0xVM Yellow Paper V1

Björn Lindfors

bjorn.l@0xvm.com

Abstract. As Bitcoin's influence in the digital economy grows, its inherent limitations in supporting Turing-complete smart contracts and achieving high transaction throughput have necessitated enhancements to its foundational infrastructure. Existing layer-2 solutions, which employ off-chain Ethereum-like VMs and rollup transactions, struggle with issues of security and centralization. To address these challenges, 0xVM is proposed—a Turing-complete VM that operates directly within Bitcoin's consensus layer. 0xVM integrates an execution layer and a data consistency layer, leveraging Bitcoin's robust consensus mechanism to ensure secure, immutable transaction logging and network integrity. Our design circumvents traditional security and scalability issues by encoding VM operations into Bitcoin's UTXOs, thus avoiding the need for additional transaction signatures. Moreover, a novel transaction ordering mechanism based on VM gas fees is introduced to mitigate the economic impact of MEV, prioritizing profits for the 0xVM nodes. Future developments aim to enhance 0xVM's scalability through improved encoding efficiency and parallel processing capabilities. 0xVM not only upholds the security and decentralization ethos of Bitcoin but also extends its utility by facilitating complex DApps directly on its blockchain.

Keywords: Bitcoin · Turing-complete · Scalable

1 Introduction

As the blockchain ecosystem evolves, the quest to amplify Bitcoin's foundational infrastructure to meet the burgeoning demands of the digital economy intensifies. Satoshi Nakamoto's pioneering vision of Bitcoin [8] catalyzed a revolution in DeFi, but the platform's intrinsic limitations, notably its lack of native support for Turing-complete smart contracts and constrained transaction throughput, have emerged as critical constraints. The introduction of Turing-complete platforms such as Ethereum [13] has broadened the possibilities, enabling a diverse range of DApps that extend blockchain utility beyond simple currency transactions.

Existing enhancements to Bitcoin's functionality largely rely on layer-2 solutions that incorporate Ethereum-like VMs off-chain, maintaining alignment with the layer-1 Bitcoin blockchain through rollup transactions. These solutions, however, encounter notable security and decentralization issues. From a security perspective, the off-chain VMs lack the robustness of on-chain alternatives, with the risk of irreversible data loss if metadata are compromised. Moreover, their security depends on layer-1 Turing-complete operations, which Bitcoin does not

support. On the decentralization front, these VMs tend to be centralized, typically controlled by a single entity.

To overcome these issues, 0xVM is introduced as a Turing-complete VM integrated directly into Bitcoin's consensus layer. Unlike existing layer-2 solutions, 0xVM leverages Bitcoin's consensus mechanism for security, ensuring all transactions are recorded on the Bitcoin blockchain. 0xVM consists of an execution layer and a data-consistency layer—the former enables VM executions, while the latter ensures all transactions and state transitions are accurately and immutably logged, crucial for maintaining network integrity over time. Even if all 0xVM nodes crashed, the status of 0xVM and its layers can be restored from the Bitcoin blockchain.

Although the decentralization and security of 0xVM are underpinned by Bitcoin, its Turing completeness and scalability present ongoing challenges. In contrast to Ethereum's recent EIP-4844 upgrade [2], which advances scripting capabilities, Bitcoin lacks native support for complex on-chain VM operations. Therefore, 0xVM transactions are developed based on BRC20 [5], encoding VM operations into UTXOs for processing by the execution layer. The incompatibility between Ethereum's account model and Bitcoin's UTXO system necessitates a unique solution for integrating EVM. Previous approaches have involved separate VM accounts requiring additional signatures, increasing storage demands and reducing scalability. The novel account mapping algorithm dynamically links UTXOs to VM accounts, streamlining transactions by eliminating the need for extra signatures.

0xVM nodes earn from gas fees within the VM. To enhance user experience, these fees can be automatically converted from Bitcoin to 0xVM tokens post-transaction. Additionally, the issue of MEV, wherein Bitcoin miners might dominate bid revenue due to layer-1 fee structures influencing transaction order, is addressed. The proposed mechanism prioritizes transactions based on VM gas fees rather than layer-1 fees, retaining more profits for the 0xVM nodes.

To further enhance the scalability of 0xVM, a no downtime node update process is proposed. This mechanism will optimize the encoding efficiency of VM transactions and enable parallel executions.

In summary, the key features of 0xVM are:

- A VM built directly on Bitcoin's consensus layer,
- Security and decentralization guaranteed by Bitcoin,
- Turing-completeness and scalability achieved through specialized transaction structures and account mapping,
- A novel transaction ordering mechanism to optimize profits for the 0xVM nodes, and
- A no downtime node update process with potential for further scalability improvements through encoding optimization and parallel processing.

2 Related Work

There have been previous attempts to bring expressive smart contracts and scalability to Bitcoin. In terms of expanding Bitcoin's payment scenarios, Liquid

Network [9], functioning as Bitcoin's sidechain, primarily addresses BTC liquidity concerns, enhancing transaction speed and privacy. The Lightning Network [10], a second-layer solution, operates as a payment channel network system atop the Bitcoin blockchain, enabling rapid, cost-effective, and private transactions. However, neither network supports smart contracts.

Adding Turing completeness to bitcoin is an active area of research. Stacks [1] enables smart contracts through its Clarity language and supports atomic swaps, though its reliance on a native token for network operations and the niche programming language restricts broader developer engagement and adoption. The smart contracts on 0xVM support the Solidity language, lowering the entry barriers for developers. Additionally, there are plans to support more programming languages in the future, such as Rust and Go. Bitcoin's mainnet upgrade, the Taproot protocol, significantly enhances Bitcoin's scripting capabilities by compressing and concealing complex scripts, supporting more intricate logic without additional data overhead. BitVM [6], leveraging Taproot, operates on a fraud-proof and challenge-response model, enhancing off-chain smart contract functionality while minimizing on-chain data footprint. However, Taproot's complexity and its focus on off-chain computation introduce challenges in scalability and interaction. 0xVM, using Taproot technology, enhances Bitcoin by enabling complex smart contracts directly on its blockchain. It streamlines transaction processes and extends functionality without compromising security. Veda [1], as a Layer 2 solution, aims to integrate Ethereum's account-based model into Bitcoin's UTXO framework. It embeds signature verification into UTXO transactions to enable Ethereum-like advanced functionalities such as smart contracts. However, this approach adds complexity and data payload to the Bitcoin network, contrary to Bitcoin's minimalist and efficient design principles. Arch [2] uses a ZK rollup protocol to build a Turing-complete virtual machine over Bitcoin. However, ZK rollups suffer significantly from centralized provers without a sound solution so far [12]. 0xVM serves as the execution layer for Bitcoin, aiming to integrate Bitcoin's security while dynamically linking UTXOs to VM accounts through an innovative account mapping algorithm. This simplifies transactions by eliminating the need for additional signatures.

Ethereum, through its EVM, provides a Turing-complete scripting language, enabling developers to write complex smart contracts. This capability has led to the development of numerous Layer 2 solutions, enhancing scalability, security, and versatility. Optimistic Rollups execute transactions off-chain but rely on the Ethereum mainnet for security through periodic batch submissions and fraud proofs. ZK-Rollups use zero-knowledge proofs to verify off-chain transactions, then submit concise proofs on-chain to ensure transaction integrity and correctness. The primary challenge in directly applying Ethereum's Layer 2 solutions to Bitcoin lies in Bitcoin's scripting limitations. Bitcoin's scripting language, Script, is not Turing-complete, limiting the complexity of smart contracts that can be executed on its blockchain. This limitation impedes the implementation of

[1] `https://docs.vedaord.com/`.
[2] `https://arch-network.gitbook.io/arch-documentation`

complex Layer 2 solutions. 0xVM aims to integrate Turing-complete functionality into Bitcoin's architecture to overcome these limitations.

3 Preliminaries

This section introduces the fundamental concepts and technologies essential for understanding the rest of the paper. These concepts include Turing-complete smart contracts (§3.1), the account model (§3.2), the transaction architecture (§3.3), and the parallel VM (§3.4) that underpin blockchain networks.

