10.1145/3403954
7. Hariharasitaraman S., Balakannan S.P. A dynamic data security mechanism based on position aware Merkle tree for health rehabilitation services over cloud. *Journal of Ambient Intelligence and Humanized Computing*, 2019, in press. https://doi.org/10.1007/s12652-019-01412-0
8. Alzubi J.A. Blockchain-based Lamport Merkle Digital Signature: Authentication tool in IoT healthcare. *Computer Communications*, 2021, vol. 170, pp. 200–208. https://doi.org/10.1016/j.comcom.2021.02.002
9. Dhumwad S., Sukhadeve M., Naik C., Manjunath K.N., Prabhu S. A peer to peer money transfer using SHA256 and Merkle tree. *Proc. of the 23rd Annual International Conference in Advanced Computing and Communications (ADCOM)*, 2017, pp. 40–43. https://doi.org/10.1109/ADCOM.2017.00013
10. Zhang D., Le J., Mu N., Liao X. An anonymous off-blockchain micropayments scheme for cryptocurrencies in the real world. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020, vol. 50, no. 1, pp. 32–42. https://doi.org/10.1109/TSMC.2018.2884289
11. Ojetunde B., Shibata N., Gao J. Secure payment system utilizing MANET for disaster areas. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019, vol. 49, no. 12, pp. 2651–2663. https://doi.org/10.1109/TSMC.2017.2752203
12. Zhou Z., Wang B., Dong M., Ota K. Secure and efficient vehicle-to-grid energy trading in cyber physical systems: Integration of blockchain and edge computing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020, vol. 50, no. 1, pp. 43–57. https://doi.org/10.1109/TSMC.2019.2896323
13. Mao J., Zhang Y., Li P., Li T., Wu Q., Liu J. A position-aware Merkle tree for dynamic cloud data integrity verification. *Soft Computing*, 2017, vol. 21, no. 8, pp. 2151–2164. https://doi.org/10.1007/s00500-015-1918-8
14. Li H., Lu R., Zhou L., Yang B., Shen X. An efficient Merkle-tree-based authentication scheme for smart grid. *IEEE Systems Journal*, 2014, vol. 8, no. 2, pp. 655–663. https://doi.org/10.1109/JSYST.2013.2271537
15. Jakobsson M., Leighton T., Micali S., Szydlo M. Fractal Merkle tree representation and traversal. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003, vol. 2612, pp. 314–326. https://doi.org/10.1007/3-540-36563-X_21
16. Buchmann J., Dahmen E., Schneider M. Merkle tree traversal revisited. Lecture Notes in Computer Science (including subseries *Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5299, pp. 63–78. https://doi.org/10.1007/978-3-540-88403-3_5
17. Chelladurai U., Pandian S. HARE: A new hash-based authenticated reliable and efficient Modified Merkle Tree data structure to ensure integrity of data in the healthcare systems. *Journal of Ambient Intelligence and Humanized Computing*, 2021, in press. https://doi.org/10.1007/s12652-021-03085-0

Литература

1. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system // Decentralized Business Review. 2008. P. 21260.
2. Lamport L. Password authentication with insecure communication // Communications of the ACM. 1981. V. 24. N 11. P. 770–772. https://doi.org/10.1145/358790.358797
3. Merkle R.C. A digital signature based on a conventional encryption function // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 1988. V. 293. P. 369–378. https://doi.org/10.1007/3-540-48184-2_32
4. Wang S., Ouyang L., Yuan Y., Ni X., Han X., Wang F.-Y. Blockchain-enabled smart contracts: architecture, applications, and future trends // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2019. V. 49. N 11. P. 2266–2277. https://doi.org/10.1109/TSMC.2019.2895123
5. Das K., Bera B., Saha S., Kumar N., You I., Chao H.-C. AI-envisioned blockchain-enabled signature-based key management scheme for industrial cyber-physical systems // IEEE Internet of Things Journal. 2022. V. 9. N 9. P. 6374–6388. https://doi.org/10.1109/JIOT.2021.3109314
6. Sharma P., Jindal R., Borah M.D. Blockchain technology for cloud storage: A systematic literature review // ACM Computing Surveys. 2020. V. 53. N 4. P. 3403954. https://doi.org/10.1145/3403954
7. Hariharasitaraman S., Balakannan S.P. A dynamic data security mechanism based on position aware Merkle tree for health rehabilitation services over cloud // Journal of Ambient Intelligence and Humanized Computing. 2019. in press. https://doi.org/10.1007/s12652-019-01412-0
8. Alzubi J.A. Blockchain-based Lamport Merkle Digital Signature: Authentication tool in IoT healthcare // Computer Communications. 2021. V. 170. P. 200–208. https://doi.org/10.1016/j.comcom.2021.02.002
9. Dhumwad S., Sukhadeve M., Naik C., Manjunath K.N., Prabhu S. A peer to peer money transfer using SHA256 and Merkle tree // Proc. of the 23rd Annual International Conference in Advanced Computing and Communications (ADCOM). 2017. P. 40–43. https://doi.org/10.1109/ADCOM.2017.00013
10. Zhang D., Le J., Mu N., Liao X. An anonymous off-blockchain micropayments scheme for cryptocurrencies in the real world // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2020. V. 50. N 1. P. 32–42. https://doi.org/10.1109/TSMC.2018.2884289
11. Ojetunde B., Shibata N., Gao J. Secure payment system utilizing MANET for disaster areas // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2019. V. 49. N 12. P. 2651–2663. https://doi.org/10.1109/TSMC.2017.2752203
12. Zhou Z., Wang B., Dong M., Ota K. Secure and efficient vehicle-to-grid energy trading in cyber physical systems: Integration of blockchain and edge computing // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2020. V. 50. N 1. P. 43–57. https://doi.org/10.1109/TSMC.2019.2896323
13. Mao J., Zhang Y., Li P., Li T., Wu Q., Liu J. A position-aware Merkle tree for dynamic cloud data integrity verification // Soft Computing. 2017. V. 21. N 8. P. 2151–2164. https://doi.org/10.1007/s00500-015-1918-8
14. Li H., Lu R., Zhou L., Yang B., Shen X. An efficient Merkle-tree-based authentication scheme for smart grid // IEEE Systems Journal. 2014. V. 8. N 2. P. 655–663. https://doi.org/10.1109/JSYST.2013.2271537
15. Jakobsson M., Leighton T., Micali S., Szydlo M. Fractal Merkle tree representation and traversal // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2003. V. 2612. P. 314–326. https://doi.org/10.1007/3-540-36563-X_21
16. Buchmann J., Dahmen E., Schneider M. Merkle tree traversal revisited // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2008. V. 5299. P. 63–78. https://doi.org/10.1007/978-3-540-88403-3_5
17. Chelladurai U., Pandian S. HARE: A new hash-based authenticated reliable and efficient Modified Merkle Tree data structure to ensure integrity of data in the healthcare systems // Journal of Ambient Intelligence and Humanized Computing. 2021. in press. https://doi.org/10.1007/s12652-021-03085-0

