MVC Technical Documentation

MVC Labs

https://microvisionchain.com

2023 March

Version 1.1

Abstract

The major problem of the blockchain industry has always been the Scaling problem. Existing blockchains proposed various innovative solutions with different technical routes in the past, for example, the Sharding solution of Ethereum. Meanwhile, due to the high concurrency of UTXO, some blockchains adopt the UTXO design to scale. Yet, the traditional UTXO model is considered to lack Layer 1 smart contracts capabilities and cannot satisfy complex business needs; it also lacks a decentralized identity protocol (DID) and data structure model of Layer 1, making it challenging to build complex applications and achieve data interoperability. Consequently, the traditional UTXO model blockchain cannot meet the future needs of Web3 applications.

MVC is a revolutionary blockchain based on the UTXO model. It altered the Layer 1 TXID generation method to improve the parallel performance significantly. It also innovatively proposed the Layer-1 smart contract and DID on the UTXO model. Unlimited scaling can be achieved with a high level of decentralization simultaneously. This document will demonstrate MVC's concept, technical details, and mining mechanism, and explain why and how MVC was the perfect blockchain for Web3.

Disclaimer

This white paper is a conceptual document illustrating the MVC's background, technical principles, and essential related content. Any content in this white paper does not constitute legal, financial, business, or taxation suggestions; please consult your law, finance, tax, or other professional consultants if necessary.

Introduction

Why We Build MVC?

After 14 years of development, blockchain technology has showed its unique technical advantages and the potential to change the society. However, the scaling problem has always been the major showstopper to the blockchain industry. Scalability issues also bring a series of problems, such as high fees, low performance, limited application scenarios, etc to the existing blockchain. With the development of technology and people's expectation for the future Metaverse, we need a new blockchain that can fundamentally solve this problem and can support numerous Web3 applications in the future. MicroVisionChain (hereinafter referred to as MVC) is the revolutionary blockchain that was born for Web3 with unlimited scalability.

The Main Problems of Blockchain Technology Nowadays

- Scaling

The main problem of today's blockchain is scaling, prohibiting blockchain technology from being widely utilized by billions of users. It is also the main obstacle to the vast adoption of blockchain technology for society, causing the poor performance of most Web3 applications.

- High Transaction Fees

The transaction fee of traditional blockchains is generally high, which is a derivative problem of the scaling. Some blockchain claim to be able to reduce the fees, but often at the expense of anti-decentralization. The high fees had built a gap between blockchain applications and massive user adoption.

- Cross-chain Storage

To solve the high fees, low performance, and scaling problems in today's blockchain, cross-chain storage solutions are proposed. However, cross-chain cause explicit concerns like poor security and data isolation, leading to poor experience in usage and high development cost.

Besides, existing blockchains are flawed in data interoperation, smart contracts performance, and transaction confirmation time. MVC aims to propose innovative and practical solutions to tackle these issues.

The MVC Solution

- High-level Decentralization

MVC adopts the same POW consensus solution and the same SHA256 mining algorithm as Bitcoin to ensure that the MVC is open, permissionless, and decentralized. Decentralization is the foundation of all blockchain technology that provides the health and scalability of the system. It is also the boost of fair competition to bring the entire system continuously vibrant.

- High Performance

MVC has upgraded the transaction format of the UTXO model to create a high-performance parallel computing supernode to ensure one million TPS. With an optimized UTXO model that enables high concurrency of transaction confirmation, MVC will be the first scaling blockchain used by global users.

- Built-in Distributed Identity Protocol

We invented the MetaID, a distributed identity protocol under the UTXO model to empower discrete UTXO models with the advantages of the account model and high concurrency. Each piece of data links to a specific user. Users can truly own and meanwhile interoperate their data between apps. MetaID builds the foundation for high-performance web3 applications with simplified procedures and lower costs.

- Layer-1 UTXO-based Smart Contract

MVC is the first blockchain to support Layer-1 Turing-completed smart contracts based on the UTXO model. It can achieve the full logic of EVM but with high performance and extremely low transaction fees, thanks to the advantages of the UTXO model.

MVC Design Goal

Web3 is the next generation of blockchain-based Internet that satisfies everyone's expectations for the Metaverse. It is the prerequisite of Metaverse and will be successful only if it can be used by massive users simultaneously, efficiently, and economically. Undoubtedly, the underlying public blockchain must be top-performing to meet the various needs. In addition to the high performance, built-in DID, and Layer-1 smart contract, MVC is equipped with the following features that satisfy billions of potential users.

