Extending the SPV contract concept with privacy gadgets

Oleksandr Kurbatov, Bohdan Skriabin, Anton Levochko, Dmitriy Zaharov

December 27, 2024

Abstract

The SPV contract allows for synchronizing the actual state of Bitcoin system to the designation system and uses it as a trustless source for cross-chain operations. Using the SPV contract, the user can prove that a particular transaction was confirmed in the Bitcoin mainnet and trigger the defined action in the destination system. Additionally, in this paper, we present the mechanism of untraceable but verifiable proofs of actions with defined pattern in the Bitcoin network.

1 Introduction

SPV (simple payment verification) is a method to verify Bitcoin transactions without running a full Bitcoin node, but only headers of blocks [BTC]. Each header equals 80 bytes, which makes the size of all confirmed block headers less than 70 megabytes in March of 2024. This method allows running a Bitcoin node on light devices like laptops, smartphones, etc. At the same time, the security of the mentioned approach stands the same as maintaining the full node. This paper describes the process of deploying the SPV node as a smart contract with lightweight verification architecture that allows the organization of the trustless source of Bitcoin events. Additionally, this work proposes some methods for reaching privacy while proving that some actions can be performed on the Bitcoin blockchain on the counterparty network.

It would be great to have SPV node as a smart contract for the whole infrastructure in general. That assumes implementing and running the node contract in number of destination systems. With such SPV contracts the users can prove that a particular transaction was confirmed in Bitcoin and trigger some defined actions. So, the contracts act as a trustless sources of Bitcoin events. Also calling SPV contract methods is more efficient than running own SPV node.

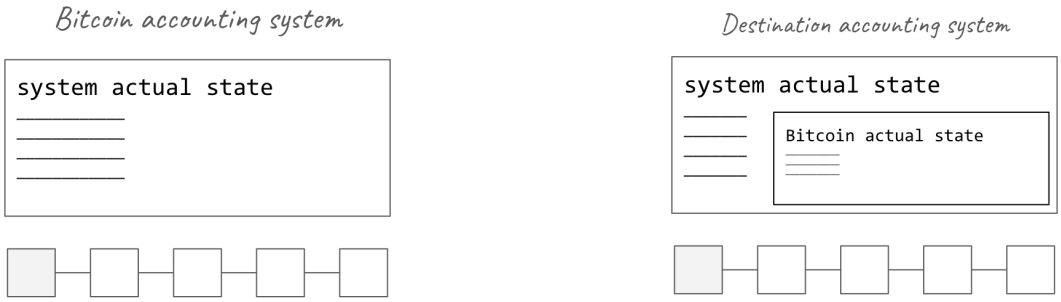

Figure 1: Concept overview.

Here are some features of SPV contract. Anyone can submit the Bitcoin block header to the contract and if it passes all the verifications, it will be saved in the contract and accessible for others. Anyone can reliably access data about any stored block header using the read methods with assurance of its correctness. One more important thing that users a free for setting custom confirmation parameter and it is highly recommended to use that parameter as big as possible even bigger than 6.

A number of different use cases could be covered with SPV contracts and they are basically about the followings:

- getting Bitcoin transaction statuses by other contracts;
- DAOs and their applications can just call read methods;
- cross transactions between destination system and Bitcoin;
- searching Bitcoin events with templates and using for triggers.

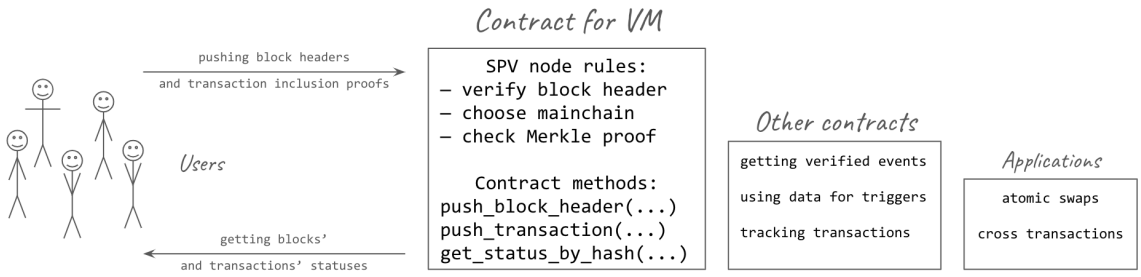

Figure 2: SPV contract overview.

2 Preliminaries

2.1 SPV node functioning principles

The SPV node concept was proposed in the original Bitcoin whitepaper [BTC]. This proposal allows the use of Bitcoin without maintaining the full node. By default, each block header includes the Merkle Root field—the special type hash of all block transactions. The SPV node does not synchronize full blocks from the network, only their headers and all Merkle Roots.

