Revisiting Bitcoin's Merkle Tree Security: Practical Implications and an Attack on Core Chain

Anonymous

Abstract. This paper revisits a flaw in the Merkle tree construction underlying Bitcoin's light client security model. Although Bitcoin was innovative in introducing Proof of Work and simple payment verification (SPV), a design shortcoming in its Merkle tree structure can be exploited to prove the inclusion of a malicious or nonexistent transaction in a block. We investigate how this fault remains not only theoretical but can compromise the security of real-world systems. In particular, we examine Core Chain (an EVM-compatible blockchain using delegated Proof of Work from Bitcoin miners), which relies on a Bitcoin light client for consensus. We show that an attacker can exploit the Merkle tree flaw to forge proofs of mining power delegation. By removing all mining power from legitimate validators, the entire consensus process can be disrupted. Furthermore, we outline potential mitigations and best practices for light-client developers, and emphasize why those do not suffice, and necessary changes within Bitcoin itself are required to eliminate this vulnerability.

Keywords: Blockchain · Security · Bitcoin · Merkle Tree · Light Client

1 Introduction

Bitcoin [15] introduced the first decentralized and permissionless cryptocurrency system, leveraging Proof of Work (PoW) to secure the network. Its key innovation lies in a protocol that allows participants to agree on a global state without central authorities. One of the fundamental ideas presented in the Bitcoin whitepaper is the concept of a *Simple Payment Verification* (SPV) client - often referred to as a *light client*. By verifying the work in block headers while using cryptographic Merkle proofs to check the inclusion of specific transactions, light clients can verify the inclusion of transactions in the Bitcoin network using a constant amount of external information and no trust.

Despite its remarkable design, Bitcoin has certain known but often understated shortcomings. Among these is a subtlety in the Merkle tree construction used to compute a block's Merkle root. In principle, to compute the Merkle root, each transaction is hashed at the leaves of the Merkle tree, and internal nodes are generated by concatenating and hashing their child nodes. However, Bitcoin does not strictly enforce that the leaves' data format (transaction structure) differs from the internal nodes. As a result, it is possible to interpret the value of an internal node as if it were a leaf, a valid 64-byte transaction, enabling malicious "inclusion" [13].

Although this issue has been publicly disclosed, there remains limited awareness of its real-world implications and many practical implementations may still be at risk. In this work, we demonstrate how the Core Chain consensus mechanism can be exploited, **risking more than $500M** at the time of reporting [7], to illustrate that the vulnerability persists in certain systems, highlighting the need for a more thorough examination of the Merkle tree construction at the Bitcoin protocol level.

Core Chain uses special form of a Proof of Stake (PoS) for its consensus mecahnism. In a standard PoS system, validators lock up funds ("stake") to receive voting power in the network. The validators can then use their voting power to participate in the chain's consensus and prevent censorship. When the validators act corecctly, they recieve rewards, which incentivize them to participate honestly. Otherwise, if the validators act incorrectly or malicously, their locked up funds are taken away from them in a process called "slashing".

Core Chain extends this protocol with *Delegated Proof of Work*. The network allows validators to increase their voting power by delegating (assigning) mined Bitcoin blocks to their Core Chain's address. This is a form of *restaking*, a mechanism in which you can use the same resources to secure more than one application. Core Chain uses a Bitcoin SPV client to verify the delegation, making it a high value target for an attacker that looks to disrupt the network.

Responsible Disclosure. After discovering this vulnerability within the Core Chain delegated PoW mechanism, we conducted responsible disclosure through the Immunefi bug bounty platform [1]. Both Immunefi and the Core team later agreed to allow the publication of the findings. All findings are up to the date of reporting the issue on March 2024.

1.1 Our Contributions

In this work, we make the following contributions:

- We review the Merkle tree flaw in Bitcoin's design in a practical setting, and clarify why it allows adversaries to prove the inclusion of non-existent or manipulated transactions.
- We show a novel attack on the *Core Chain* blockchain that exploits this design. We show how an attacker can exploit Core Chain's reliance on Bitcoin's light client to remove everyone's voting power from the system, by carefully forging Merkle proofs of non-existant transactions.
- We revisit possible mitigations for the issue and discuss where they fail in practice, arguing in favor of a Bitcoin fork.

1.2 Related Works

The SPV client security issue emerging from the Bitcoin Merkle tree design was discussed in prior works and online discussions [13,16,5,2], previously affecting

Bitcoin Core (assigned CVE-2017-12842 [4], not to be confused with Core Chain) and Keep Network [17].