- Proof of Work consensus and UTXO data model
- Unlimited scaling
- Fee decrease mechanism with the growth of users
- Zero confirmation of transactions
- Unity of data storage, smart contract execution and transaction in one chain
- New MetaTXID solution that solves the tracking problem of the UTXO model
- Built-in distributed identity protocol MetaID
- Layer-1 smart contract MetaContract

About MetaTXID

MVC Adopts a New Transaction Identifying Method,called "MetaTXID." It Is the Core Technology that Differentiates MVC from Other UTXO Public Blockchains.

Introduction

MetaTXID is a special transaction identifier generated by a hierarchical hash algorithm during the data uploading on-chain.

Its hierarchical hash calculation on data segments can be conducted without altering the original data set and transmission format. Therefore, transaction data on different hierarchies can be pruned separately without interfering with the hash verification of other parts of the data. In short, transaction identifiers can be calculated with the hash value of the pruned data and other un-pruned data.

The same prunable data based on the same hash pointer in different transactions can be collectively used. Thereby data can be reused, and the efficiency of data storage and transfer will be significantly improved.

Significance of MetaTXID

The MetaTXID solution proposes an on-chain method that can prune the data in the transaction hierarchically, which solves the inefficiency in data reuse in existing blockchains. Essentially, it makes the Layer 1 smart contracts to deploy on pure UTXO models possible without compromising its parallel capabilities.

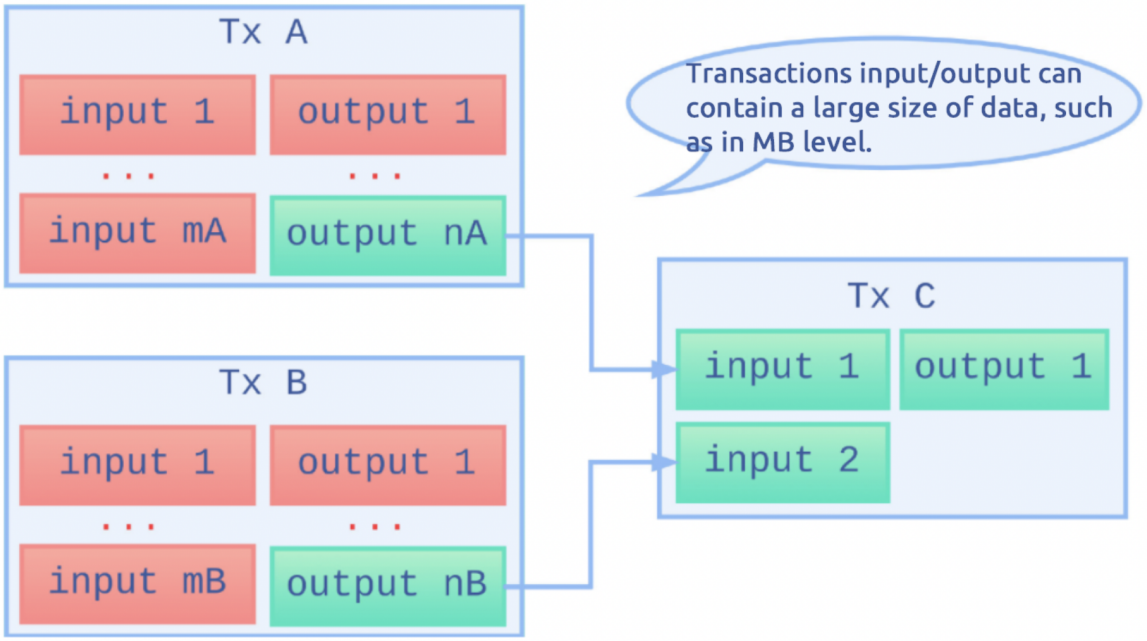
In the existing blockchain technology, the transaction ID (TXID) is generated through hash calculation of the entire piece of data, which leads to redundant data reception during transaction verification by nodes or contracts using a hash algorithm. This method unnecessarily increases the cost of data storage, transfer, and reuse, meanwhile bringing a burden to the user experience.

In order to tackle the problem of inefficiency of intra-transaction data reuse in current blockchain technology, MetaTXID proposed an innovative on-chain method of pruning the Intra-transaction data hierarchically.

In particular, when applying MetaTXID transaction identifiers to the UTXO Smart Contract, the data size required in the verification process can be significantly reduced since the irrelevant data can be pruned during the smart contract verification on the targeted data. This is an more elegant solution for data traceability and a solution for Layer1 smart contract on the UTXO blockchain.

Technical Mechanism

Under the UTXO model, the relationship of blockchain transaction data is shown in the following figure:



Because a large number of transactions in the UTXO model can be validated concurrently, and a single block already has a transmission capacity of GB level, a single transaction can contain MB-level data in fact.