2.2 Block header validation rules

For transaction verification, the SPV node asks the full node to return the proof of inclusion of this transaction into the block. As proof, in this case, Merkle Branch is being used. Then, the SPV node performs the list of verifications to ensure the transaction is valid and is added to the chain. We can list these verifications as follows:

1. Block header verification:
   (a) Structure and existence of all fields:
      i. Version (4 bytes), Previous block (32 bytes), Merkle Root (32 bytes), Timestamp (4 bytes), Bits (4 bytes), Nonce (4 bytes), Tx count (1+ bytes).
   (b) Previous block hash
      i. Verifies that the block is part of the chain and refers to the existing previous block.
   (c) Timestamp
      i. Verifies if the timestamp value exceeds the median value of the previous 11 blocks. According to their clock, full nodes won't accept blocks with timestamps for more than two hours in the future.
   (d) Difficulty target
      i. The block's header double hash (SHA256) value must satisfy the defined difficulty parameter (must be less than the target value).

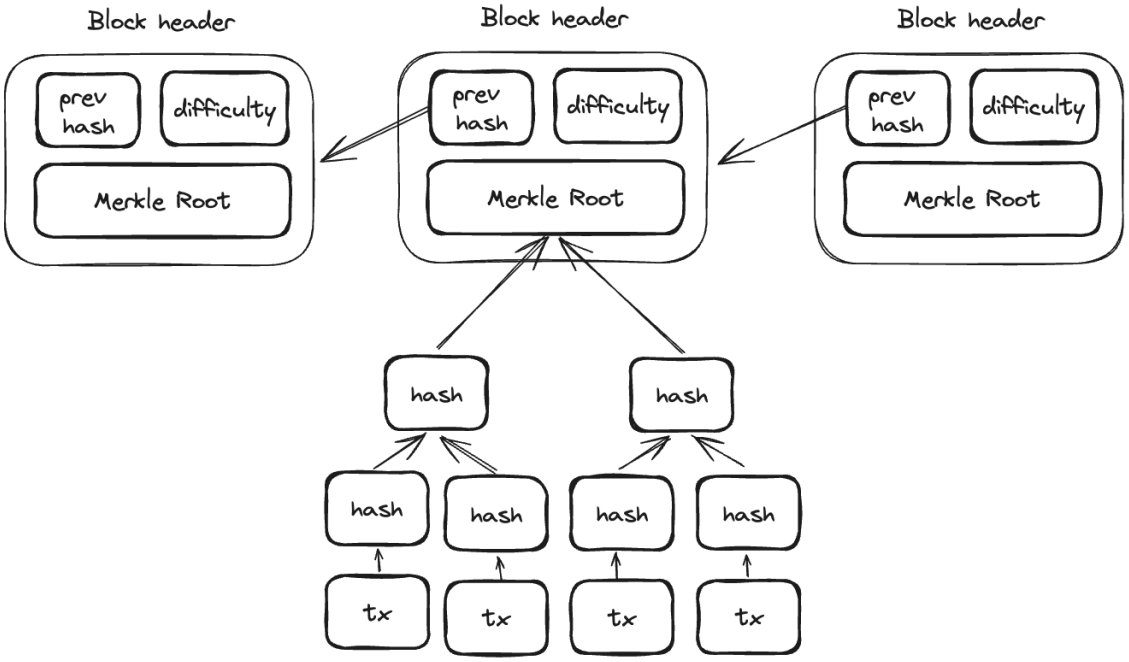

Figure 3: SPV node concept.

ii. The difficulty target parameter is changed each 2016 block to adjust the block mining time for the current network hash rate. It does so by summing up the total number of minutes miners took to mine the last 2,015 blocks [PoW], and it divides this number by the protocol's desired goal of 20,160 minutes (2,016 blocks x 10 minutes). The ratio is then multiplied by the current difficulty level to produce the new difficulty level.

iii. If the correction factor is greater than 4 (or less than 1/4), then 4 or 1/4 is used instead to prevent the change from being too abrupt.

(e) Nonce

i. The hash value of the concatenation of all previous parts of the header with the nonce value must be equal to the block hash value (satisfy difficulty target parameter).

2. Merkle Branch verification:

(a) When the SPV node receives the transaction with the Merkle Branch—it verifies that the path leads to the existing Merkle Root (defined in the mainchain block header)

3. Blocks count:

(a) SPV node must verify that the block is included in the mainchain (the heaviest chain), which means that a certain number of blocks was built on top of the provided block.

2.3 SPV contract implementations

Like the traditional SPV node, the SPV node smart contract acts as a repository for Bitcoin's block history. Anyone can submit the Bitcoin block header to the contract if it passes all the described verifications. At the same time, anyone can reliably access information about any stored block header using the contract's read method, with assurance of its accuracy.

Several implementations of the SPV contract, including Solidity and Rust, are presented in the bitcoin-SPV repository [SPV].

3 Proposed SPV contracts architecture

3.1 Confirmed and pending blocks

There is no finalization term for a Bitcoin transaction. It means that the history of Bitcoin can be reverted at any time (chain reorganization). But at the same time, users can define by themselves what they mean by the confirmed transaction (usually, if there are 6 consecutive blocks based on a particular one, we can count it as confirmed with a high probability).