3.1 Turing-Complete Smart Contracts

Turing-completeness in smart contracts [4] refers to their capability to perform any computation, given sufficient resources. On networks like Ethereum, Turing-complete smart contracts can execute any programmable function, greatly expanding their utility beyond simple transactions.

Ethereum has played a crucial role in popularizing Turing-complete smart contracts. Unlike Bitcoin, which is primarily a digital currency, Ethereum's blockchain is designed as a comprehensive platform to facilitate and execute smart contracts using its native programming language, Solidity. Ethereum's Turing-complete capability allows developers to create applications that go beyond simple transactions, including DeFi, automated governance systems, and complex DApps.

Potential applications of Turing-complete smart contracts extend beyond financial transactions. They are increasingly employed in areas like voting systems, supply chain management, digital identity verification, and more, where a secure, transparent audit trail is essential. In these systems, smart contracts enhance transparency, reduce fraud, and streamline processes.

3.2 Account Model

The account model is a fundamental framework in blockchain technology, defining how data and state are stored and manipulated. It contrasts with the UTXO model used by Bitcoin [8]. Understanding these models is crucial for grasping how blockchains handle transactions and maintain states.

3.2.1 UTXO Model

The UTXO model, used by Bitcoin and other blockchain systems, operates on a ledger of unspent transaction outputs. Unlike traditional account-based models, the UTXO system does not maintain an aggregate balance for each user but instead sums all unspent outputs that the user can spend with their private key.

In the UTXO model, transactions process by consuming previous unspent outputs (inputs) and generating new unspent outputs (outputs). These outputs serve as inputs for future transactions. This method ensures that each coin's entire transaction history is traceable back to its creation point, enhancing security by preventing double-spending.

Graphical Representation of UTXO Flow A useful way to visualize the UTXO model is through a directed graph, where nodes represent transaction outputs, and edges represent the consumption of these outputs by subsequent transactions. Each node can have multiple incoming edges but only one outgoing edge, representing the output spent in a new transaction.

Address Changes and Update Schemes A critical aspect of the UTXO model is the management of addresses. When a transaction occurs, the unspent outputs linked to the sender's address are ßpent,änd new outputs are created for the receiver's address. Often, change from the transaction is sent back to the sender but returned to a new address generated by the sender's wallet. This practice enhances privacy by dispersing balances across multiple addresses.

Furthermore, the update scheme in the UTXO model revolves around validating transaction inputs against the set of unspent outputs. A transaction is valid if the inputs correspond to a list of unspent outputs and the total value of inputs matches the total value of outputs, barring transaction fees.

This model offers several advantages:

- **Enhanced Privacy:** By not linking funds to accounts but to discrete transaction outputs that frequently change addresses, user privacy is significantly improved.
- **Scalability for Parallel Processing:** Independent transaction outputs can be processed in parallel, improving the scalability of the system.
- **Security Against Fraud:** Each transaction output must be uniquely referenced by a transaction input, preventing the same funds from being spent twice.

However, the UTXO model can pose challenges for developers, especially in smart contract design, as it does not inherently support account states or more complex transaction logic found in account-based models like Ethereum.

3.2.2 Account-Based Model

Ethereum popularized the account-based model [3], similar to traditional bank accounts, where the blockchain maintains a state for each account. This state includes balance, contract code (for contract accounts), and storage. Transactions directly alter these states, transferring value or executing smart contract functions.

Comparison with UTXO Model Unlike the UTXO model used by Bitcoin, the account-based model tracks account balances and states. This distinction simplifies transaction processes by abstracting the need to handle outputs from multiple transactions, thereby streamlining the execution of smart contracts and state management.

Execution Context In the account-based model, transactions are contextually aware of the sender through the `tx.sender()` function, pivotal in smart contract interactions. This function identifies the transaction originator, allowing smart

contracts to execute logic conditionally based on the sender's identity and current state.

This model significantly eases developing complex smart contracts that require persistent states or the execution of multi-step transactions. It naturally supports maintaining various states within each account, essential for applications needing more than simple value transfers, such as DAOs or complex DApps.

Moreover, while the account-based model enhances clarity and usability for developers, it potentially reduces privacy compared to the UTXO model by directly linking transactions to account balances and states. However, it compensates for this with increased functionality and straightforwardness in smart contract execution and state management.

3.2.3 Solana's Account Model

Solana introduces a unique twist to the account model [14], designed to optimize speed and efficiency. While adopting an account-based model similar to Ethereum, Solana's architecture allows for high throughput and low transaction costs. In Solana, accounts can hold data beyond balances, including smart contract state, which is crucial for the blockchain's scalability and performance.

Solana's model is distinctive because it leverages the concept of "rent"for account storage, where accounts pay rent to maintain their data on the network, ensuring efficient use of network resources. Additionally, Solana introduces a timestamping feature called PoH, which helps order transactions and enhances the blockchain's throughput.

In conclusion, the account model in blockchain design significantly influences how transactions are processed, how states are maintained, and the overall efficiency and usability of the blockchain. Each model—UTXO, account-based, and Solana's enhanced account model—offers unique advantages and is suited to different blockchain applications. Understanding these models is crucial for blockchain developers, as the model choice impacts the design, functionality, and scalability of blockchain applications.

3.3 Transaction Architecture

3.3.1 BTC and BRC-20 Transaction Architecture

The Bitcoin transaction process primarily involves four steps (see Fig. 1): (1) submission of transactions to the node pool. (2) validation of each transaction by nodes based on UTXOs. (3) mining nodes bundle validated transactions into blocks and broadcast them. (4) nodes add the blocks to the blockchain. The Bitcoin transaction architecture [11] primarily operates on the UTXO model. This process ensures that all Bitcoins are accounted for, preventing double-spending.

Transitioning from Bitcoin's UTXO model, the BRC-20 protocol [5] introduces token functionalities similar to Ethereum's ERC20 standards, enabling the deployment, minting, and transferring of tokens on the Bitcoin blockchain (see Fig. 2). Each BRC-20 token can embed unique data, turning sats into versatile digital assets.

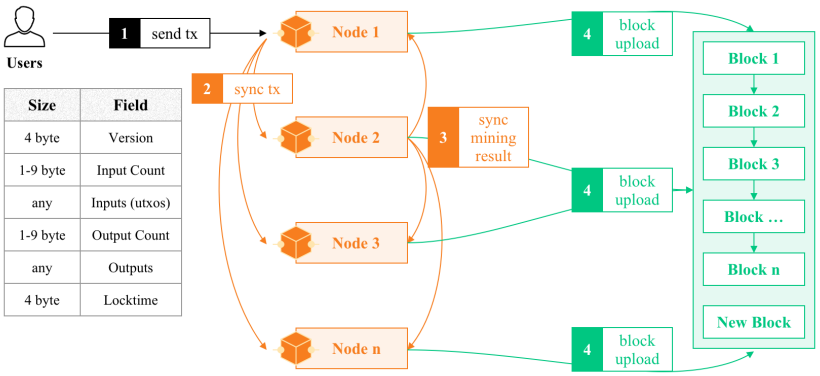

Fig. 1. Bitcoin transaction architecture

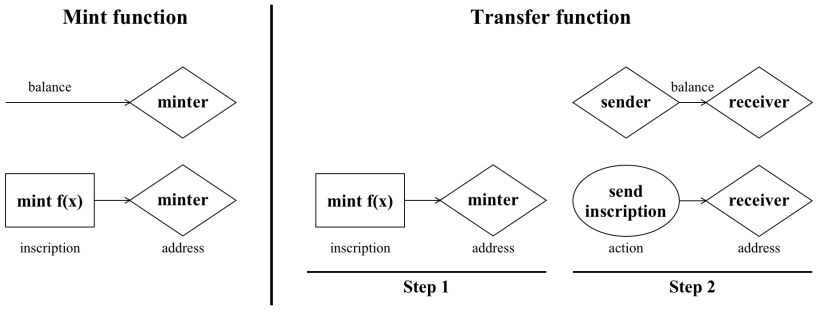

Fig. 2. Brc20 transaction architecture

The BRC-20 transactions are processed and recorded through a series of steps maintained by indexing servers: (1) indexing: servers catalog each BRC-20 deployment to retrieve and chronicle token data. (2) verification: each transaction is validated against the balances recorded in the local ledger to confirm its legitimacy. (3) ledger update: following a transaction, the local ledger is amended to accurately reflect updated token balances, with indexers ensuring the ledger's continuous accuracy.