Even though large blocks allow extremely low-fee transactions, users and developers need to minimize the data volume as much as possible. As shown in the figure above, transaction C will refer to some output data in transactions A and B without considering other parts of the data.

When the Simplified Payment Verification (SPV) lightweight nodes are implemented correctly and efficiently, irrelevant transactions in transactions A and B need to be deleted, and the verifiability of the required nA and nB data needs to be ensured at the same time. It means that the pruned data still belongs to transaction A and transaction B and those data should be included in the TXID calculation with their hash value, which is concluded as the

MetaTXID hierarchical data pruning calculation. Based on this method, the data size of SPV lightweight nodes can be significantly reduced based on the original Bitcoin design.

MetaTXID's hierarchical data pruning model is the technological premise of the smart contract verification between data sets of transactions A, B and C. Without the data pruning verification, output of transaction C will contain all the data of transactions A and B, this will dramatically increasing the gas fees on UTXO model blockchains and cause exponential growth of data size when the blocks are continuously mined. The MetaTXID method is an elegant solution to the UTXO model that avoids the use of address-based technology and impacton transaction parallel verification.

Specifically, TXID transaction format is optimized as follows:

```
TXID = SHA256_Encode(SHA256_Encode({
        Version,
    LockTime,
        InputCount.toString(8Byte,Little),
    OutputCount.toString(8Byte,Little),
        SHA256_Encode ({
            TxIn1:TXID, TxIn1:VOUT, TxIn1:Sequence,
            TxIn2:TXID, TxIn2:VOUT, TxIn2:Sequence,
            … ,
            TxInN:TXID, TxInN:VOUT, TxInN:Sequence,
        }),
        SHA256_Encode ({
            SHA256_Encode ({ TxIn1:UnlockingScript }),
            SHA256_Encode ({ TxIn2:UnlockingScript }),
            … ,
            SHA256_Encode ({ TxInN:UnlockingScript }),
        }),
        SHA256_Encode ({
            TxOut1:Value,
```

```
                    SHA256_Encode ( TxOut1:LockingScript ),
                    TxOut2:Value,
                    SHA256_Encode ( TxOut2:LockingScript ),
                    … ,
                    TxOutM:Value,
                    SHA256_Encode ( TxOutM:LockingScript ),
            }),
        }))
```

MetaTXID not only satisfies the data pruning of nodes toward single UnlockingScript, but is also capable of pruning all the input UnlockScript data while maintaining the verification ability towards other kinds of data like LockingScript.

Features of MetaTXID

- Due to the irreversibility of TXID calculation, it can only be used in indexing TX data since different transactions will generate different TXIDs. Therefore the calculation of TXID is low coupling and highly independent, and the new calculation logic will be applied only when it is needed. During the calculation, other programming logicincluding validation of the full script of transaction data (Key & Lock) used by minders and the process of SPV(Simplified Payment Verification), will not be affected.
- Different TXID calculation functions will be adopted with the upgraded TX version to identify old and new versions of transactions. Two kinds of transactions can coexist without conflict.
- Developers do not need to alter any of the verification units to upgrade the Application. All they need to do is to add a new TXID calculation function.
- This solution follows the prunable dataset design from Merkle Tree. It is applied to every transaction and generates the needs of the completed data pruning. In particular, once the Key data is verified and uploaded on-chain by miners, SPV will maintain higher security and meanwhile allow giant Key data to be removed safely.

- This solution is essential in the path of Big-Block Blockchain development. Big blocks require massive transactions and high efficiency of script execution. When the script program gets larger, the storage needed (Tapes in a Turing machine) is also getting bigger, which is similar to the phenomenon that modern computing programs need to recycle memory to maintain the functionality of the Turing machine. MetaTXID follows the same spirit and takes full use of the limited Turing Tape, ensuring to recycle resources by data pruning to the full extent.

- MetaTXID enhances the transaction data ability to succeed and evolve, empowering the bitcoin system to become a biologically evolving organism. It is similar to the natural phenomenon that when diploid cells undergo meiosis to produce haploid gametes (sperm and egg), part of the chromosomes is allocated to the polar bodies, which are usually degraded.

  Without discarding some chromosomes, the organism will not be able to evolve. Therefore, the ability to dispose and recycle is a must.

  For any bitcoin transaction requiring self-evolution, its mutation ability is realized by Turing's complete computer instructions, while the recycling ability (data pruning and storage management) is realized by its proposal.