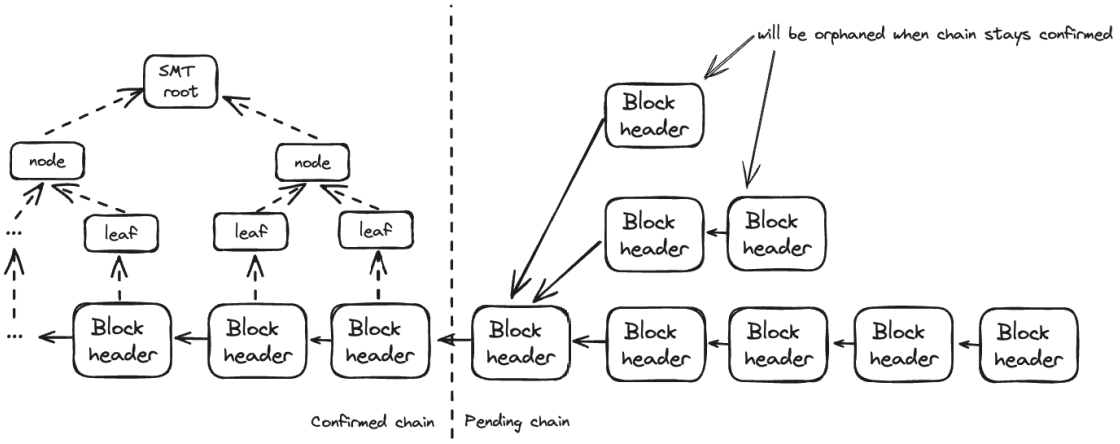

Figure 4: Confirmed and pending chains.

We propose the SPV contract architecture that allows setting this parameter n and defining the length of the *confirmed* and *pending* chain:

- Confirmed chain: the chain considered is final with a high probability. This chain can be reverted, but 1) random reorg can happen only in very rare cases (reorganization of the Bitcoin blockchain on the n last blocks); 2) the cost of the malicious reorg is very high [BFT].
- Pending chain: last n blocks in the chain (with a higher probability of reverting).

There are different mechanisms for processing confirmed and pending chains:

- Blocks of the confirmed chain represent the leaves of the State Merkle Tree (SMT). Based on the root of SMT, it's possible to prove that some transactions are included in the confirmed chain.
- Blocks of the pending chain are stored in the form of a cache. It's possible to have several alternative blocks for the same height in the pending chain; some will be orphaned when the mainchain is defined.

3.2 The mechanism of block header acceptance

As mentioned, anyone can propose a new block header to the SPV contract. We can present the mechanism of header acceptance in the following diagram:

3.3 Long reorganizations

As we discussed before, the chance for blockchain reorganization decreases with the adding new blocks but, at the same time, can't be equal to 0 despite the mainchain length. It means that there can be situations where users need to update the confirmed chain.

It's highly recommended to use the parameter n as big as possible while keeping an acceptable waiting time from the user's perspective. The owners of decentralized applications can regulate this parameter for their needs and balance the risk of double-spending attacks with users' comfort.

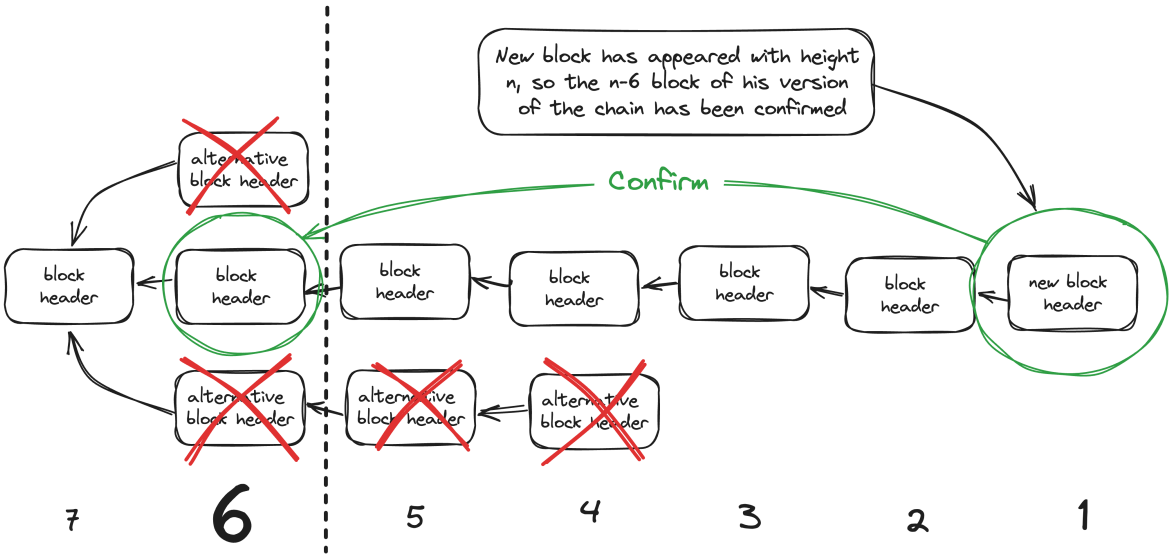

Figure 5: The process of adding a new header.