Additionally, a different vulnerability with the Bitcoin Merkle tree design was previously found and patched at the protocol level (assigned CVE-2012-2459 [3]). This issue was due to padding the block with extra transactions in case the number of transactions was odd, making the real block indistinguishable from the padded block.

Our attack on Core Chain is a form of exploiting restaking mechanisms. Another perspective on the subject is how the delegated PoW affect the economics of the network and wether this can be exploited as well (for example, with enough similar networks, a small portion of the compute power on the Bitcoin network could provide unproportional aggregate voting power across the other networks). The theory of economic security of restaking networks has been explored in [11,6,12].

1.3 Outline

We begin by reviewing Bitcoin's Merkle tree design and its flaw in the next section. In Section 3, we then show our attack on Core Chain that exploits this design flaw. We conclude in Section 4, where we discuss mitigations and next steps for the community.

2 Bitcoin's Design Flaw

The primary function of a light client (SPV client) in Bitcoin is to verify that a given transaction is included in a specific block *without* downloading the entire block. This is achieved using block headers and Merkle proofs [14].

2.1 What Is a Light Client?

A *light client* (or SPV client) differs from a full node in that it does not store or validate all transactions. Instead, it:

- **Downloads block headers:** The client obtains each block header, which includes the Merkle root of transactions and the nonce used in Proof of Work.
- **Verifies Proof of Work:** The client checks that the header hash is below the network's target, thereby confirming that miners expended computational effort.
- **Receives Merkle proofs:** For any transaction of interest, the client obtains a Merkle proof (the path in the Merkle tree from the leaf to the root).

By hashing the transaction with the intermediate nodes in the proof, the client can reconstruct the Merkle root and compare it against the root included in the verified block header. If they match, the transaction is deemed part of that block.

2.2 Why the Merkle Tree Design Is Faulty

In a *typical* Merkle tree, the leaves represent atomic data items (e.g., transactions), and each internal node is a hash of its children:

$$\text{parent} = \text{Hash}(\text{child1} \,||\, \text{child2}).$$

To provide a proof of inclusion for a leaf, you must provide the internal nodes adjacent to the path from the leaf to the root, as illustrated in Figure 1.

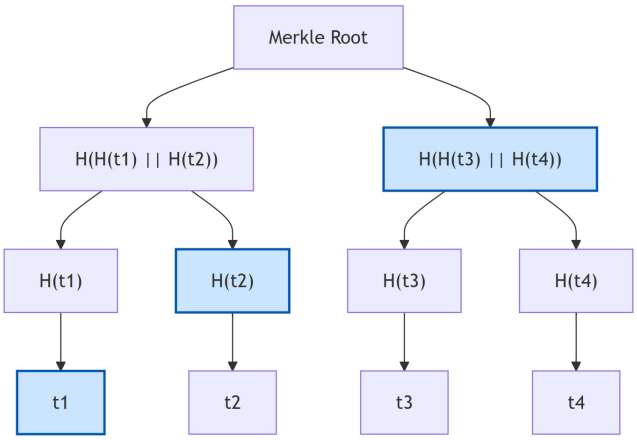

Fig. 1. A simplified Merkle tree representing four transactions (H represents a hash function). The shaded nodes are used as the proof for transaction $t1$.

In Bitcoin, the leaves are specifically the double-SHA256 hash of each transaction's serialized bytes. As we progress up the tree, the internal nodes are likewise double-SHA256 hashes of their children. Crucially, in *secure* Merkle tree usage, leaves are *distinguishable* from internal nodes: an internal node is, semantically, the hash of two children, while a leaf is the hash of a transaction.

64-Byte Transactions Are Valid However, Bitcoin does not *enforce* a difference in data format. A node that is a concatenation of hashes of two children can itself be interpreted as if it were a 64-byte "transaction", if the resulting data meets certain conditions.

Bitcoin sets no absolute minimum size for a transaction aside from a handful of rules that do not preclude a carefully crafted 64-byte object from being *interpreted* as a transaction. For instance, a valid Bitcoin transaction must contain:

- A 4-byte version field
- An input count and output count

– Each input (with a previous output reference, script length, script, sequence)
– Each output (with a value, script length, script)
– A 4-byte lock time

While many typical transactions are much larger, there is no built-in rule disallowing a transaction from being 64 bytes if these fields appear well-formed. Consequently, if internal node data can be massaged to *look* like 64 bytes of a transaction, it might slip past naive checks in an SPV client's Merkle verification code.