This enhanced framework allows the Bitcoin network to support a wider range of functionalities, from NFTs to complex smart contracts, significantly broadening Bitcoin's utility and establishing a more versatile platform for digital assets.

3.3.2 EIP-4844 Transaction Architecture

Ethereum Improvement Proposal 4844 [2], commonly referred to as "Proto-Danksharding,ïs a forward-looking enhancement that was part of Ethereum's

recent Cancun upgrade. This proposal significantly improves scalability of the Ethereum network through a shard-based solution. By partitioning the network into multiple segments, or ßhards,ït distributes workload and increases the capacity for processing transactions.

A pivotal feature of EIP-4844 is the introduction of "blob-carrying transactions,"which support operations such as BRC20. These transactions incorporate an additional data component, termed a "blob,"which allows for substantial storage at a fraction of the usual cost. This innovation optimizes data storage within the Ethereum ecosystem and significantly reduces transaction fees by segregating blob storage from traditional transaction data. Blobs offer a cost-effective solution for rollups to manage large-scale data requirements essential for complex operations, enhancing the efficiency and scalability of network operations.

This unique approach provided by EIP-4844 marks a significant advancement in Ethereum's capability to handle a higher volume of transactions and complex contract interactions efficiently, paving the way for broader adoption and more sophisticated applications on the blockchain.

3.4 Parallel VM

Both Sui and Solana introduce innovative solutions to address scalability and performance challenges in blockchain technology, emphasizing parallel processing and advanced consensus mechanisms to achieve high throughput and low latency in transaction processing.

3.4.1 Sui's Parallel VM

Sui [7] utilizes the Move programming language to facilitate parallel transaction processing, significantly enhancing system scalability and efficiency. This architecture allows the network to dynamically scale horizontally, adjusting its capacity in response to demand without requiring consensus for straightforward transactions.

Status Sharding in Transactions Sui employs ßtate sharding"to effectively manage transactions in parallel. This involves partitioning the ledger state into multiple shards, with each shard handling transactions independently. This sharding allows for simultaneous processing across different shards, drastically improving throughput and reducing synchronization overhead.

Dynamic Resource Allocation Sui dynamically adjusts computing resources based on workload, maintaining efficient network operation under varying load conditions. This adaptability helps manage transaction peaks without compromising performance.

Economic Efficiency The system's efficiency is further enhanced by its low gas fees, achieved through reduced computational overhead in transaction processing via status sharding. This makes Sui cost-effective for developers and users alike, particularly beneficial for high-throughput applications requiring cost efficiency.

3.4.2 Solana's Parallel VM

Solana's architecture [14] is distinguished by its high-throughput capabilities, supported by the PoH consensus mechanism and parallel transaction execution. This combination significantly reduces processing times and increases overall throughput.

PoH PoH serves as a decentralized clock that timestamps transactions, facilitating a faster consensus process by allowing nodes to verify the order and timing of events independently.

Sealevel: Parallel Transaction Processing Solana's runtime environment, Sealevel, maximizes throughput by enabling parallel execution of smart contracts. It identifies and processes non-overlapping transactions concurrently across the network.

Status Sharding in Transactions Although Solana does not use a traditional sharding architecture, it emulates ßtatus sharding"by segmenting the state into manageable parts for parallel processing. This segmentation helps optimize hardware utilization, speeding up transaction validation and state replication.

Pipeline Processing The network incorporates a pipelined approach to transaction processing, delegating different stages of the transaction lifecycle to specialized hardware components, thus enhancing data flow and processing speed.

Turbocharging Throughput with Gulf Stream Gulf Stream optimizes transaction handling by pushing caching and forwarding decisions to the network's edge, enabling pre-execution of transactions and reducing confirmation times.

4 0xVM Structure

The 0xVM structure represents a sophisticated framework designed to enhance and optimize smart contract execution within a blockchain environment. This structure is fundamental to the system's ability to support complex and scalable applications, ensuring robust, secure, and efficient transaction processing. By integrating advanced VM technology, 0xVM facilitates a wide range of DApps, driving innovation and functionality in the blockchain space. This section delves into the participants and layers that constitute the 0xVM architecture, each playing a pivotal role in the network's dynamics and overall performance.

4.1 Participants

The 0xVM ecosystem comprises various key participants, each playing distinct roles crucial for the efficient functioning and governance of the network. This section explores the integral roles of users, 0xVM operators, data consistency providers, and 0xVM validators. Each group contributes uniquely to the network's

operations, from interacting with the system and processing transactions to ensuring data consistency and network integrity. Understanding these roles provides insight into the complex interplay of activities that maintain the 0xVM system's robustness and reliability.

4.1.1 Users

In the 0xVM ecosystem, users are the primary agents of interaction, engaging with the network through a 0xVM-supported frontend interface.

Users interact with the 0xVM network by submitting transactions encoded with 0xVM-specific code to the Bitcoin blockchain. These transactions facilitate engagement with the 0xVM network and contribute to the broader Bitcoin ecosystem. Once processed within the 0xVM framework, these transactions lead to changes in the network's state or activate specific functionalities.

4.1.2 0xVM Operators

0xVM Operators play a crucial role in maintaining the network's efficiency and performance. They are responsible for executing transactions and ensuring state consensus. Operators run the transaction execution process, including encoding and decoding, and updating the network's state. Post-execution, they broadcast the new state across the network for consensus, ensuring smooth and efficient network operations.

4.1.3 Data Consistency Providers

Data Consistency Providers ensure the accuracy and reliability of the historical record of all 0xVM transactions. They help supply a consistent and transparent network state by providing a storage platform for 0xVM state updates. These updates are derived from the execution layer, focusing on identifying the majority state to maintain data consistency.

4.1.4 0xVM Validators

0xVM Validators are essential for overseeing the network's integrity. Their responsibilities include:

- Verifying the consistency between Bitcoin's history and the data consistency layer. In case of discrepancies, they issue an SPV to alert all network participants not to trust the affected node, effectively isolating dishonest nodes.
- Ensuring the accuracy of historical state changes recorded by the Data Consistency Providers. Incorrect state updates are broadcasted to the network to prompt corrective actions.
- Although the execution layer lacks explicit consensus, Validators play a crucial role in confirming state updates by seeking the majority state from execution outcomes, thereby indirectly contributing to consensus through the Data Consistency Layer.

These updates to the mechanism highlight the Validators' critical role in maintaining network integrity, facilitating a robust and transparent 0xVM ecosystem.

4.2 0xVM Layers

4.2.1 BTC Layer

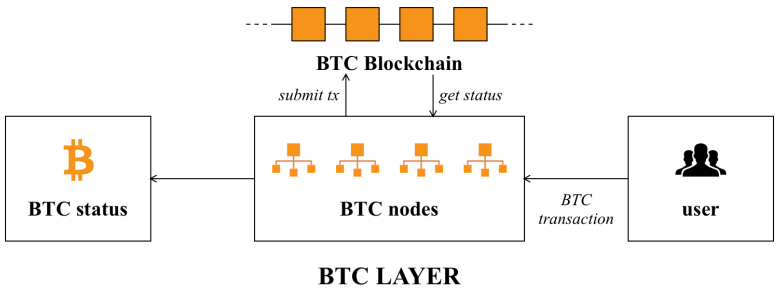

Fig. 3. Bitcoin layer

The BTC layer acts as the foundational stratum through which users initiate interactions by submitting transactions embedded with 0xVM code. These transactions are propagated to the Bitcoin nodes, where consensus is achieved, leading to the broadcast of these transactions and the consequent update of the Bitcoin blockchain's state (see Fig. 3).

