Authenticating Account Balances in the Bitcoin Ecosystem via Merkle Trees

Zichen He

Faculty of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, 710049, China

HeZichen_1@stu.xjtu.edu.cn

Abstract. Bitcoin, as the trailblazing cryptocurrency, has profoundly impacted global markets and stands as a formidable contender to traditional financial paradigms. Yet, even within the sophisticated framework of Bitcoin, there remains potential for enhancement. Within the Bitcoin ecosystem, lightweight nodes operate without storing all transaction details. Instead, they retain only the block headers' content and specific transactional data pertinent to their own operations. To authenticate a transaction's presence in the blockchain, these lightweight nodes seek a Merkle proof from the full nodes. However, a challenge arises when lightweight nodes aim to validate the accuracy of their account balances sourced from full nodes. The latter relies on a data structure known as the Unspent Transaction Outputs (UTXO) for streamlined balance computation. In contrast, lightweight nodes grapple with ascertaining the veracity of such balance calculations. A viable solution lies in leveraging the available space in every block's coinbase transaction, which permits arbitrary modifications. By arranging the UTXO within a Merkle tree structure and embedding its root into the coinbase transaction's available space, account balance verification for lightweight nodes can be significantly enhanced using the Merkle proof. While such modifications to the Bitcoin protocol might trigger soft forks, the risk of hard forks remains absent.

Keywords: Bitcoin, Account balances, Merkle proof, Coinbase transaction, UTXO.

1. Introduction

In 2008, a mysterious figure known as Satoshi Nakamoto unveiled the Bitcoin white paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" on the P2P Foundation website, heralding the revolutionary concept of Bitcoin [1]. With its distinct features of decentralization, anonymity, security, and stability, Bitcoin rapidly gained global traction, experiencing both meteoric rises in value and notable volatility. Through ups and downs, Bitcoin has maintained its position as a leading cryptocurrency.

Within the Bitcoin system, users have several options to manage their Bitcoin holdings, such as the Bitcoin core software, various centralized institutions, or simplified payment verification (SPV) wallets [2]. The Bitcoin core system is a deterministic software that establishes a full node within the Bitcoin network. These full nodes house the entire blockchain, encompassing all block headers and block bodies with transactional details. On the other hand, SPV wallets function as lightweight nodes, possessing only block header information and select transaction specifics. As running a full node incurs considerable resources, users desiring mere Bitcoin management on mobile devices often gravitate towards SPV wallet applications. For SPV wallet users looking to verify the existence and legitimacy of a particular transaction on the blockchain, a Merkle proof becomes indispensable. By requesting requisite data from the Bitcoin network's full nodes, users can swiftly confirm a transaction's validity and presence. Yet, challenges arise when users aim to retrieve account balances from full nodes. Typically, they receive the result without a straightforward method for verification. Beyond storing the entire blockchain, full nodes also retain the Unspent Transaction Output set, encompassing all yet-to-be-spent transaction outputs. These full nodes can efficiently compute the account balance associated with a user's address (public key) without delving into every transaction. However, potential vulnerabilities exist: if a full node harbors malicious intent, it might dispense inaccurate data. For instance, a compromised full node might offer all unspent transaction outputs linked to a user's address, along with the account balances for the lightweight node, but discreetly

alter an output value. Under these circumstances, the lightweight node operator remains oblivious to the discrepancy, receiving erroneous balance information.

This paper introduces an innovative approach harnessing the available space within a coinbase transaction to counteract this challenge. Typically, every transaction comprises an input script and an output script for validation [3]. This verification entails matching the payer's public key with their digital signature. In coinbase transactions, however, the input script becomes redundant for verification, granting flexibility to alter its content. This available space approximates 100 bytes [4]. By structuring the UTXO set in the form of a Merkle tree, akin to standard transactions, the tree's root can nestle within the coinbase transaction's free space. Leveraging both the UTXO Merkle tree and the transaction Merkle tree, the Merkle proof undergoes enhancements, bolstering the verification accuracy of account balances. However, realizing this solution necessitates modifications to the existing Bitcoin protocol, potentially triggering a soft fork in the blockchain.

2. Underlying Background

2.1. Block Header and Block Body

Blockchain represents a data structure or system comprised of a sequence of blocks. These blocks are interconnected through hash pointers, creating a secure, append-only ledger [5]. Each Bitcoin block is structured into two primary components: a block header and a block body. With a size of 80 bytes, the block header encapsulates pivotal information about the block. Contained within the block header are the version number, timestamp, difficulty target, a random nonce, a hash pointer to the antecedent block, and the Merkle tree root. The difficulty target offers an encoded rendition of the actual target threshold, playing a pivotal role in steering the mining process. The random nonce stands as a variable element that miners can adjust to ensure the block header's hash remains below the set target. Meanwhile, the hash pointer present in the block header alludes to the hash value of the preceding block's header, a mechanism that fortifies the blockchain's integrity.