Since the Merkle root is computed over the entire tree, an attacker can craft a scenario in which:

– **They pick/create two real transactions** whose hashes line up in such a way that concatenating them forms a 64-byte blob that decodes as a valid transaction.
– **They demonstrate a Merkle path** that claims this 64-byte entity is included as a leaf, thus forging a transaction's inclusion that never actually appeared as an original leaf in the block.

Figure 2 illustrates how a valid proof can be crafted for an internal node, to prove its inclusion as an actual 64-byte transaction.

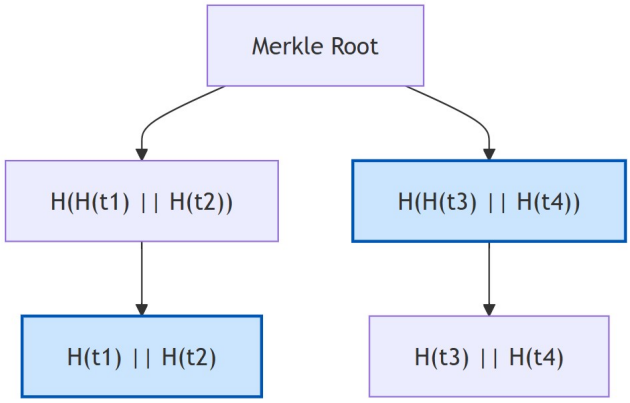

Fig. 2. Fake, but indistinguishable, view for the same Merkle tree from Figure 1. The shaded nodes are used as the proof for the (originally) internal node H (t1) || H (t2).

3 Attack on Core Chain

In this section, we focus on *Core Chain* [8] as a real-world example of an unsafe SPV-based design. Core Chain is an EVM-compatible blockchain that uses

delegated Proof of Work from Bitcoin miners, combined with Proof of Stake, to determine validator voting power.

Accurately computing voting power is essential for the correct execution of the network. The security of the funds being held on the network is determined by how hard it is to distrupt transaction processing and consensus on transaction inclusion. By having the wrong voting power, legitimate validators cannot enforce the correct behavior of the network, and other, perhaps malicious validators, can remove transactions, stop the network, and in the worst cases change the history of the chain and remove funds from wallets.

Our analysis shows how we can leverage the Bitcoin Merkle tree issue to cause the network to compute the voting powers incorrectly.

3.1 Core Chain Overview

Core Chain is a blockchain project advertising a *Satoshi Plus* consensus algorithm aimed at combining:

1. **Staking CORE**: Locking the native token of the chain.
2. **Mining Bitcoin**: Bitcoin miners can delegate the blocks they mine to validators in Core Chain, to increase their effective stake in the network.

To delegate a mined block to a Core validator, the Bitcoin miner includes a coinbase transaction (the first transaction in the block) that specifies the *Core validator address* in the `op_return` transaction field. A predefined amount of blocks is defined as an *epoch*, and the more blocks delegated to a single validator address in a single epoch, the more voting power this validtor has.

To see this, let the set of validators be V. A validator $v \in V$ on Core Chain will have a total voting power computed as follows:

Definition 1 (Voting Power). *Let C_v be the amount of native tokens staked for a validaotr $v \in V$, and let M_v be the amount of blocks delegated to it in the current Bitcoin epoch, as configured by the chain. The validaor has a* total voting power *of [10]:*

$$p_v = \rho M_v \sum_{u \in V} C_u + C_v \sum_{u \in V} M_u, \tag{1}$$

where ρ is a constant factor (configured by Core Chain) that determines how much voting power each delegated Bitcoin block provides.

We can see from this definition a useful artifact that we will make use of later:

Fact 1 *If* all *mining powers can be nullified ($M_v = 0$ for all $v \in V$), then* ***nobody has voting power****, even if they stake the native cryptocurrency CORE.*

3.2 SPV Verification on Core Chain

Core Chain must provide a secure way to verify block delegation from Bitcoin. Since these two chains operating independently, with the Bitcoin protocol even being unaware of the existance of Core Chain, this is tricky.

To do this, Core Chain uses an SPV client. By only storing Bitcoin headers on-chain instead of the transactions themselves, Core Chain avoids the massive storage overhead of full Bitcoin blocks. Instead, the system expects a Merkle proof that the coinbase transaction that delegates voting power to a validator, is indeed included in a Bitcoin block in the current epoch.