Algorithm 1 Interaction with the BTC Layer

```
procedure SubmitTransaction(0xVM_tx)
    btc_tx ← Embed0xVMCode(0xVM_tx)
    Broadcast(btc_tx)
end procedure

function Embed0xVMCode(0xVM_tx)
    return btc_tx                    ▷ Embed 0xVM code into a BTC transaction
end function

procedure Broadcast(btc_tx)          ▷ Implementation for broadcasting the
transaction to the Bitcoin network
end procedure
```

This pseudocode outlines the process within the BTC Layer, starting with the submission of a transaction containing 0xVM code, embedding this code

into a Bitcoin transaction, and finally broadcasting it to the network. The `SubmitTransaction` procedure demonstrates how users interact with the 0xVM system, while the `Embed0xVMCode` function and the `Broadcast` procedure elucidate the subsequent steps of transaction processing and network propagation, respectively.

4.2.2 Execution Layer

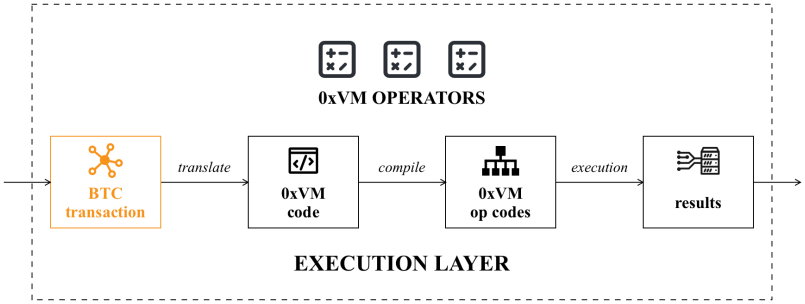

Fig. 4. Bitcoin execution layer

In the 0xVM ecosystem, becoming a legitimate node requires staking 0xVM tokens, which ensures operational integrity and aligns the interests of node operators with the network's well-being. Once committed, 0xVM nodes continuously monitor the Bitcoin blockchain for transactions containing 0xVM code.

Upon identifying such transactions, nodes initiate a decoding process to extract the 0xVM code and convert it into executable operations or op codes. These operations are then executed to update the network's state, with the new state being disseminated across the network to achieve consensus (see Fig. 4).

In the Execution Layer, nodes serve as the operational core, executing 0xVM transactions and updating the network state. The process includes staking tokens for legitimacy, monitoring the blockchain for the 0xVM transactions, decoding these transactions into executable code, executing these operations, updating the network state, and propagating this state across the network for consensus. This layer ensures the network's functionality and integrity, with mechanisms in place to handle discrepancies and penalize nodes for erroneous executions, thereby maintaining a robust and reliable 0xVM ecosystem.

4.2.3 Data Consistency Layer

The Data Consistency Layer (see Fig. 5) forms the archival backbone of the 0xVM network, ensuring a synchronized and immutable record of state transitions. This

Algorithm 2 Node Operation in the Execution Layer

```
1: procedure StakeTokens(tokens)
2:     Lock tokens as collateral for node operation
3: end procedure

4: procedure MonitorBlockchain
5:     while true do
6:         tx ← Get0xVMTransaction
7:         if tx is not null then
8:             ProcessTransaction(tx)
9:         end if
10:    end while
11: end procedure

12: function Get0xVMTransaction
13:    Scan Bitcoin blockchain for the 0xVM transactions
14:    return tx
15: end function

16: procedure ProcessTransaction(tx)
17:    op_codes ← Decode(tx)
18:    Execute(op_codes)
19:    UpdateState
20:    PropagateState
21: end procedure

22: function Decode(tx)
23:    Decipher 0xVM code to op codes
24:    return op_codes
25: end function

26: procedure Execute(op_codes)
27:    Execute operations to derive new state
28: end procedure

29: procedure UpdateState
30:    Integrate new state into the 0xVM network
31: end procedure

32: procedure PropagateState
33:    Broadcast new state for consensus
34: end procedure
```

Algorithm 3 Data Consistency Operation

```
1: procedure SynchronizeState
2:     while new state available do
3:         RetrieveNewState
4:         UpdateLedger
5:     end while
6: end procedure

7: function RetrieveNewState
8:     Fetch the latest consensus state from the 0xVM network
9:     return new_state
10: end function

11: procedure UpdateLedger(new_state)
12:     Incorporate new state into the historical ledger
13: end procedure

14: procedure QueryLedger(query)
15:     Retrieve current or historical state based on query
16:     return query_result
17: end procedure

18: procedure ObtainStateFromExecution
19:     Identify majority state from execution layer outcomes
20:     UpdateLedgerWithExecutionState
21: end procedure
```

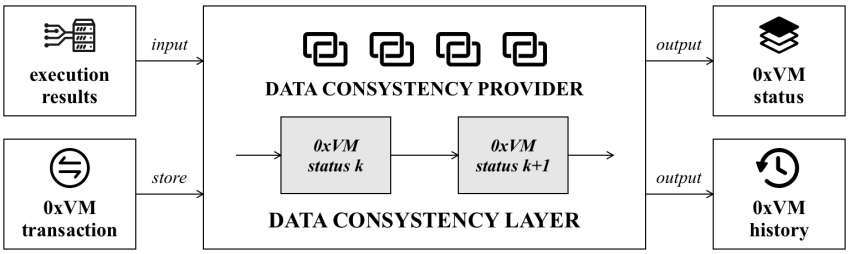

Fig. 5. Data consistency layer

layer is essential for maintaining the integrity of both the historical and current network states, continually updating the ledger with validated transactions.

This enhanced approach to data consistency highlights the 0xVM network's commitment to accuracy, reliability, and transparency, ensuring the system remains robust and trustworthy.

4.3 System Progress

The operational progression of the 0xVM system (see Fig. 6) is meticulously structured to ensure its functionality and coherence within the broader Bitcoin ecosystem. This overview outlines the complete execution journey of 0xVM, detailing each procedural step.

(a) **Transaction Submission**: The operational cycle begins when users submit transactions to the Bitcoin Layer. These transactions, embedded with 0xVM-specific code, integrate into the Bitcoin blockchain without disrupting the network's existing flow.

(b) **Code Deciphering by 0xVM Operators**: Once embedded, 0xVM Operators process these transactions. By decoding the embedded 0xVM code, converting it into executable operations or op codes, which trigger the desired functionalities within the 0xVM network.

(c) **Role of Data Consistency Providers**: Simultaneously, Data Consistency Providers capture and archive the data and state transitions resulting from these transactions, ensuring a transparent and immutable history of all network activities.

(d) **State Change and Broadcast**: The culmination of execution and data recording leads to a state change within the 0xVM network. This new state is broadcast across the network, requiring consensus among 0xVM Validators and other participants to validate and synchronize the updated state.

(e) **Consensus and Validation**: This consensus is crucial for legitimizing the updated state, confirming that it is recognized as valid by all network participants.