More information of MetaTXID: https://www.microvisionchain.com/development/metatxid

About MetaID

MVC has a built-in native distributed ID protocol (DID), called “MetaID”. It is the first Layer1 DID implemented in a UTXO public blockchain. DID solution is a vital underlying protocol for Web3 applications. Only through DID can the complexity and cost of developing Web3 applications on MVC be remarkably reduced, which is the prerequisite of a large-scale popularization for Web3 applications.

Introduction

MetaID is a distributed identity protocol based on blockchain that aims to facilitate the development of Web3 applications. It is a cross-chain unified identity protocol realizing data interoperability between applications.

MetaID has the following features:

- Users can use all MetaID-based applications with only one private key;
- The user's basic information and the transaction data are recorded on the nodes and owned by themselves. Users’ data is independent of the wallets and applications, breaking the data monopoly in Web2;
- MetaID converges discrete UTXO transactions to a vertex. All Web3 data are allocated to specific users according to their actions. Users own their data this time.
- The data between different applications can be interoperable and reused, eliminating the information barriers between Web2 mega-corporations;
- The data of different protocols can be combined with each other and reused with MetaID therefore the workload of Web3 application development is significantly reduced;
- MetaID can be deployed in multiple UTXO model blockchains. User data is distributed in multiple blockchains and converges into a unified tree-structure dataset.

MetaID Protocol Format

The overall MetaID protocol format is:

```
<chainFlag><P(node)><(parentChainFlag:)txID(parent)><metaIDFlag><nodeName><dat
a><encrypt><version><dataType><encoding>
```

The first three components point to the node position of MetaID, and the last seven components contain the node information of MetaID. MetaID is created using the OP_0 OP_RETURN opcode in the transaction output of the UTXO model.

- The Node Positioning Components

The first three components are

```
<chainFlag><P(node)><(parentChainFlag:)txID(parent)>
```

Detail explanation:

1. < chainFlag > records which chain the data is on.
2. < P(node) > is the public key of the node
3. < parentChainFlag:）txID(parent) > is the transaction ID of the parent node. “parentChainFlag” records which chain the parent node is on. If the information of parentChainFlag is empty or missing, the “chainFlag” will be used to represent that the parent node is on the same blockchain with the current node.

- The Node Information Components

The last seven components are

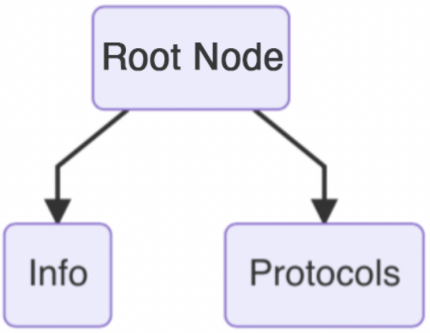
```
<metaIDFlag><nodeName><data><encrypt><version><dataType><encoding>
```

Detail explanation:

key	value
metaidFlag	**Fixed as "metaid".**
nodeName	The node identification name.
data	The data content corresponds to the storage node.
encrypt	Identifies whether or not the contents of the node are encrypted. The current version protocol supports two methods of encryption: 0 represents no encryption; 1 represents the ECIES (Elliptic Curve Integrated Encryption Scheme), using the corresponding node's public key to encrypt, and the private key to decrypt. The default setting is 0, which means no encryption.
version	It is the version of the node type. Different versions represent different formats of the data content.
dataType	This item is optional. It represents the corresponding data type. For available data types, please refer to: https://www.iana.org/assignments/media-types/media-types.xhtmls The default setting is text/plain.
encoding	This item is optional. It represents the corresponding encoding format of data. For available encoding types please refer to: https://www.iana.org/assignments/character-sets/character-sets.xhtml The default setting is UTF-8.

MetaID Structure

In the current version protocol, there are only two sub-nodes under the MetaID Root Node, respectively "Info" and"Protocols".



- Info is the node of users' basic information recording the user's name, profile picture and other basic information.
- Protocols is the node of protocol recording transactions generated by users using relevant protocols.

MetaID is fixed with the node format with Root, Info and Protocols. The names of these three nodes can not be changed and also do not accept version update operations.

We appoint that MetaID is the TransactionID of the Root Node.

- **Root Node**

The Root Node is the vertex of MetaID. When a MetaID transaction is generated, NULL TxIDparent is the vertex.

When the Root Node is constructed, the nodeName is fixed as “Root”. Here's an example of constructing a vertex:

```
OP_0 OP_RETURN mvc <P(node)> NULL metaid Root NULL NULL NULL NULL NULL NULL
```

TxIDparent needs to be NULL in a vertex node. Root Node should be named as Root, and the remaining contents can all be set as NULL.