Contrastingly, the block body employs the Merkle tree data structure to chronicle transactional details transpiring within the designated block mining interval. These transactions find a permanent record within the blockchain, earning its distinction as the distributed ledger. Given its role, the block body demands a substantially larger storage allocation compared to the block header. A more detailed exposition on the Merkle tree is provided in section 2.3.

2.2. Full Nodes vs. Lightweight Nodes

In the Bitcoin ecosystem, a notable redundancy exists in data storage. Many nodes often store the entirety of blocks within the blockchain, leading to considerable storage space consumption. For those who aim to establish their blockchain node for project purposes without engaging in mining, synchronizing this extensive data can be an arduous task, both in time and resources.

To circumvent this challenge, Bitcoin distinguishes nodes into two types: full nodes and lightweight nodes. Full nodes house an exhaustive copy of the blockchain on their systems, encompassing both block headers and block bodies. Furthermore, they maintain a specific data structure referred to as Unspent Transaction Outputs. Within the Bitcoin framework, full nodes are truly comprehensive entities. Their responsibilities range from verifying the authenticity of every transaction to determining which transactions integrate into the candidate block. Their mandate also includes validating the legitimacy of new blocks by checking the proof of work and the integrity of every transaction housed within. Their involvement extends to mining processes as a means to gain incentives. They are instrumental in tracing the longest valid chain and spotting forks. Their presence fortifies decentralization, bolstering the security of the entire blockchain network.

Conversely, lightweight nodes are not burdened with storing all transaction details. Their retention scope is limited to block headers and select transaction specifics pertinent to their operations. Through the employment of Merkle trees, these nodes can ascertain the inclusion of a transaction in the existing blockchain transaction roster. However, they are not equipped to confirm the longest valid chain.

Furthermore, their capabilities are limited when it comes to validating the majority of transactions or affirming the accuracy of blocks disseminated by the blockchain network. Simplified Payment Verification wallets stand as examples of lightweight nodes.

2.3. Merkle Trees and Merkle Proofs

The Merkle tree, conceptualized by Ralph Merkle, boasts a wide range of applications across various domains, chiefly reducing data transmission needs and streamlining computational intricacies [6]. Because of its intricate structure, the Merkle tree can be harnessed to execute Simplified Payment Verification (SPV) in Bitcoin, ensuring efficient transaction validation [7]. Within the Bitcoin framework, all transactions reside in the block body, effectively forming a Merkle tree. This tree is a complete binary structure, interconnected by hash pointers. The leaf nodes of the Merkle tree encompass data blocks housing transaction details. Every parent node emerges from the combination of the SHA-256 hash of its two child nodes. This entails hashing the child nodes and concatenating the subsequent results to compute parent nodes. This methodology is reiterated layer by layer in an ascending manner. Given that Merkle tree operations in blockchain mandate an even number of leaf nodes, a block with an odd number of transactions sees the last transaction duplicated to achieve evenness. Notably, intermediate hashes aren't retained; they merely serve the computational procedure. Full nodes solely archive all transactions, or the leaf nodes. Conclusively, the root of the Merkle tree's hash finds its place in the block header, often referred to as the Merkle root.

Lightweight nodes, in contrast, aren't tasked with discerning the longest valid chain or authenticating the veracity of auxiliary transactions. Their sole function is to confirm the inclusion of pertinent transactions into the blockchain, a process underpinned by Merkle proofs. To illustrate, consider Alice (A) transferring three bitcoins to Bob (B). Upon notifying Bob of the transaction, she provides the transaction's ID, fundamentally the hash of the entire transaction, akin to a block's hash. Now, aiming to validate the transaction's incorporation into the blockchain and its integrity (for instance, verifying that the transferred amount mirrors Alice's declaration), Bob's lightweight node (an SPV wallet on his device) directs a verification request to a full node. This request encompasses the block header information and the hash value (ID) of the intended transaction. It's pivotal to recognize that the lightweight node doesn't archive the transaction; the objective lies in retrieving genuine transaction data. Subsequently, the full node assembles a Merkle tree. During this construction phase, the full node organizes the transactions in their block appearance sequence, mirroring the order of Merkle tree leaf nodes. Thereafter, the full node computes the hash value for each Merkle tree node, initiating from the leaves and progressing to the root.