Through these interconnected steps, the 0xVM system demonstrates its ability to operate within the Bitcoin network's constraints while showcasing its unique

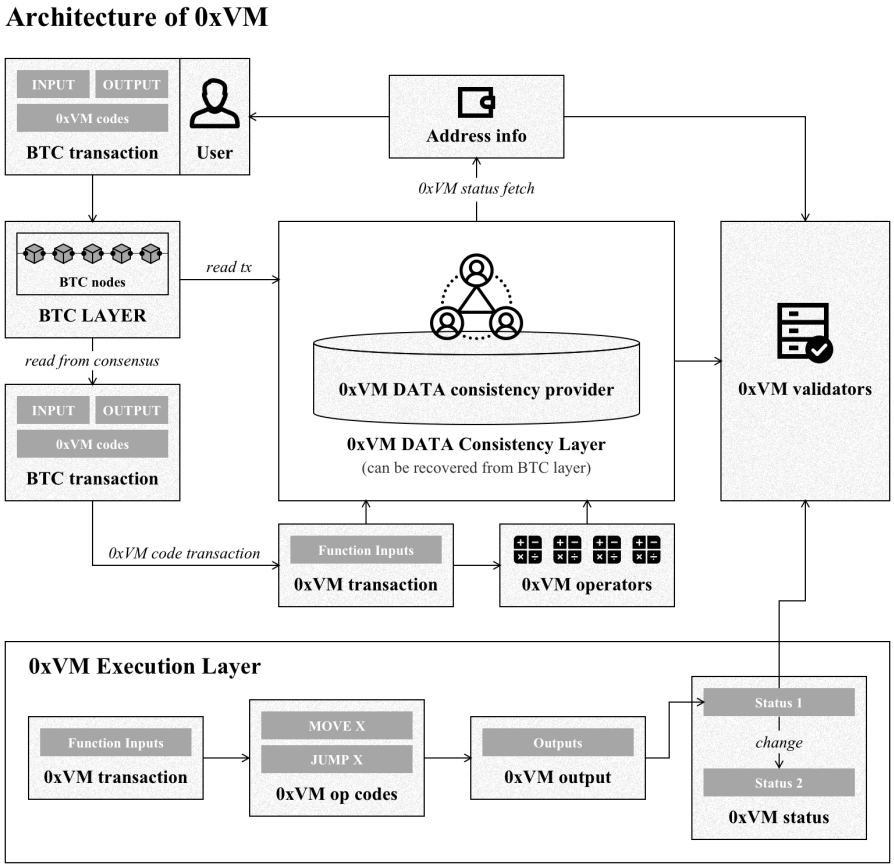

Fig. 6. Architecture of 0xVM

operational features. This synergy enables 0xVM to provide a dynamic, secure, and reliable environment for transaction execution and recording, solidifying its role as a scalable and efficient extension of the Bitcoin blockchain.

This structured explanation aims to offer a comprehensive and coherent overview of the 0xVM system's operational mechanics, emphasizing the interaction of its various components.

5 0xVM Transactions

This section outlines the architecture of 0xVM Transactions (§5.1), account mapping (§5.2), details asset transfer through 0xVM, smart contract creation and invocation (§5.3), and discusses successful versus failed executions, transaction gas, and order protocols (§5.4).

5.1 Transaction Structure

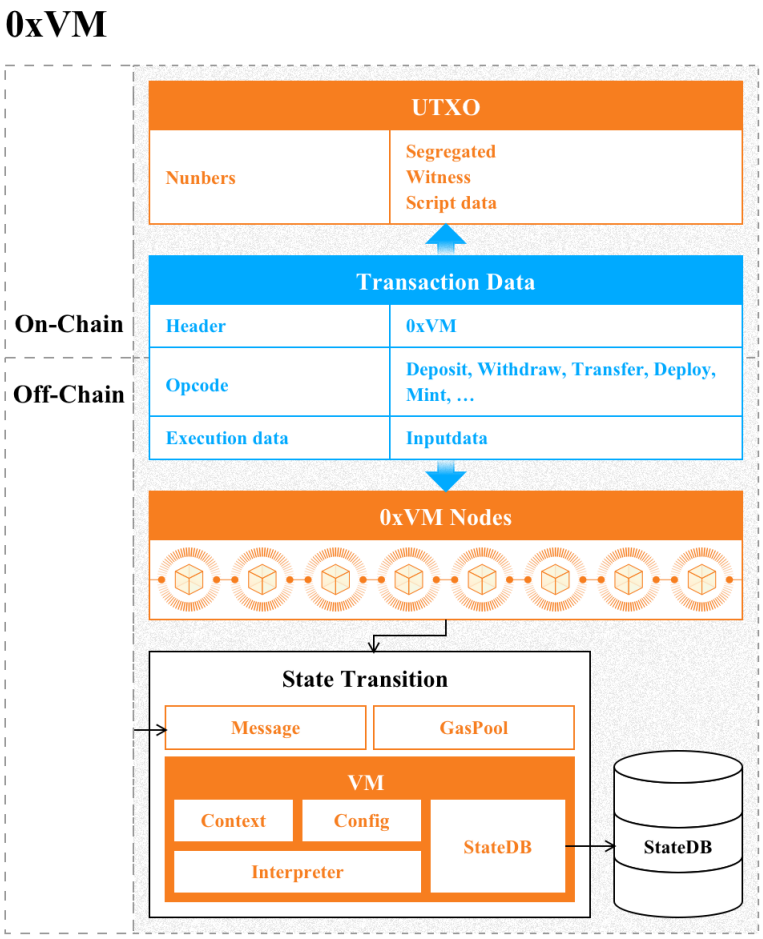

Fig. 7. 0xVM structure

The essence of 0xVM lies in its novel approach of integrating a naming system for specific protocols, adding opcodes, and establishing a VM execution layer above the existing Bitcoin consensus layer. This execution layer is designed to handle user commands, enabling the Bitcoin ecosystem to support complex smart contracts and decentralized application functionalities (see Fig. 7).

Similar to the BRC20 protocol, 0xVM initiates into the Bitcoin network by defining a unique protocol name that distinctly marks its operations. This approach is pivotal as it introduces a suite of new opcodes essential for executing complex instructions within the 0xVM layer. These opcodes significantly enhance

Bitcoin's existing scripting language, enabling it to support a broader array of functionalities necessary for operating smart contracts and DApps. The extension of Bitcoin's scripting capabilities through these opcodes allows for wider range of operations, including those required for the sophisticated management and interaction with smart contracts. Execution data, which comprises the input data necessary for user calls to smart contracts, is a critical component of this system. This development represents a substantial leap forward in Bitcoin's evolution, equipping it with the necessary tools to host and manage DApps directly on its secure and decentralized platform, thereby expanding its utility and scalability within the blockchain ecosystem.

5.2 Account Mapping

The need for an account model within 0xVM is arises from the necessity to bridge the operational differences between Bitcoin's UTXO model and Ethereum's account-based system. Bitcoin, fundamentally relying on UTXOs, tracks coins as individual pieces of data that are neither linked to a specific identity nor carry a state beyond their spend-status. Conversely, Ethereum employs an account-based model where states such as balances, contract code, and storage are continuously tracked and updated. The integration of 0xVM as an execution layer on Bitcoin requires a robust account mapping mechanism to align these fundamentally different models, ensuring that transactions can be seamlessly processed across both systems without compromising the structural integrity of either blockchain.

Veda's account model, while innovative, illustrates certain limitations inherent in direct adaptating of Ethereum's architecture onto Bitcoin's framework. Veda integrates Ethereum's account system by embedding signature verifications directly within UTXO transactions, inevitably increasing the redundancy within the Bitcoin network. This approach imposes additional data loads on a system not originally designed to handle such complexities, resulting in inefficiencies that are undesirable. In contrast, 0xVM introduces an account mapping model that decouples user authentication from transaction verification. By eliminating the need to verify signatures within each transaction, 0xVM enhances processing speed and significantly improves user experience without compromising security.

The account mapping model proposed by 0xVM aims to associate each Bitcoin address with a corresponding 0xVM account. This model allows 0xVM to continuously monitor all transactions on the Bitcoin blockchain. When an input in a transaction is identified as originating from a Bitcoin address managed by 0xVM, the system performs the necessary operations on the corresponding 0xVM account without requiring signature verification. This process involves direct modifications to account states and balance adjustments, efficiently mapping each UTXO to a fixed 0xVM account address. This methodology effectively maintains the integrity and traceability of transactions, simplifying the management of state changes across the integrated blockchain platforms.This design not only simplifies the transaction process but also minimizes potential security risks associated with traditional signature verifications. By adopting this method, 0xVM fosters

a more transparent, efficient, and secure transaction processing framework, which is crucial for integrating the system into broader blockchain applications.