The public key of MetaID Pnode is generated based on the HD solution [2] , and the Root Node address is based on the 0/0 path of the HD solution.

- **Info Node**

The Info Node is the node that stores the user's basic information, and the nodeName is fixed as “Info”.

The constructed Info Node is similar to Root Node; The only difference is that TxIDparent needs to point to the Root Node and the nodeName should be “Info”. Here is an example of building an Info Node:

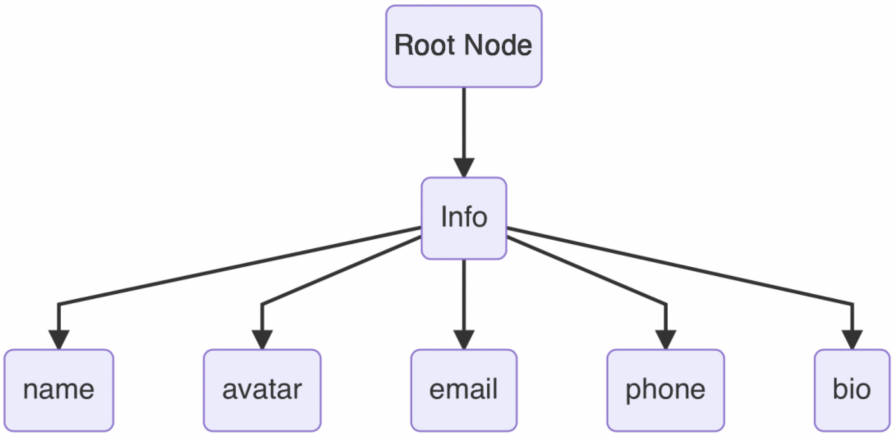
```
OP_0 OP_RETURN mvc <P(node)> <parentChainFlag:Root TxID> metaid Info NULL NULL NULL NULL NULL NULL
```

Basic Information about the User

In the current version protocol, the basic user information is under the Info Node. The Info Node will constantly contain the following five sub-nodes:

- name: User name, which is not recommended for encryption. The format is fixed as text/plain.
- avatar: User profile picture, which is not recommended to be encrypted. It is in binary format, where the node stores the binary format of the image data.
- email: User email, which is not recommended for encryption. The format is fixed as text/plain.
- phone: User mobile phone, which is recommended for encryption. The format is fixed as text/plain.
- bio: User introduction, which is not recommended for encryption. The format is fixed as text/plain.

The structure is as follows：



The above five nodes can be set up by application developers according to different needs. It is acceptable to not create the node or leave it NULL. However, it is recommended that when generating a new MetaID, at least the name node should be confirmed.

The following is an example of building a “Name” node. If you want to set the user's name as “Alice”, you need to create a MetaID transaction and point its parent node to the Info node, then set the nodeName as “name” and the data as “Alice”:

```
OP_0 OP_RETURN mvc <P(node)> <txID(Info)> metaid name Alice 0 1 NULL NULL
```

The Avatar node stores the user's profile picture in a fixed binary format. The structure can be referred as below:

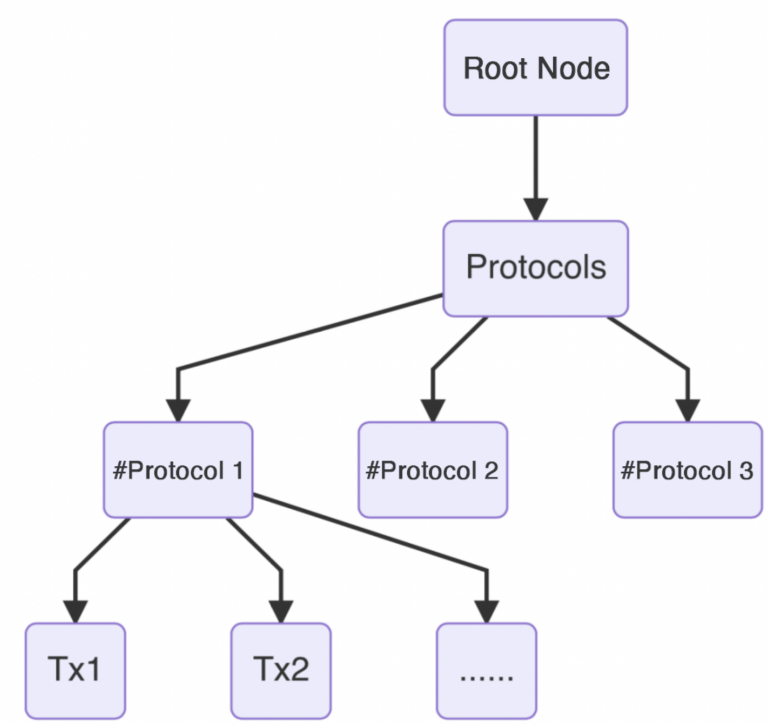
```
OP_0 OP_RETURN mvc <P(node)> <txID(Info)> metaid avatar <IMAGE BUFFER> 0 1 image/png binary
```