18. Luo L., Guo D., Ma R.T.B., Rottenstreich O., Luo X. Optimizing bloom filter: Challenges, solutions, and comparisons. *IEEE Communications Surveys and Tutorials*, 2019, vol. 21, no. 2, pp. 1912–1949. https://doi.org/10.1109/COMST.2018.2889329
19. Suzuki K., Tonien D., Kurosawa K., Toyota K. Birthday paradox for multi-collisions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006, vol. 4296, pp. 29–40. https://doi.org/10.1007/11927587_5
20. Gilbert H., Handschuh H. Security analysis of SHA-256 and sisters. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2004, vol. 3006, pp. 175–193. https://doi.org/10.1007/978-3-540-24654-1_13
21. Lee D., Park N. Blockchain based privacy preserving multimedia intelligent video surveillance using secure Merkle tree. *Multimedia Tools and Applications*, 2021, vol. 80, no. 26-27, pp. 34517–34534. https://doi.org/10.1007/s11042-020-08776-y
22. Kiss S.Z., Hosszu É., Tapolcai J., Rónyai L., Rottenstreich O. Bloom filter with a false positive free zone. *IEEE Transactions on Network and Service Management*, 2021, vol. 18, no. 2, pp. 2334–2349. https://doi.org/10.1109/TNSM.2021.3059075

18. Luo L., Guo D., Ma R.T.B., Rottenstreich O., Luo X. Optimizing bloom filter: Challenges, solutions, and comparisons // IEEE Communications Surveys and Tutorials. 2019. V. 21. N 2. P. 1912–1949. https://doi.org/10.1109/COMST.2018.2889329
19. Suzuki K., Tonien D., Kurosawa K., Toyota K. Birthday paradox for multi-collisions // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2006. V. 4296. P. 29–40. https://doi.org/10.1007/11927587_5
20. Gilbert H., Handschuh H. Security analysis of SHA-256 and sisters // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2004. V. 3006. P. 175–193. https://doi.org/10.1007/978-3-540-24654-1_13
21. Lee D., Park N. Blockchain based privacy preserving multimedia intelligent video surveillance using secure Merkle tree // Multimedia Tools and Applications. 2021. V. 80. N 26-27. P. 34517–34534. https://doi.org/10.1007/s11042-020-08776-y
22. Kiss S.Z., Hosszu É., Tapolcai J., Rónyai L., Rottenstreich O. Bloom filter with a false positive free zone // IEEE Transactions on Network and Service Management. 2021. V. 18. N 2. P. 2334–2349. https://doi.org/10.1109/TNSM.2021.3059075

Authors

Maher A. Maalla — Student, ITMO University, Saint Petersburg, 197101, Russian Federation, https://orcid.org/0000-0002-4806-8608, maher.malla7@gmail.com
Sergey V. Bezzateev — D. Sc., Associate Professor, Professor, ITMO University, Saint Petersburg, 197101, Russian Federation; Saint Petersburg State University of Aerospace Instrumentation, Head of Department, Saint Petersburg, 190000, Russian Federation, SC 6602425996, https://orcid.org/0000-0002-0924-6221, bsv@aanet.ru

Авторы

Маалла Махер Аднан — студент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, https://orcid.org/0000-0002-4806-8608, maher.malla7@gmail.com
Беззатеев Сергей Валентинович — доктор технических наук, доцент, профессор, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация; заведующий кафедрой, Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, 190000, Российская Федерация, SC 6602425996, https://orcid.org/0000-0002-0924-6221, bsv@aanet.ru

Received 09.02.2022
Approved after reviewing 24.03.2022
Accepted 15.05.2022

Статья поступила в редакцию 09.02.2022
Одобрена после рецензирования 24.03.2022
Принята к печати 15.05.2022