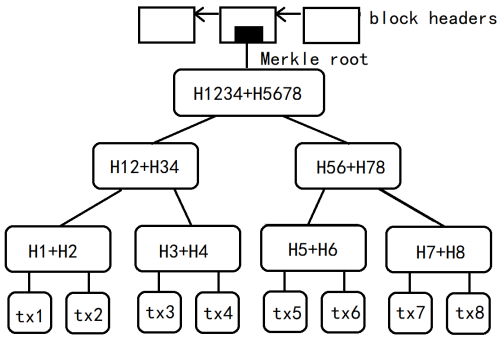

Fig. 1 Merkle root (Photo/Picture credit: Original).

As shown in Fig. 1, assume this is the Merkle tree the full node constructs and tx3 represent the transaction to be verified. Note that H1+H2 represents two hash values (H(tx1) and H(tx2)) are concatenated as one node. Since the hash value (ID) of this transaction is known, it can be directly located in the Merkle tree. Specifically, the order of leaf nodes corresponds to the order of transactions in the block, allowing for the direct identification of the transaction's position in the Merkle tree. The full node will provide the lightweight node with two components: 1) all the details of the transaction

to be verified (tx3) and 2) the hash values of H4, H12 and H5678. Actually, the full node provides a Merkle path (or Merkle proof), which is essentially a sequence or list [tx3, H4, H12, H5678].

Following this list sequentially, the lightweight node computes H3=hash(tx3), H34=hash(H3+H4), H1234=hash(H12+H34), and finally, the Merkle root=hash (H1234, H5678). The lightweight node then compares this calculated Merkle root with the Merkle root stored in the block header. If they match, it indicates that the transaction exists in the block and is unaltered.

2.4. Unspent Transaction Output Set

In Bitcoin system, Full nodes maintain a data structure known as Unspent Transaction Outputs, which represents transaction outputs that have not been spent yet. UTXO model and account/balance model are two popular transaction record models [8]. The former record the account balances of users. In contrast, the account balances should be calculated by inputs and outputs of transactions in the latter used in Bitcoin.

The UTXO set is stored in the memory of full nodes, while the entire blockchain is stored on their hard drives. Each element in the UTXO set includes transaction hash, output index, amount of Bitcoin and locking script. Transaction hash is the pointer to a specific transaction in the blockchain containing the unspent output [9]. Output Index indicates the position of the output within the output list of the corresponding transaction. Since a single transaction can have multiple outputs, relying solely on the transaction hash pointer is insufficient to pinpoint the location of the current output. Therefore, the output index field is essential. The combination of the transaction hash and output index uniquely determines the location of the current unspent output within the blockchain. Locking script is associated with the recipient's address or public key and defines who has the authority to spend the UTXO.

The UTXO set can be leveraged to swiftly detect whether a transaction is susceptible to double spending attacks. Furthermore, it can be utilized to calculate account balances effectively, instead of query all transactions [10].

2.5. Free Space in Coinbase Transaction

The coinbase transaction is the first transaction in every block and is intended to give an incentive to the miner [4]. Every transaction has both an input script and an output script, which respectively specify the source and destination of the Bitcoin involved. During the verification process, these scripts play a crucial role. However, in the case of coinbase transactions, the source of the Bitcoin is definite as it represents the reward to miners. Consequently, miners can modify the input script field at will, without any impact on the verification process. Miners can even write down their moods during mining in this free space.

The free space is about 100 bytes [4]. In practice, part of it has already been utilized. In order to meet the increased mining difficulty, the 4-byte nonce field is often not sufficient. Many people have started using some bytes of the coinbase transaction's input as additional random values. Changing the content of the coinbase transaction will change the Merkle root of the block header, and ultimately change the overall hash of the block header. This operation expands the search space, preventing miners from tried a new candidate block.

3. Methodology

3.1. Merkle Proof with UTXO Merkle Tree

The free space in coinbase transaction is much bigger than the space required to expand the search space as the supplement of nonce in the block header. This paper proposes a mechanism to fully utilize the free space and help lightweight nodes to verify account balances effectively.

Lightweight nodes can easily verify the existence and validity of a transaction due to the presence of Merkle tree and Merkle proof. However, the verification of account balances, which is based on the UTXO set, lacks a similar mechanism. To address this problem, this paper proposes to organize

the UTXO set into a Merkle tree, similar to the structure used for transactions. The root of this UTXO Merkle tree can then be included in the free space in coinbase transaction. Since the coinbase transaction is also stored within the original Merkle tree, any changes to it will affect the value of the original Merkle root in the block header. This means that the Merkle proof can be utilized to verify whether an element in the UTXO set has been modified.