Our attack allows an attacker to forge Merkle proof for non-extant transactions, providing them with a way to delegate Bitcoin blocks to non-legitamate validators. To do this, the attacker needs to craft a special valid transaction and make sure it is included as the first transaction after the coinbase transaction in the Bitcoin block.

When crafting this transaction, the attacker must make sure that its hash concatenated with the real coinbase transaction hash is a possible serialization of another, non-existant transaction:

$$\text{Hash}(\text{coinbase\_tx} \,||\, \text{crafted\_tx}) = \text{Valid transaction serialization.}$$

To understand how to craft such transaction, we first need to understand how a Bitcoin transaction is deserialized by the Core Chain smart contract. Bitcoin transactions are serialized the following way:

- 4 bytes: `version`
- >1 bytes: `input count`
- Each input includes:
  - 32 bytes: previous output hash
  - 4 bytes: previous output index
  - >1 bytes: script length
  - (variable size) script
  - 4 bytes: sequence
- >1 bytes: `output count`
- Each input includes:
  - 8 bytes: Value
  - >1 bytes: script length
  - (variable size) script
- 4 bytes: `locktime`

The bug The bug in Core Chain that allows our attack is that there is minimal verification for the serialized transaction. If the transaction fails more complex verifications, the block is simply delegated to no one instead of raising an error.

Mainly, the verification is only that the lengths match the amount of data expected [9]. Specifically, we must provide at least one input and one output. By carefully setting input count, output count, and scripts length, we can produce a valid 64-byte structure that Core Chain accepts, without needing the content to

make sense. This means we only need to control a few key bytes in the serialized transaction – we can use a single input with a script length of 0 and a single output with a script length of 4 as a possible solution. We can verify that the resulted transaction length is indeed 64:

- [Total: 4] 4 bytes: `version`
- [Total: 5] 1 byte: `input count`
- [Total: 5] A single input:
  - [Total: 37] 32 bytes: previous output hash
  - [Total: 41] 4 bytes: previous output index
  - [Total: 42] 1 byte: script length
  - [Total: 42] 0 bytes: script
  - [Total: 46] 4 bytes: sequence
- [Total: 47] 1 byte: `output count`
- [Total: 47] A single output:
  - [Total: 55] 8 bytes: Value
  - [Total: 56] 1 bytes: script length
  - [Total: 60] 4 bytes: script
- [Total: 64] 4 bytes: `locktime`

This solution requires the attacker to control only 4 bytes (32 bits) in the serialized transaction – the input length, output length, and input and output script lengths. Thus, one out of ∼4B randomly crafted transactions would work for the exploit, easily enumerated by a computer.

Thus, the complete flow will be as following:

1. Randomly generate a valid Bitcoin transaction (e.g. by randomizing the `locktime`).
2. Compute the resulted serialized transaction, by hashing the crafted transaction and concatenating to the hash of the predicted coinbase transaction.
3. Repeat until the resulted hash has a single input with an empty script and a single output with 4 bytes of script.
4. Send the crafted transaction to Bitcoin, for it to be included as the first non-coinbase transaction.
5. Send the 64-byte hash as a coinbase transaction to Core Chain, delegating the block to no one.

Outcome The net effect of the exploit is to create a "fake coinbase transaction" that delegates voting power to no real validator. Repeating this for all blocks in the epoch will nullify the delegated proof of work from all validators, which as we saw in Eq. (1), means no validator will have any voting power.

Its important to note that the attack requires the attacker to accurately predict the contents of the coinbase transaction, which can be tricky - but it is not random and thus not cryptographically secure. Over a long period of time, it is within reason that an attacker will successfully execute this attack.

4 Discussion

A number of mitigations for the Bitcoin Merkle design issue have been proposed in prior work ([13]), many of which avoid modifying Bitcoin itself. One mitigation is to simply verify the transaction is serialized to more than 64 bytes in length. Given that valid 64-byte Bitcoin transactions are extremely rare under standard checks, a light client can safely assume fraud when encountering such data.

Another possible mitigation is to additionally include a proof for the rightmost transaction. If the number of transactions is not a power of two, the rightmost transaction is duplicated in the Merkle tree, thereby revealing the tree's actual depth. Yet, its hard to guarantee that the tree is never a perfect power-of-two without a protocol-level restriction.