5.3 Transaction Types

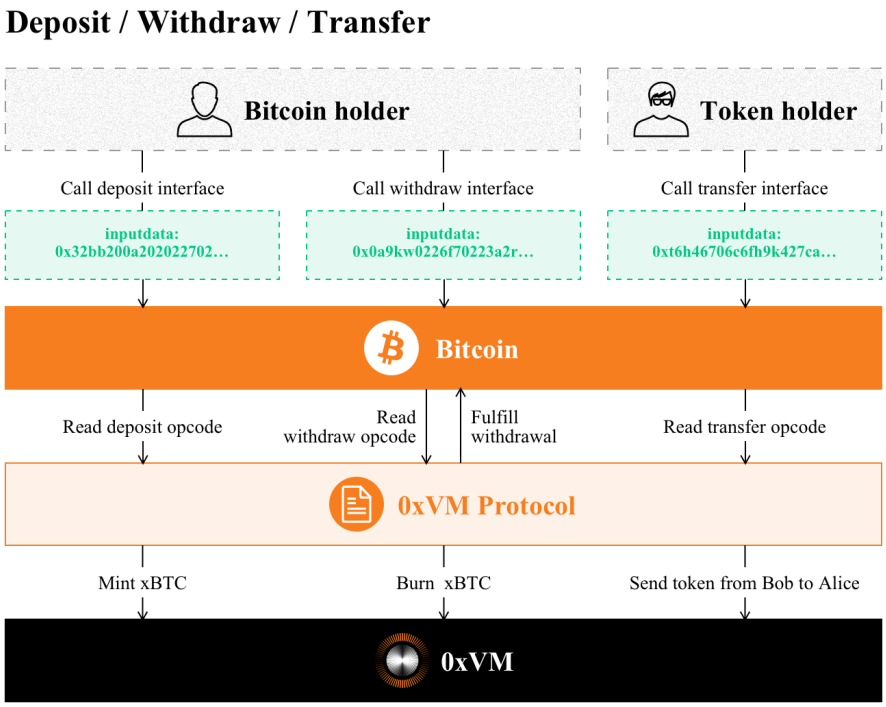

Fig. 8. 0xVM's deposit, withdraw and transfer structure

5.3.1 Deposit and Withdraw

Within the 0xVM framework, the mechanism anchoring xBTC to BTC operates in a trustless and decentralized manner, ensuring that the minting process is recorded within Bitcoin's existing scripting without the need for additional anchoring incentive fees. By integrating smart contracts into the Bitcoin blockchain via 0xVM, Bitcoin can serve as a currency asset in various DeFi transactions, such as lending and the creation of BTC-backed stablecoins (see Fig. 8).

Pseudocode for Asset Deposit and Withdrawal in 0xVM

```
Procedure DepositAsset(userAddress, assetAmount)
    If not IsValidAddress(userAddress) then
        Print "Deposit failed: Invalid address."
        Exit
    EndIf
    IncreaseUserBalance(userAddress, assetAmount)
    LogDeposit(userAddress, assetAmount)
    Print "Deposit successful."
EndProcedure

Procedure WithdrawAsset(userAddress, assetAmount)
    If not IsValidAddress(userAddress) then
        Print "Withdrawal failed: Invalid address."
        Exit
    EndIf
    If not HasSufficientBalance(userAddress, assetAmount) then
        Print "Withdrawal failed: Insufficient balance."
        Exit
    EndIf
    DecreaseUserBalance(userAddress, assetAmount)
    LogWithdrawal(userAddress, assetAmount)
    Print "Withdrawal successful."
EndProcedure
```

Listing 1.1. Asset Deposit and Withdrawal

(1) Deposit Operation

Users initiate a deposit transaction on the Bitcoin blockchain by transferring funds to a 0xVM wallet address, which notifies 0xVM protocol of the amount of BTC deposited and the user's receiving 0xVM address. The 0xVM protocol then mints a 1:1 equivalent of xBTC tokens and sends them to the user's receiving address.

(2) Withdrawal Operation

Users create a withdrawal transaction on the Bitcoin blockchain, informing the 0xVM protocol of the amount of xBTC to be withdrawn, the 0xVM addresses involved, and the receiving Bitcoin address on the blockchain. Subsequently, 0xVM protocol destroys the specified amount of xBTC from the given 0xVM addresses and issues an equivalent amount of BTC to the provided Bitcoin address, completing the withdrawal process.

During this process, 0xVM execution layer nodes are required to provide signatures for the BTC withdrawal transactions. A withdrawal operation can only be successfully executed once a predetermined number of signatures has been obtained.In this framework, 0xVM acts as a bridge facilitating Bitcoin's integration with DeFi by leveraging xBTC as a pegged asset.

5.3.2 Token Transfer

In 0xVM, defining a new protocol name, such as "TransferProtocol", and adding dedicated opcodes, such as "TRANSFER", for executing transfer operations is imperative. 0xVM execution layer interprets and executes opcodes based on the TransferProtocol. Here's the detailed process of Alice transferring funds to Bob (see Fig. 8):

Pseudocode for Token Transfer in 0xVM

```
Procedure TransferToken(sourceAddress, destinationAddress,
    tokenAmount)
    If not IsValidAddress(sourceAddress) or not
        IsValidAddress(destinationAddress) then
        Print "Transaction failed: Invalid address(es)."
        Exit
    EndIf
    If not HasSufficientTokens(sourceAddress, tokenAmount) then
        Print "Transaction failed: Insufficient tokens."
        Exit
    EndIf
    LockTokens(sourceAddress, tokenAmount)
    MoveTokens(sourceAddress, destinationAddress, tokenAmount)
    UnlockTokens(sourceAddress)
    If VerifyTransfer(destinationAddress, tokenAmount) then
        LogTokenTransfer(sourceAddress, destinationAddress,
            tokenAmount)
        Print "Token transfer successful."
    Else
        RollbackTransaction(sourceAddress, tokenAmount)
        Print "Token transfer failed: Rolled back transaction."
    EndIf
EndProcedure
```

Listing 1.2. Token Transfer

5.3.3 Contract Creation and Invocation

The principle of 0xVM involves adding opcodes on top of the Bitcoin consensus layer and establishing an execution layer. It includes three steps: (1) Contract definition, allowing the storage and updating of values. (2) Contract deployment, placing the compiled bytecode of the contract into Bitcoin's UTXO, deploying the contract on 0xVM nodes, and assigning it a unique address. (3) Contract invocation, specifying the contract address, the function to be called, and any necessary parameters, and executing the call (see Fig. 9).

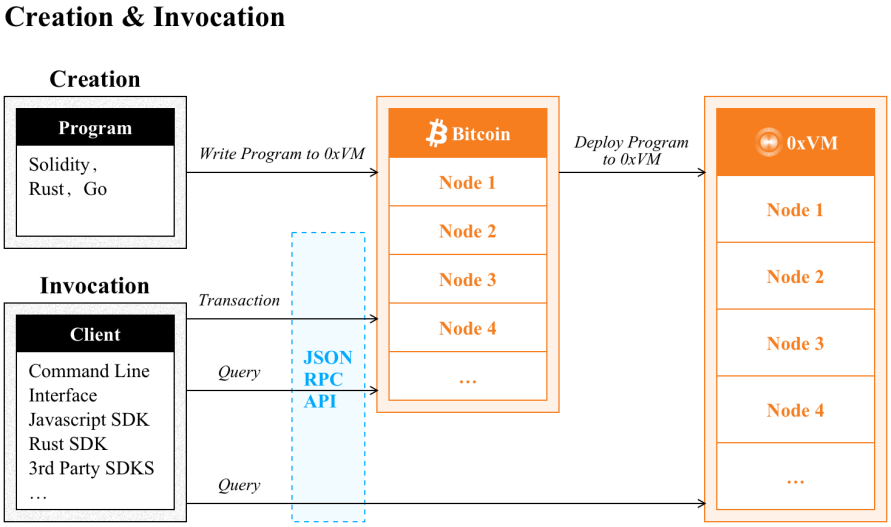

Fig. 9. 0xVM contract creation and invocation

5.3.4 Contract Transaction

The process of a user sending a valid transaction involves several key steps: (1) Constructing transaction data: The user needs to construct a transaction, which includes specifying the transaction type (e.g., transfer, contract invocation), the destination address, the transaction amount, and any necessary parameters. (2) Signing the transaction: The user needs to sign the transaction using their private key. (3) Submitting the transaction: The signed transaction is submitted to the Bitcoin network. Subsequently, it will be executed by the 0xVM execution layer based on the logic specified in the transaction data. (4) Verifying and executing the transaction: The transaction will be processed through different execution paths based on its type. For example, if the transaction is a smart contract invocation, it will be forwarded to the logic handling of the contract; if it is a transfer, it will directly affect the account balance.