The process of organizing the UTXO set into a Merkle tree is similar to that of transactions. Each leaf node represents an element of the UTXO set. Each parent node is calculated by hashing the values of its two child nodes and concatenating the results. The hash of the UTXO Merkle tree's root is included int the free space of the coinbase transaction.

If a lightweight wishes to obtain its account balance from a full node in Bitcoin network and verify the validity, it can request an enhanced Merkle proof. What differs from before is that this improved Merkle proof depends on two Merkle trees: the original transaction Merkle tree and the new UTXO Merkle tree. And the verification process mainly consists of two layers of Merkle proof verification. The inner layer verifies the UTXO Merkle tree, while the outer layer verifies the transaction Merkle tree.

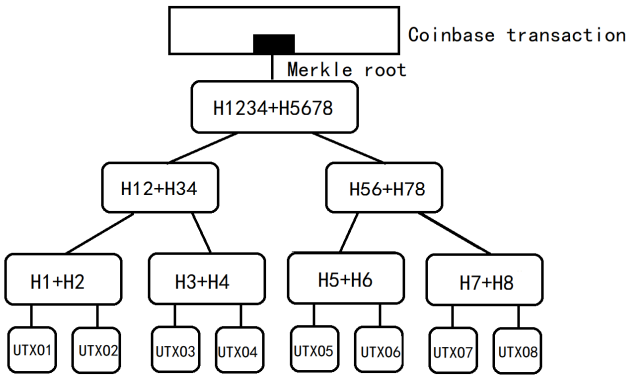

Fig. 2 UTXO Merkle tree (Photo/Picture credit: Original).

The lightweight node sends its account address (public key) to a full node in the Bitcoin network, and the full node returns the necessary information. Fig. 2 illustrates a simplified representation of a Merkle tree composed of all UTXO set elements, with the leaf nodes representing unspent outputs. Assume that UTXO3 and UTXO8 are the outputs corresponding to the current lightweight node's account. The full node provides the details of these two outputs, including transaction hash, output index, amount of Bitcoin and locking script. According to the details, the lightweight node can calculate the account balances easily. Additionally, the full node provides the values of H4, H7, H12 and H56. Lightweight node can calculate other required values as follows: H3=hash (UTXO3), H8=hash (UTXO8), H34=hash (H3+H4), H78=hash (H7, H8), H1234=hash(H12+H34), H5678=H (56) +H (78). Finally, the root is equal to hash(H1234+H5678). In this scenario, the full node should return more detailed Merkle path rather than a simple sequential list. The lightweight node can calculate each required value sequentially based on the provided Merkle path. Furthermore, the full node should provide the content of the coinbase transaction, because the lightweight node does not store it. The free space in the coinbase transaction now includes the newly introduced UTXO Merkle tree root. The lightweight node then compares the calculated results with this provided content. If they match, the inner-layer verification is completed, and the process proceeds to the outer-layer verification.

The outer-layer verification is essentially the same as the transaction verification using transaction Merkle trees, as specified in the original Bitcoin protocol mentioned in section 2.3. The transaction to be verified through Merkle proof is the coinbase transaction. Upon completion of the outer-layer verification, it means that the calculated Merkle root matches the Merkle root stored by the lightweight node.

If both the inner-layer and outer-layer verification are completed, it indicates that the unspent outputs provided by the full node for the corresponding account are valid and legitimate, and the calculated account balance is also valid and legitimate.

3.2. Forking Problem

The improvement of the Merkle proof to make the verification of account balances easy by the application of the UTXO Merkle tree means the modification of the Bitcoin protocol. When the Bitcoin protocol undergoes an update, and more than 50 percent of the network's hashing power has upgraded to the new protocol version, there is a possibility of a protocol fork occurring.

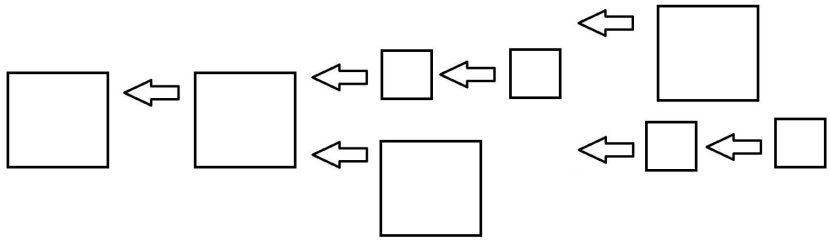

Fig. 3 Soft forks (Photo/Picture credit: Original).