3.4 Mechanism of proving the Bitcoin transaction by the user

Having an SPV contract on the blockchain, the user can prove the existence of a particular transaction in Bitcoin. For this, the user needs to provide:

- The transaction itself;
- The transaction identifier;
- Block hash value;
- The Merkle Branch, which leads the transaction to the Merkle Root.

If the verification passes successfully—the user can trigger some action based on the defined logic. The SPV contract could be a universal socket for contract applications (with automatic verification).

3.4.1 Reduction in the cost of Merkle proof verification with zk-SNARK

It's possible to reduce the verification cost of proving transaction inclusion. Instead of providing the Merkle Branch as an inclusion proof, the user can generate the zk proof as follows:

- Private inputs: $MerkleBranch$.
- Public inputs: $transaction$, $blockhashvalue$, $MerkleRoot$.
- Proof logic: $MerklePath(hash(transaction), MerkleBranch) == MerkleRoot$.

In this case, instead of calculating the sequence of SHA256 hash values, the contract verifies the ZKP with the same logic.

4 Gadget for pattern-driven untraceable actions

An additional property that could be reached with the SPV contract is untraceability of the referred transaction. Imagine that the application's logic requires a transaction that burns 0.1 BTC, but the user doesn't want to reveal where exactly they did it. In this case, the user must prove:

1. There is a transaction in the Bitcoin network:

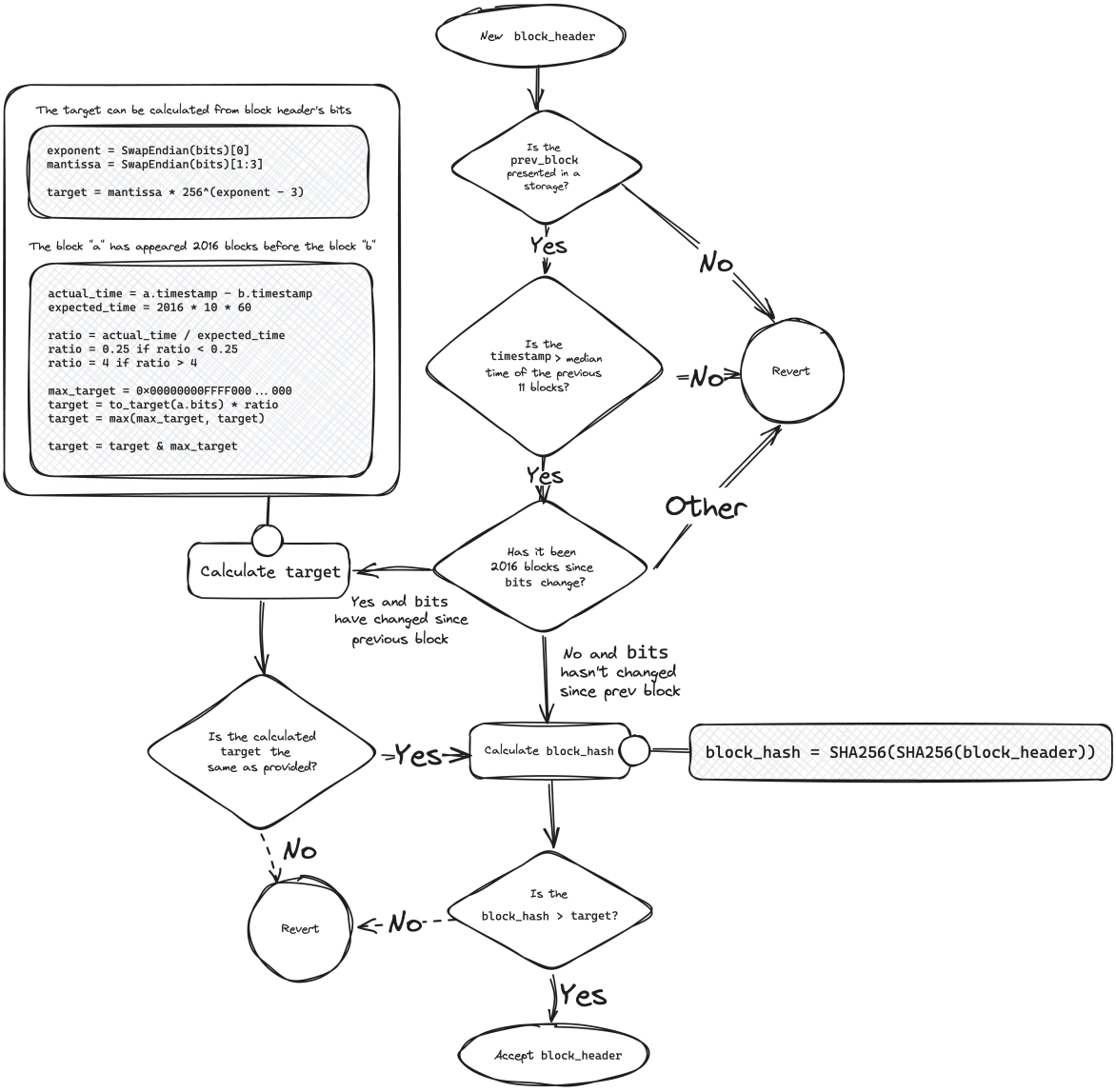

Figure 6: The process of verifying the block header by the SPV contract.