Other nodes can be built in a similar way. Please note that some information, such as the phone node, should be encrypted by setting the Encrypt value as 1 and using the ECIES method that encrypts the relevant characters with the public key. The encrypted information can only be decrypted by the user using the corresponding private key.

- **Protocols Node**

The Protocols Node records user transactions using a variety of third-party Protocols. The sub-node under the Protocols Node is a third-party Protocol Node, whose nodeName should

be the protocol name. The sub-nodes under the Protocol Node are the specific transactions generated by the user using the protocol. The structure is as follows:



Since the Protocols are permissionless, any application developers can build their own protocols. Therefore, the number of sub-nodes under Protocols is unlimited, and the structure under each Protocol is determined by the creator/application developer. However, it is necessary to ensure that the identity of Protocol Nodes is unique.

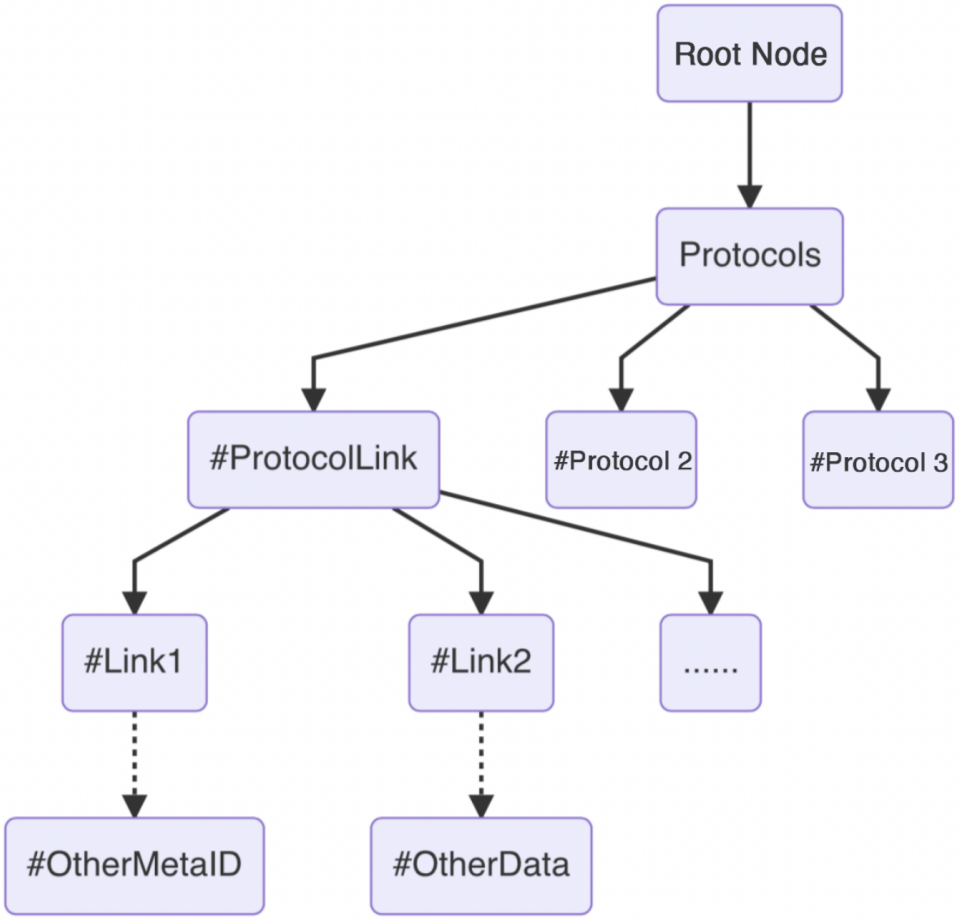
MetaID's Privacy Model

The privacy model of the protocol is developed by the protocol creator/application developer. If users do not want their data to be public, it can be set as follows:

- Set the Encrypt value of the node as 1. It makes the user's data only visible to themselves.
- The data encryption of this protocol is implemented in the way of ECDH (Elliptic Curve Diffie–Hellman Key Exchange). Data is only visible to users and application developers; meanwhile, users can authorize third parties to access their data and the application developers to do so.