Protocol forks can be categorized as either hard forks or soft forks. In a soft fork, the new version of the protocol is not recognized by the old version nodes, while the old version nodes can still accept both old and new blocks. Soft forks are temporary if it is supported by over 50 percent of the network's hashing power. As shown in Fig. 3, assume that the big blocks represent old blocks, while the small blocks represent new blocks. In theory, the new blocks should not accept the old blocks, but when the first block that has upgraded to the new protocol is mined, it must accept the previous chain. Afterward, the new blocks will no longer accept the old blocks, and the chain will extend only with new blocks. The old blocks can accept both new and old blocks, which can lead to various chain extensions, as shown in the diagram. However, as the hashing power of the new version nodes exceeds 50 percent, the old blocks will be extended along the new chain. But since the new blocks do not recognize the old blocks, they are overwritten each time, leading to a soft fork. This type of soft fork is not persistent because the new blocks always form a new chain, and the old blocks are consistently overwritten, resulting in a loss of block rewards. Thus, the soft fork motivates the old version nodes to upgrade their protocols to participate in block reward generation. Therefore, this soft fork will not cause serious harm to the Bitcoin system.

In the suggested protocol enhancement, the input field of the coinbase transaction undergoes modification, aligning with the stipulations for a soft fork as detailed above. Nodes operating on the newer version will disregard blocks that haven't undergone alterations to the coinbase transaction. Conversely, nodes on the older version can recognize blocks originating from the newer version. This implies that if nodes accounting for more than 50% of the total computational power adopt the protocol modification, it will lead to a soft fork, manifesting as a transient divergence.

4. Conclusion

This paper delves into foundational elements of the Bitcoin system, covering topics such as block headers, light nodes, the Merkle tree with its associated Merkle proof, the UTXO set, and coinbase transactions. Stemming from this foundational knowledge, challenges encountered by light nodes during account balance verification are illuminated. Following this exposition, potential remediations are put forward. Specifically, structuring the UTXO set into a Merkle tree and anchoring its root within the available space of the coinbase transaction paves the way for account balance verification via Merkle proofs. Remarkably, this solution bypasses the need for intricate methodologies, demanding mere rudimentary amendments to the existing Bitcoin protocol. Its efficiency stands out, ensuring light nodes are not burdened with hefty local data storage or with undertaking cumbersome

computations. Moreover, the volume of data relayed to light nodes by their full node counterparts remains within Bitcoin network's permissible parameters. However, a caveat arises: this proposition may overlook certain practical nuances inherent to the Bitcoin marketplace. In the operational landscape, altering the Bitcoin protocol emerges as a daunting challenge, compelling agreement from nodes commanding over half the computational might. Still, as a theoretical construct, this methodology could illuminate pathways for crafting future blockchain frameworks.

References

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin.

[2] Kaushal, P. K., Bagga, A., & Sobti, R. (2017). Evolution of bitcoin and security risk in Bitcoin Wallets. 2017 International Conference on Computer, Communications and Electronics (Comptelix).

[3] Zheng, P., Luo, X., & Zheng, Z. (2023). BSHUNTER: Detecting and tracing defects of bitcoin scripts. 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).

[4] Jeon, K., Lee, J., Kim, B., & Kim, J. J. (2023). Hardware accelerated Reusable Merkle tree generation for bitcoin blockchain headers. IEEE Computer Architecture Letters, 22(2), 69–72.

[5] Bruen, A. A., Forcinito, M. A., & McQuillan, J. M. (2021). Cryptography, information theory, and error-correction: A handbook for the 21st Century (2nd ed.). John Wiley & Sons, Inc. pp.549-560.

[6] Jing, S., Zheng, X., & Chen, Z. (2021). Review and investigation of merkle tree's technical principles and related application fields. 2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA).

[7] Liu, H., Luo, X., Liu, H., & Xia, X. (2021). Merkle tree: A fundamental component of blockchains. 2021 International Conference on Electronic Information Engineering and Computer Science (EIECS).

[8] Dai, W., Wang, Q., Wang, Z., Lin, X., Zou, D., & Jin, H. (2021). Trustzone-based secure lightweight wallet for Hyperledger Fabric. Journal of Parallel and Distributed Computing, 149, 66–75.

[9] Hu, K., Zhang, Z., & Guo, K. (2019). Breaking the binding: Attacks on the Merkle approach to prove liabilities and its applications. Computers & Security, 87, 101585.

[10] Bailey, B., & Sankagiri, S. (2021). Merkle trees optimized for stateless clients in bitcoin. In Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25 (pp. 451-466). Springer Berlin Heidelberg.