(a) Now, we can do it with the SPV node! Additionally, we want to achieve the privacy aspects of this verification.

2. The output of this transaction pays 0.1 BTC:

   (a) Can be ejected from the transaction data

3. The output of the transaction pays it to a non-spendable address (zero address, for example):

   (a) Can be ejected from the transaction data

4. Exactly the user controls the address from which the payment is performed:

   (a) This chapter is designated to resolve this issue.

5. Additionally, the user needs to prove the uniqueness of the transaction (double-spending protection).

We propose a protocol that allows us to make all these proofs private and guarantee the prevention of double-spending attacks. The core of the protocol is zk circuit that performs the following logic:

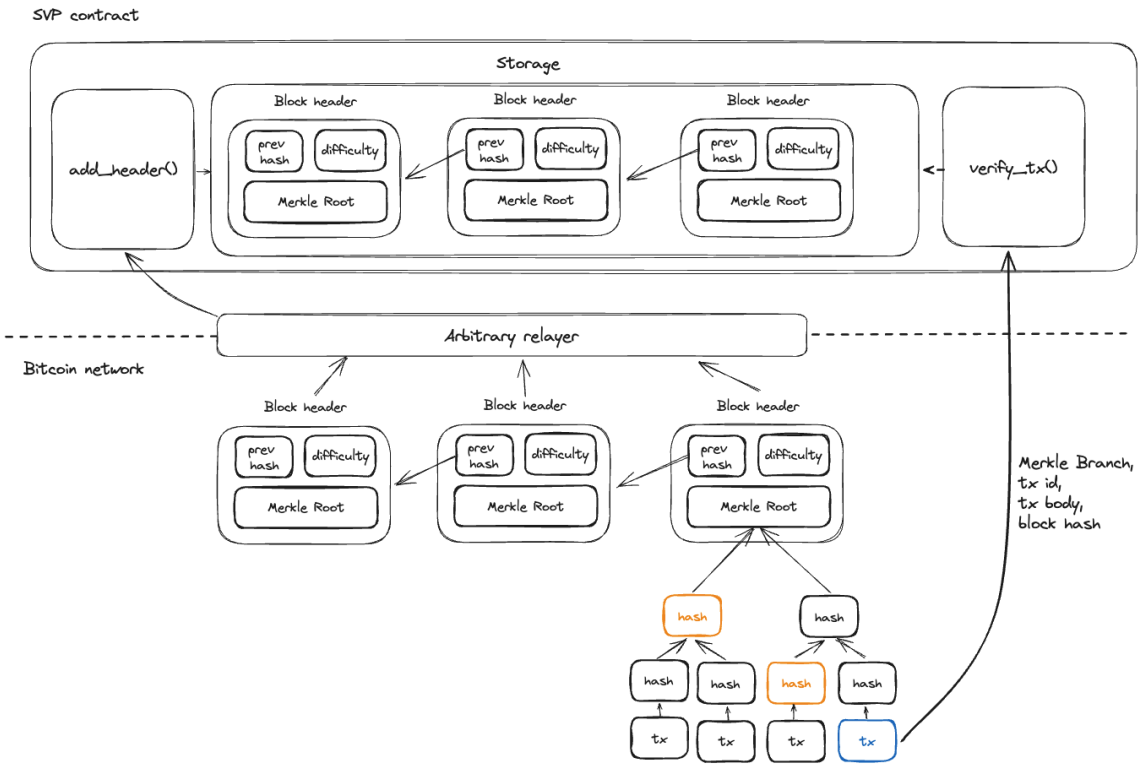

Figure 7: The process of transaction inclusion proving.

1. As public inputs (signals) for the circuit, we will use:
   (a) *amount* (the number of satoshis allocated in the transaction output). This data needs to be processed by the application logic (for example, issuing an equivalent amount of wrapped BTC based on a burn event);
   (b) $SMTroot$. Building the Merkle tree based on confirmed Bitcoin blocks is a public result.
   (c) *hash*. It's the zk-friendly hash value of the user's private key, used for spending coins. We will use this value as a nullifier—it uniquely binds to the private key (without compromising it) and prevents the same output from being used twice.
2. The list of private inputs is following:
   (a) *privatekey*. It's the key that was used for signing the Bitcoin transaction.
   (b) *transaction*. All fields of the Bitcoin transaction.
   (c) *previoustransaction*. The transaction, which includes the address generated based on the private key.
   (d) $MerkleBranch_{tx}(BlockRoot)$. Merkle Branch for proof of inclusion transaction in the block.
   (e) $MerkleBranch_{txprev}(BlockRoot)$. Merkle Branch for proof of inclusion of the previous transaction in the block.
   (f) $MerkleBranch_{tx}(SMTRoot)$. Merkle Branch for proof of block (with transaction) in the confirmed chain of blocks.
   (g) $MerkleBranch_{txprev}(SMTRoot)$. Merkle Branch for proof of block (with previous transaction) in the confirmed chain of blocks.
3. The user generates the proof which satisfies the following logic:
   (a) $hash(privkey) == hash$. The hash value of the private key equals the nullifier value.