5.4 Transaction Execution

5.4.1 Successful Execution

In the 0xVM network, upon successful transaction processing, the execution layer updates its internal state to reflect changes in account balances, smart contract states, or any other relevant modifications. After the execution of transactions by the execution layer, the data consistency layer receives notifications regarding the resulting data changes. Subsequently, it updates its archival records to

incorporate these modifications and meticulously documents each alteration to ensure accurate historical data maintenance.

5.4.2 Failed Execution

In the 0xVM network, transaction execution is meticulously monitored, with mechanisms in place to halt transactions upon detecting discrepancies during the execution phase. Unlike Bitcoin transactions, where gas fees are deducted regardless of transaction outcomes, 0xVM does not deduct gas fees if a transaction fails. Additionally, the 0xVM execution layer is responsible for recording detailed information about the nature of transaction failures. The data consistency layer archives this detailed information, ensuring comprehensive records of all failed transactions.

To ensure the security of user assets and the robustness of transactions, 0xVM introduces features similar to atomic transactions, particularly evident in scenarios such as user deposits. For example, when a user successfully deposits BTC on the Bitcoin blockchain, the corresponding transaction on 0xVM is guaranteed to succeed. This cross-layer success mirroring ensures fault tolerance for certain critical transactions, thereby enhancing user confidence and safeguarding the integrity of assets within the 0xVM network.

5.4.3 Transaction Gas and Automated Gas Market

In 0xVM, a novel gas fee mechanism has been designed where the execution layer scans Bitcoin blocks to identify and retrieve transactions related to 0xVM activities. Gas fees are levied only after a transaction is successfully processed by the execution layer. This mechanism also allows transactions to fail without any gas fees being deducted if the gas available is insufficient to complete the transaction.

This fee mechanism is distinct from typical implementations found in other Layer 1 and Layer 2 protocols, where fees may be deducted regardless of the transaction outcome. 0xVM does not suffer from DDoS attacks because it only scans on-chain transactions. 0xVM's approach ensures fairness and prevents the loss of gas coins due to transaction failures. To address potential shortages of gas coins among users, 0xVM has integrated a BTC-to-gas coin exchange functionality into its infrastructure to facilitate the acquisition of additional gas coins.

This gas mechanism not only enhances the network's capacity to withstand potential security threats but also ensures the fairness and efficiency of the transaction process. By processing only verified transactions and linking gas fees directly to transaction execution success, 0xVM enhances network integrity and user trust. Users can send BTC to a designated address to exchange for gas coins, a process that is seamlessly integrated into the transaction workflow. This integration not only simplifies the user experience but also enhances accessibility, allowing users to manage their transactions effectively without interruption.

5.4.4 Transaction Orders and Priority Gas Auction

In traditional Bitcoin transactions, miners sort by fee-per-byte to maximize profitability within the limited block space. This sorting can lead to inefficiencies and vulnerabilities, such as MEV attacks, where miners exploit their ability to reorder transactions for additional profit.

To address these challenges, 0xVM has implemented an execution priority bidding system, allowing users to directly encode their willingness to pay gas fees within their transactions. Both the bids for gas fees and the original transaction order within Bitcoin blocks are considered. This hybrid method ensures that transactions are processed not only based on miner preferences but also allows users to bid to accelerate execution, thereby creating a dynamic market for transaction processing. By leveraging its unique sorting mechanism and decoupling from Bitcoin's original sequencing, 0xVM helps distribute the benefits of gas auctions more evenly among network participants and mitigates risks associated with gas auctions and MEV attacks. (Fig. 10) shows the operation process of transactions in the network. When the transaction2 fee is insufficient, the 0xVM network provides a liquidity pool to facilitate the dynamic swap of gas and ensure the success of the transaction.

The 0xVM transaction processing example:

- **Arbitrage Opportunity Detected:** An arbitrageur detects a price discrepancy for Token A between two DEXs.
- **Arbitrageur Submits Transaction with Gas Bid:** The arbitrageur submits transactions to buy Token A on DEX1 and sell on DEX2, but this time includes a bid for gas fees within each transaction to prioritize execution.
- **0xVM Node Prioritizes Based on Gas Bid:** The 0xVM node processes transactions based on the encoded gas fee bids, not merely the fee-per-byte model. The arbitrageur's transactions, with higher gas bids, are prioritized and executed first, preventing the miner from front-running the trades.

In 0xVM, the implementation of a market-driven transaction sorting process represents a novel paradigm for blockchain execution layers. By decoupling execution priority from the original Bitcoin transaction sequence and introducing a gas fee bidding mechanism, 0xVM provides an adaptive and economically efficient environment for transaction processing. This system not only improves the utilization of network resources but also prevents the concentration of auction profits and potential developments related to MEV. Furthermore, by establishing its own fair transaction sorting mechanism, 0xVM ensures a more balanced and equitable distribution of network interests, thereby fostering a more resilient and user-centric blockchain ecosystem.

6 Future Optimization

As the 0xVM system continues to evolve, identifying and implementing advancements in its operational mechanisms remains pivotal. We explore refined protocol

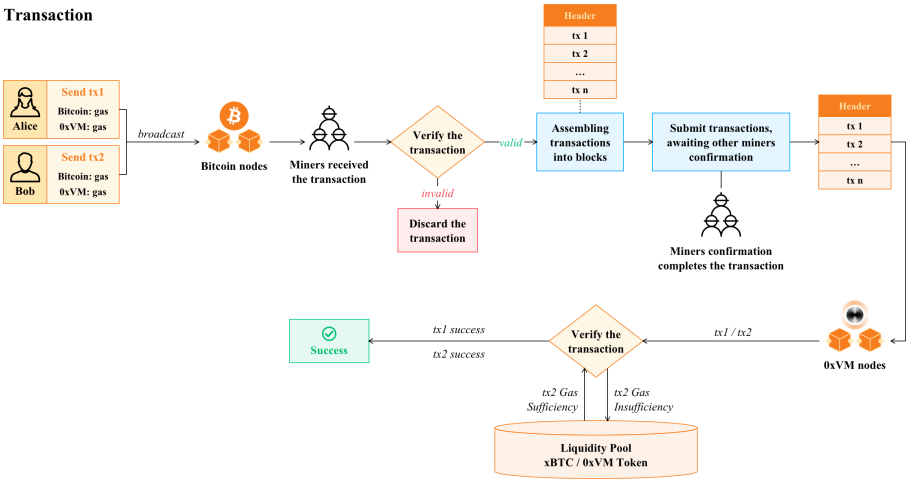

Fig. 10. 0xVM transaction orders and priority gas auction

updates that ensure the system's adaptability (§6.1), innovative approaches to encoding that optimize data processing (§6.2), and the adoption of parallel VM architectures to meet escalating transaction demands (§6.3). Each of these optimizations is designed to fortify 0xVM's infrastructure, ensuring it remains robust and scalable in the face of evolving technological demands and increasing network loads.

6.1 Protocol Update

In the 0xVM ecosystem, the protocol update mechanism is designed to be both flexible and democratic, enabling seamless evolution of the system in response to emerging requirements and consensus among participants. This approach resembles the functionality of an ABI, where changes to the protocol are encapsulated within transactions and committed to the blockchain, ensuring transparency and immutability.