Since MetaID is an open-source protocol, all applications can join the MetaID ecosystem without permission. Meanwhile, data itself can be encrypted but its structure can be transparent simultaneously. For those users who do not want any of the interrelation of their data to be uncovered, MetaID provides the Anonymous Node solution.

Via bridge nodes, the protocol can relate one dataset to another; each dataset belongs to a different MetaID or different organizations. This allows users to use Anonymous Nodes or MetaID Protocol Nodes respectively.



More information about MetaID: https://www.microvisionchain.com/development/metaid

About MetaContract

Introduction

MetaContract is a contract framework based on the UTXO model of the MVC Blockchain. Compared with the existing contracts base on the global state, it has the following advantages:

- **Scalability**

Different UTXO contracts can be executed and verified parallelly in MVC nodes. It can take full advantage of the multi-core modern computers to achieve high speed of contract execution, which means the TPS of the MVC contract can be exceedingly high.

- **Low Latency**

By virtue of the MVC's zero-confirmation feature, the result of contract execution can be returned as soon as the TX enters the mempool of the miner nodes without waiting for its confirmation.

- **Security**

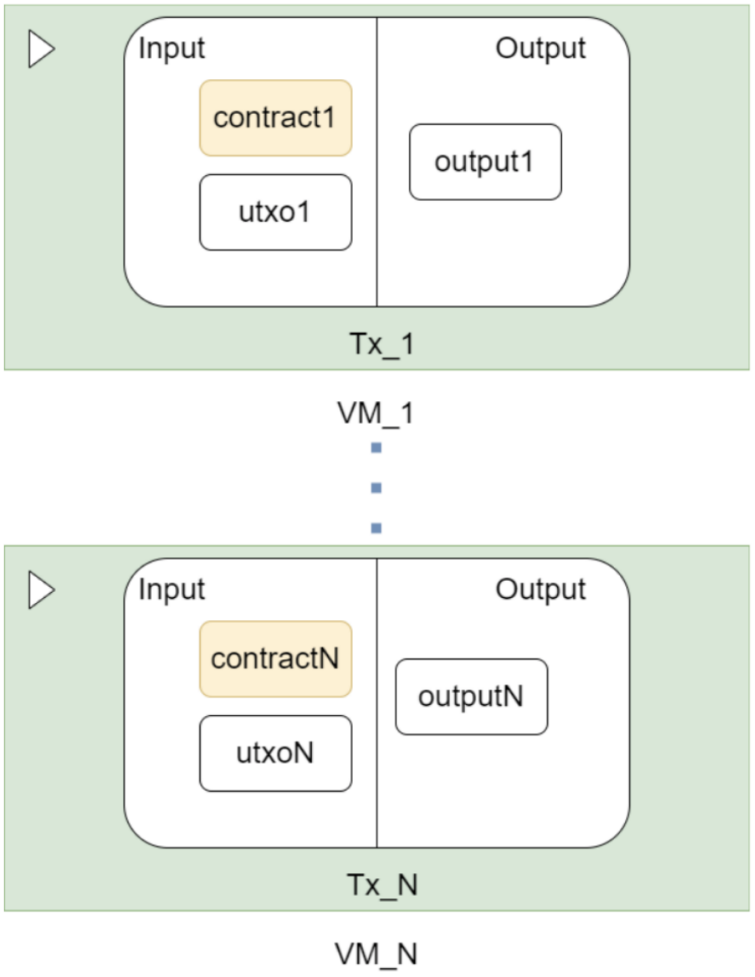
Due to the blockchain-dependency of UTXO contracts, the TX execution sequence of a same contract can be guaranteed, thus making MEV (Miner Extractable Value) impossible. At the same time, as the way UTXO contracts invoke each other is different from the one of those contracts based on Global State, some security threat like Re-Entrancy Attack, will not occur on MetaContract.

Parallel Verification

Since MVC implements the UTXO model, the smart contract execution result is in the transaction output when nodes receive transaction requirements.