(b) $address_gen(pkgen(priv_key)) == tx_prev.scryptPubKey$. The private key is valid and was used to generate the address.

(c) $MerklePathSMT(MerklePathBlock(hash(tx), MerkleBranchtx(BlockRoot)), MerkleBranchtx(SMTRoot)$ $SMTroot$. Transactions exist in the block that is part of a confirmed chain.

(d) $MerklePathSMT(MerklePathBlock(hash(txprev), MerkleBranchtxprev(BlockRoot)), MerkleBranchtxpre$ $SMTroot$. The previous transaction exists and is also included in the confirmed chain.

(e) $tx.amount == amount$. The number of coins in the output equals some defined value (more complex conditions, like "greater/lower than" could also be added).

(f) $tx.scriptPubkey == 0x0$. This equation is relevant for the burn transaction. In the same way, the user can prove the payment to any needed address (or to more complex conditions).

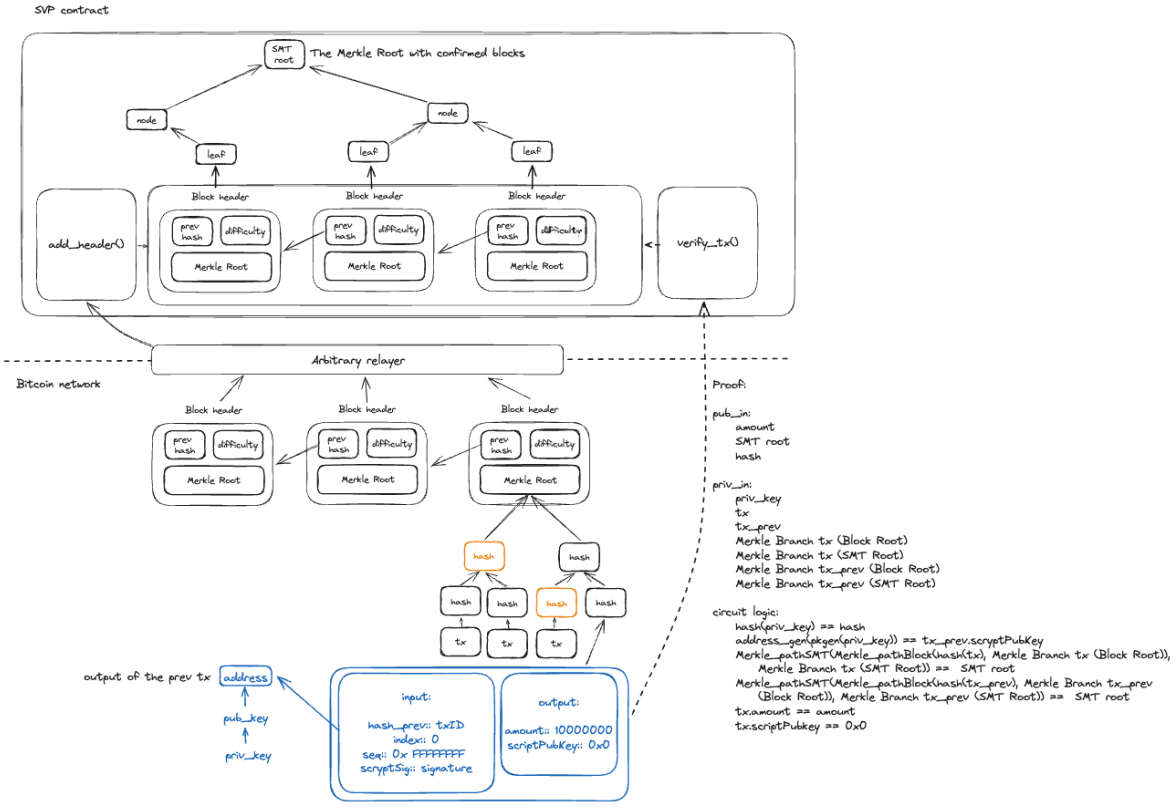

Figure 8: The process of transaction inclusion verifying.

5 Security claims

A few security issues and operating assumptions about utilizing SPV contract. Do not run SPV from genesis block to avoid huge amount of verifications in the beginning (initial synchronization). Create trustless snapshot in some point in the past and deploy SPV contract with it as predefined state.

The utilization of Simplified Payment Verification (SPV) contracts in Solidity offers a powerful means of interacting with external blockchain networks such as Bitcoin while maintaining a high degree of security and efficiency. However, to ensure the safety and reliability of SPV contracts, several security considerations and operating assumptions must be carefully addressed.

One critical security concern revolves around the initial synchronization process of SPV contracts. To avoid overwhelming the contract with a massive number of verifications during the genesis block, it is advisable not to run the SPV contract from the genesis block onwards. Instead, developers can opt to create a trustless snapshot of the blockchain at a specific point in the past, capturing relevant

state data, and deploying the SPV contract with this predefined state. This approach streamlines the synchronization process and mitigates the risk of performance bottlenecks during contract deployment.