Update Mechanism

The protocol update process in 0xVM is initiated through a special type of transaction, known as a protocol update transaction. This transaction carries the new protocol rules or code changes that need to be applied to the 0xVM system. For an update to take effect, it must garner the approval of a majority of the network participants, reflecting a consensus-driven approach to system evolution.

The steps for a protocol update are as follows:

1. **Proposal Submission:** A participant or a group of participants proposes a change to the 0xVM protocol by creating and submitting a protocol update

transaction. This transaction details the proposed changes and justifications for the update.
2. **Validation and Verification:** Upon submission, the proposal is disseminated across the network, where nodes validate the transaction's integrity and verify the authenticity and necessity of the proposed changes.
3. **Consensus Building:** Participants in the network review the proposal and engage in a consensus process. If a majority agrees on the update, the proposal is approved for implementation. The consensus mechanism ensures that only widely accepted changes are enacted, preventing unilateral or contentious modifications.
4. **Protocol Upgrade:** Once consensus is achieved, the protocol update transaction is appended to the blockchain. The update logic contained within the transaction is then executed, automatically applying the changes to the 0xVM system. This automated execution ensures that the protocol upgrade is uniformly adopted across the network, minimizing discrepancies and potential forks.
5. **System Adaptation:** Following the upgrade, nodes in the 0xVM network automatically adapt to the new protocol rules, maintaining the network's operational continuity and integrity.

This flexible yet structured approach to protocol updates allows the 0xVM ecosystem to dynamically adapt to changing technological landscapes and community needs, fostering an environment of continuous improvement and innovation. Through this consensus-based update mechanism, 0xVM ensures that it remains a cutting-edge and user-responsive system, capable of addressing the evolving challenges and opportunities within the blockchain domain.

6.2 Encoding Efficiency

As part of the future updates for the 0xVM system, enhancing encoding efficiency for transactions is a key objective. The aim is to minimize the transaction encoding size, potentially employing Huffman encoding to achieve theoretical optimality.

Rationale and Approach

The initiative to streamline transaction encoding is driven by the need to optimize network and storage resources. By implementing a shorter, more efficient encoding method, such as Huffman encoding, the 0xVM system can reduce the data footprint of transactions, leading to improved network throughput and storage efficiency.

Implementation of Huffman Encoding

Huffman encoding, a widely recognized method for lossless data compression, can be adapted to optimize transaction encoding in 0xVM. This algorithm assigns shorter binary codes to more frequent elements and longer codes to rare elements, thus reducing the average length of the encoded data.

Steps for Enhancing Encoding Efficiency:

1. **Frequency Analysis:** Conduct a comprehensive analysis of transaction data to determine the frequency of various elements within transactions.
2. **Huffman Tree Construction:** Utilize the frequency data to construct a Huffman tree, which will serve as the basis for the new encoding scheme.
3. **Optimized Encoding Scheme:** Implement the Huffman tree in the transaction encoding process, ensuring that each transaction element is encoded efficiently.
4. **System Integration:** Seamlessly integrate the new encoding scheme into the 0xVM transaction processing system, maintaining backward compatibility.
5. **Network Update:** Roll out the updated encoding scheme across the network, achieving consensus among participants for adoption.

Anticipated Outcomes

The adoption of Huffman encoding for transaction data in 0xVM is expected to yield significant benefits:

- **Reduced Transaction Size:** The overall size of transactions will decrease, leading to lower storage and bandwidth requirements.
- **Increased Network Efficiency:** With smaller transactions, the network can handle a higher volume of transactions, enhancing throughput.
- **Scalability and Performance:** Improved encoding efficiency directly contributes to the scalability and performance of the 0xVM network.

This future update to encoding efficiency, particularly through the adoption of Huffman encoding, signifies a strategic enhancement to the 0xVM system's data handling capabilities, fostering a more scalable, efficient, and high-performing network environment.

6.3 Parallel VM

In an effort to manage the increasing TPS rate, the 0xVM system could significantly benefit from adopting a parallel processing methodology. This can be effectively realized through state sharding, where the blockchain's state is partitioned into multiple, independent segments known as shards. Each shard is responsible for handling a specific subset of transactions. This division enables multiple transactions to be processed simultaneously across different shards, drastically improving the system's overall throughput.

Particularly, when a transaction is executed within the 0xVM framework, it will specify certain state shards that it interacts with. By designating these shards, transactions that operate on different state shards can be processed in parallel. This not only maximizes transaction processing speed but also optimizes network resources. Inspired by blockchains like Solana, which utilize a finely sharded structure to achieve remarkable levels of parallelism, 0xVM can adopt a similar approach. Implementing such a model not only enhances processing

capabilities but also maintains core blockchain attributes such as security and decentralization.

The integration of a sharded architecture in 0xVM will thus allow for a scalable, efficient, and high-performing blockchain network. This development is instrumental in supporting a higher volume of transactions and promoting a thriving ecosystem for DApps, all without relying on mechanisms like Proof of History. This positions 0xVM to address the dual challenges of scalability and performance in the burgeoning landscape of blockchain technology.

7 Conclusion

0xVM, a Turing-complete VM integrated directly into Bitcoin's consensus layer, addresses the limitations of existing layer-2 solutions with enhanced security and decentralization. By leveraging Bitcoin's robust ledger, 0xVM ensures secure, immutable record-keeping and reduces transaction overhead through an innovative account mapping algorithm. A novel transaction ordering mechanism prioritizes VM gas fees to mitigate MEV issues. Future work will focus on improving transaction encoding efficiency and enabling parallel processing to boost scalability. 0xVM significantly enhances Bitcoin's utility and opens new possibilities for DApps on its platform.

Bibliography

1. Muneeb Ali. Stacks 2.0 apps and smart contracts for bitcoin. *Recuperado el*, 8, 2020.
2. Davide Crapis, Edward W Felten, and Akaki Mamageishvili. Eip-4844 economics and rollup strategies. *arXiv preprint arXiv:2310.01155*, 2023.
3. Vikram Dhillon, David Metcalf, Max Hooper, Vikram Dhillon, David Metcalf, and Max Hooper. Unpacking ethereum. *Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You*, pages 37–72, 2021.
4. Marc Jansen, Farouk Hdhili, Ramy Gouiaa, and Ziyaad Qasem. Do smart contract languages need to be turing complete? In *Blockchain and Applications: International Congress*, pages 19–26. Springer, 2020.
5. Ningran Li, Minfeng Qi, Qin Wang, and Shiping Chen. Bitcoin inscriptions: Foundations and beyond. *arXiv preprint arXiv:2401.17581*, 2024.
6. Robin Linus. Bitvm: Compute anything on bitcoin. *BitVM white paper. URL https://bitvm.org/bitvm.pdf*, 2020.
7. mystenlabs. The sui smart contracts platform. *Whitepaper*, 2023.
8. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.
9. Jonas Nick, Andrew Poelstra, and Gregory Sanders. Liquid: A bitcoin sidechain. *Liquid white paper. URL https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf*, 2020.
10. Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.
11. Nicoló Vallarano, Claudio J Tessone, and Tiziano Squartini. Bitcoin transaction networks: an overview of recent results. *Frontiers in Physics*, 8:286, 2020.
12. Wenhao Wang, Lulu Zhou, Aviv Yaish, Fan Zhang, Ben Fisch, and Benjamin Livshits. Mechanism design for zk-rollup prover markets. *arXiv preprint arXiv:2404.06495*, 2024.
13. Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
14. Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain v0. 8.13. *Whitepaper*, 2018.

A Appendix

A.1 Glossary of Terms

UTXO Unspent Transaction Outputs

VM Virtual Machine
ZK Zero-Knowledge
MEV Miner Extractable Value
DeFi Decentralized Finance
DApps Decentralized Applications
DAO Decentralized Autonomous Organization
PoH Proof of History
DDOS Distributed Denial of Service
ABI Application Binary Interface
xBTC Wrapped BTC In 0xVM
TPS Transactions Per Second
DEX Decentralized Exchange