While such non-forking solutions address immediate threats at the SPV client layer, they do not eradicate the underlying design flaw from the Bitcoin protocol. In principle, a future soft-fork or hard-fork could insert explicit markers or length constraints, ensuring that internal nodes can never be misconstrued as leaf nodes. However, any modification to Bitcoin's consensus layer is inherently contentious, and the ecosystem has historically been conservative about protocol changes.

Despite the hardships, we encourage to consider a fork as the only viable solution. As the ecosystem grows with newer projects allocating large capital, we cannot trust developers to be aware of all of Bitcoin's faults and mitigate them. This is true for Bitcoin in particular, since it is considered the most mature and secure blockchain by many, as evident by this specific vulnerability being found in multiple applications.

4.1 Conclusions

In this paper, we revisited a known but under-acknowledged flaw in Bitcoin's Merkle tree structure, showing how it can be leveraged to craft invalid transaction proofs in SPV contexts. By studying the real-world example of Core Chain, an EVM-compatible blockchain that delegates validator power based on Bitcoin blocks, we demonstrated a tangible exploit: forging coinbase inclusion to nullify all legitimate validator power.

The broader lesson is that cryptographic building blocks, while powerful, require careful attention to data formatting and structural constraints. As the industry continues to build multi-chain ecosystems, vigilance is needed to ensure that mistakes are fixed (especially in mature and trusted blockchains like Bitcoin), and do not threaten large-scale capital.

References

1. Immunefi, `https://immunefi.com/`
2. Merkle tree vulnerabilities. Bitcoin Optech, `https://bitcoinops.org/en/topics/merkle-tree-vulnerabilities/`, accessed: 2025-01-02
3. Cve-2012-2459. National Vulnerability Database (NVD) (2012), `https://nvd.nist.gov/vuln/detail/CVE-2012-2459`

4. Cve-2017-12842. National Vulnerability Database (NVD) (2017), https://nvd.nist.gov/vuln/detail/CVE-2017-12842
5. Weaknesses in bitcoin's merkle root construction. Bitcoin Dev Mailing List (2019), https://gnusha.org/pi/bitcoindev/CAFp6fsGtEm9p-ZQF_XqfqyQGzZK7BS2SNp2z680QBsJiFDraEA@mail.gmail.com/2-BitcoinMerkle.pdf, accessed: 2025-01-02
6. Chitra, T., Pai, M.: How much should you pay for restaking security? arXiv preprint arXiv:2408.00928 (2024)
7. CoinMarketCap: Core dao (core) price, market cap, and info (2025), https://coinmarketcap.com/currencies/core-dao/, accessed: 2025-01-02
8. CoreDAO: Core whitepaper, https://whitepaper.coredao.org/, accessed: 2025-01-02
9. CoreDAO: btcpowermirror github repository (2023), https://github.com/coredao-org/btcpowermirror/tree/0c9db0d9189a6cc8c4e519220945984a4d401d7d, commit 367adc3d
10. CoreDAO: Core genesis contracts github repository (2024), https://github.com/coredao-org/core-genesis-contract/tree/367adc3d9779cab7b27bc4922b16c00cdf42c859, commit 367adc3d
11. Dong, X., Litos, O.S.T., Tas, E.N., Tse, D., Woll, R.L., Yang, L., Yu, M.: Remote staking with economic safety. arXiv preprint arXiv:2408.01896 (2024)
12. Durvasula, N., Roughgarden, T.: Robust restaking networks. arXiv preprint arXiv:2407.21785 (2024)
13. Lerner, S.D.: Leaf node weakness in bitcoin merkle tree design (2018), https://bitslog.com/2018/06/09/leaf-node-weakness-in-bitcoin-merkle-tree-design/, accessed: 2025-01-02
14. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Conference on the theory and application of cryptographic techniques. pp. 369–378. Springer (1987)
15. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Satoshi Nakamoto (2008)
16. Todd, P.: Trusted merkle tree depth for safe tx inclusion proofs without a soft fork. Bitcoin Dev Mailing List (2018), https://gnusha.org/pi/bitcoindev/20180607171311.6qdjohfuuy3ufriv@petertodd.org/, accessed: 2025-01-02
17. Łukasz Zimnoch: Spv merkle proof malleability allows the maintainer to prove invalid transactions. GitHub Advisory Database (2024), https://github.com/advisories/GHSA-wg2x-rv86-mmpx, accessed: 2025-01-02