Moreover, beyond the SPV contract itself, the establishment of a robust backend service is imperative for fetching specific data from the Bitcoin blockchain and providing it to the SPV contract. This backend service acts as a bridge between the Ethereum and Bitcoin networks, facilitating the transmission of relevant information while adhering to trustless principles. Importantly, developers should allocate resources for compensating the backend service providers, as they play a crucial role in executing verifications and maintaining the integrity of the SPV contract.

Furthermore, continuous monitoring and auditing of the SPV contract are essential to detect and address potential security vulnerabilities or discrepancies in data verification. Regular code reviews, security audits, and stress testing can help identify and mitigate potential risks, ensuring the robustness and reliability of the SPV contract over time.

In conclusion, while SPV contracts in Solidity offer a promising avenue for interacting with external blockchain networks, safeguarding their security requires careful attention to operating assumptions and security considerations. By implementing strategies such as avoiding initial synchronization from the genesis block, creating trustless snapshots, establishing reliable backend services, and conducting regular audits, developers can mitigate risks and enhance the safety and reliability of SPV contracts. Through diligent oversight and adherence to best practices, SPV contracts can serve as secure and efficient bridges between different blockchain ecosystems, unlocking new opportunities for decentralized finance and cross-chain interoperability.

6 Use cases and applications

Let explore the technological nuances and potential applications of using Bitcoin transaction statuses as triggers for conditions in smart contracts, alongside the creation of custom spending conditions and other tricky logic.

Smart contracts on EVM compatible platforms can access some particular Bitcoin transactions without centralized parties. This is primarily facilitated through Simple Payment Verification (SPV) contract that allow destination systems to verify Bitcoin transactions avoiding trusted sources of the data. This capability enables the development of cross-chain applications that react to events on the Bitcoin system and destination system simultaneously, offering a seamless interaction between.

That allows developers to create triggers in smart contracts based on specific activities on the Bitcoin system. These triggers can initiate actions on the destination platforms once a predefined Bitcoin transaction occurs. For instance, the release of escrowed funds, automatic contract execution, and other conditional operations can depend on Bitcoin transaction confirmations, making decentralized finance tools more interconnected.

Custom spending conditions within these smart contracts allow for the creation of sophisticated financial instruments. These conditions can govern how Bitcoin and other cryptocurrencies are spent, ensuring that certain criteria are met before transaction execution. This could include multi-signature approvals, compliance checks, or meeting specific on-chain events, thus enhancing the security and flexibility of cryptocurrency transactions.

Smart contracts enable the automation of regular payments using cryptocurrencies and digital assets. This functionality is crucial for subscription-based services, salary disbursements, or any periodic payment structure in the decentralized ecosystem.

The implementation of binary options for price predictions within smart contracts illustrates another innovative use case. These contracts can be set to pay out based on the future price of an asset, such as Bitcoin. Users can bet on whether the price of Bitcoin will rise above or fall below a certain point by a specific date, with the contract autonomously executing the payout based on the outcome.

Smart contracts can also be programmed to place bets on the future difficulty adjustments in Bitcoin. This type of derivative could be useful for validators and investors looking to hedge against fluctuations in mining difficulty or to speculate on the health and progression of the accounting system.

One of the challenges in creating reliable decentralized applications is the need for an undetermined and unpredictable source of entropy. Smart contracts that integrate Bitcoin transactions can use various attributes of these transactions, such as transaction IDs or block hashes, as on-chain sources of randomness, which are crucial for games, lotteries, and other applications requiring provable randomness.

A significant advantage of using SPV contracts is the minimal fee associated with calling read methods. Accessing Bitcoin transaction data through these methods incurs almost no cost, making it an economical choice for dApps that need-to-know Bitcoin transaction statuses in easiest way.

6.1 On-chain bridge

With SPV contract it is possible to read Bitcoin's events by Ethereum's contract. So, the contract can be sure that some assets are locked in Bitcoin and issue corresponding amounts of wrapped assets in Ethereum.

Here could be an option to burn wrapped assets in Ethereum automatically if the base assets were unlocked in Bitcoin. But this approach is not best in sense of user experience.

There is a point to think how to made such a bridge operating in two-ways. Nevertheless, that idea still actual since it assumes trustless on-chain bridge. It even can automatically track locking events in Bitcoin and issue wrapped tokens operatively.

In the ever-expanding landscape of blockchain ecosystems, interoperability between different networks has become a paramount concern. One intriguing avenue for achieving this interoperability is through a one-way bridge from Bitcoin to Ethereum, facilitated by a Solidity SPV (Simplified Payment Verification) contract. This innovative approach enables Ethereum smart contracts to ascertain events occurring on the Bitcoin network, thereby allowing the issuance of corresponding wrapped assets on Ethereum.

The essence of the SPV contract lies in its capability to extract and interpret data from Bitcoin's blockchain, ensuring that assets are securely locked within Bitcoin before issuing their equivalent on Ethereum. This process imbues the Ethereum contract with a high level of confidence regarding the validity and existence of the locked assets, fostering trust in the bridging mechanism.

While the concept of automatically burning wrapped assets on Ethereum upon unlocking the base assets in Bitcoin presents itself as a logical conclusion, it poses challenges in terms of user experience. Users may find it disconcerting to witness their assets vanish from Ethereum without a clear understanding of the underlying mechanisms. Therefore, alternative approaches that prioritize user understanding and control over asset management may be more desirable.

Moreover, the prospect of evolving such a bridge into a bidirectional mechanism warrants careful consideration. Despite the current focus on a one-way bridge, the concept of a trustless, on-chain bridge operating in both directions remains relevant. By harnessing the capabilities of SPV contracts, there is potential to automate the tracking of locking events in Bitcoin and the issuance of wrapped tokens on Ethereum in real-time. This bidirectional functionality would significantly enhance the utility and versatility of the bridge, opening up new avenues for decentralized finance (DeFi) applications and cross-chain asset transfers.

In conclusion, the utilization of Solidity SPV contracts offers a promising avenue for establishing a one-way bridge from Bitcoin to Ethereum. By leveraging Bitcoin's events and integrating them into Ethereum's ecosystem, this approach lays the groundwork for seamless asset transfer and interoperability between two of the most prominent blockchain networks. As the landscape of decentralized finance continues to evolve, the exploration of bidirectional bridging mechanisms remains pertinent, underscoring the ongoing quest for trustless and efficient cross-chain solutions.

6.2 Virtual escrow swaps

Counterparty that has digital assets in Ethereum and wants to exchange them for certain amount of assets issued in Bitcoin can make a request to Solidity Escrow Contract. Counterparty that has opposite exchange wish can found the order in such Ethereum Contract and lock Bitcoin's assets with certain conditions. Also providing the transaction to SPV Contract. Then Escrow Contract can ask SPV Contract about locking event in Bitcoin and automatically send Ethereum's assets to counterparty with opposite exchange wish. If every thing goes fine then counterparty with original order can unlock assets in Bitcoin.

In the realm of decentralized finance (DeFi), the concept of trustless escrow swaps presents a compelling solution for seamlessly exchanging digital assets between different blockchain networks. Leveraging Solidity Escrow Contracts in conjunction with SPV (Simplified Payment Verification) Contracts, users can engage in secure and efficient asset swaps between Bitcoin and Ethereum without relying on intermediaries or centralized platforms.

The process begins when a counterparty possessing digital assets in Ethereum expresses the desire to exchange them for a specific amount of assets issued in Bitcoin. This counterparty initiates a request to a Solidity Escrow Contract, detailing their exchange requirements and preferences. Simultaneously, another counterparty with an opposing exchange wish can identify the order within the Ethereum Contract and proceed to lock the specified amount of Bitcoin assets, subject to certain conditions predefined in the contract. This locking transaction is facilitated by providing the necessary transaction details to the SPV Contract, ensuring the integrity and validity of the Bitcoin assets.

At this juncture, the Escrow Contract performs a pivotal role by verifying the locking event in Bitcoin through communication with the SPV Contract. Once confirmation of the locking event is received, the Escrow Contract autonomously executes the exchange by transferring the Ethereum assets to the counterparty with the opposing exchange wish. This automated process streamlines the transaction flow and eliminates the need for manual intervention, thereby enhancing the efficiency and reliability of the escrow swap mechanism.

Crucially, the trustless nature of this escrow swap mechanism ensures that all parties involved maintain control over their respective assets throughout the exchange process. By leveraging smart contracts and SPV technology, the risk of counterparty default or fraudulent behavior is significantly mitigated, fostering a high level of confidence and security in the asset exchange process.

Upon successful execution of the swap, the counterparty with the original order can proceed to unlock the assets held in Bitcoin, thereby completing the escrow swap transaction. This final step underscores the seamless nature of the trustless escrow swap mechanism, empowering users to transact across blockchain networks with ease and confidence.

In conclusion, the integration of Solidity Escrow Contracts and SPV Contracts facilitates trustless escrow swaps between Bitcoin and Ethereum, enabling users to exchange digital assets in a secure and efficient manner. By leveraging the inherent capabilities of blockchain technology, this innovative mechanism eliminates the need for intermediaries and central authorities, paving the way for a decentralized future of cross-chain asset exchange in the realm of decentralized finance.

6.3 Conclusion

The convergence of Bitcoin's robust, secure transaction layer with the flexible, programmable features of smart contracts on platforms like Ethereum represents a substantial leap forward in blockchain interoperability and functionality. These technological advancements not only enhance the capabilities of Bitcoin but also open up a myriad of possibilities for creating complex, automated, and responsive financial instruments across multiple blockchain ecosystems. This synergy between Bitcoin and smart contracts is poised to drive significant innovation in the decentralized environment, paving the way for more sophisticated, decentralized financial solutions.

References

[BFT] Breaking bft. quantifying the cost to attack bitcoin and ethereum.

[BTC] Bitcoin whitepaper. https://bitcoin.org/bitcoin.pdf.

[PoW] Pow verification. https://developer.bitcoin.org/devguide/block_chain.

[SPV] Svp contracts repository. https://github.com/summa-tx/bitcoin-spv.