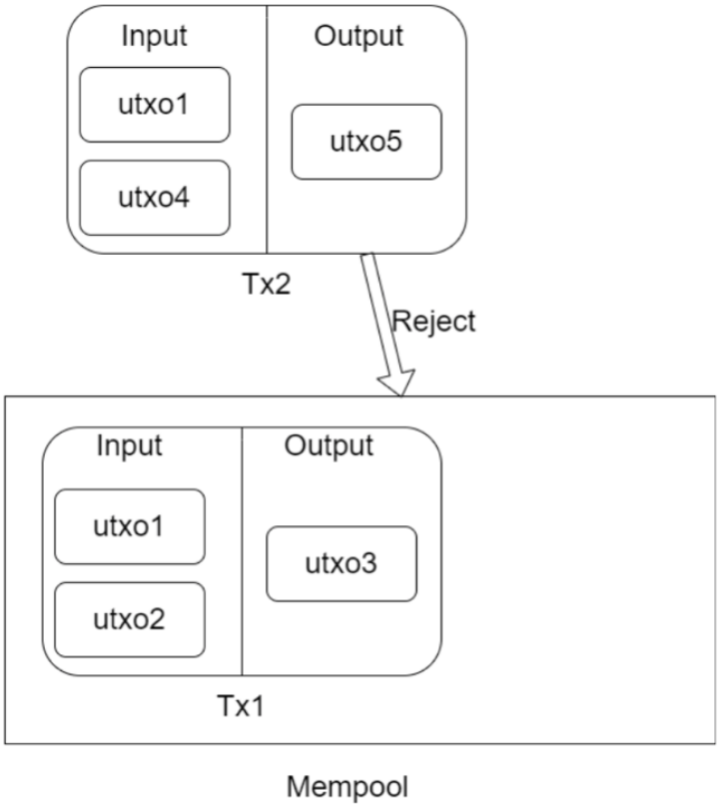
Specifically, nodes only need to verify the correctness of the result, rather than finishing the completed calculation, which indicates that we can proceed with computation off-chain for some computationally intensive contracts and only validate the results on-chain.

Meanwhile, seeing that the virtual machine in the MVC node only needs to execute the validation (read-only operation), there is no side effect (writable operation) for all contract operation, so the validation of the contract in the node can be run in parallel. As shown below, the MVC node can run multiple VM simultaneously to validate different transactions, and takes advantage of the multi-core performance of modern computers.



Zero-Confirmation Security

Since the MVC node's principle for TX sequence follows the First Seen Rule, which is as long as the transaction meets the minimum fee requirement, it will enter the mempool of the node and wait to be bundled in the next block without being replaced by a higher fee transaction. If a later transaction with the same UTXO spending arrives while the first transaction is unconfirmed, it will be verified as illegal by the node. Example:



Transaction 1 is accepted first into the mempool, and transaction 2 then arrives. TX2 and TX1 use the same UTXO1 as one of their inputs. Since TX1 arrives at the node's mempool earlier than TX2, TX2 is rejected by the node based on the First Seen Rule.

In a practical scenario, we can guarantee zero-confirmation security as long as the contract transaction can be broadcast to most of the mining pool nodes.

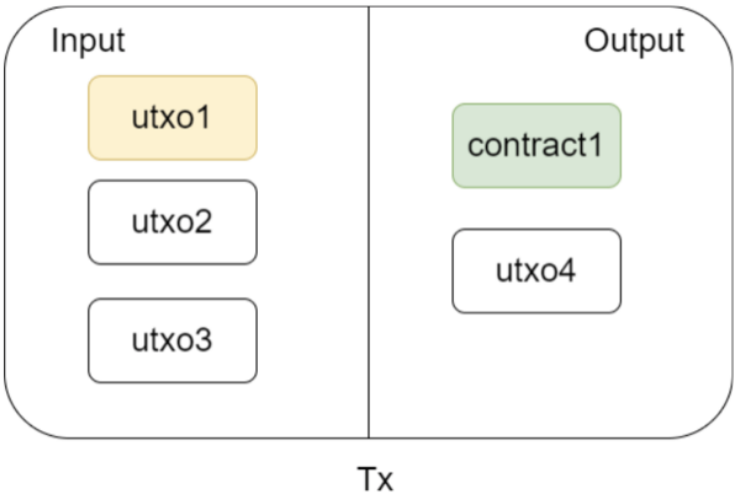
On the other hand, we consider the worst situation, when a malicious double spending attack happens and the transaction is bundled in a block by the mining pool. Tx2 in the above figure is included in a block by the mining pool, resulting in Tx1 being abandoned by the node. In this case, it will only lead to the Tx1 operation to be canceled. In the swap practice, the cancellation will cause the swap to roll back the transaction, and the user's assets will be returned to their original address with no loss. For general types of transactions, users need to decide whether to wait for confirmation based on the value of their orders. If it is a relatively large asset transfer, waiting for confirmation is a safer approach. But for the vast majority of scenarios, zero confirmation is safe enough. Users can choose their security strategy flexibly.

The Globally Unique ID

In the smart contract based on a global state like those in ETH, there are unique global-state addresses to identify every smart contract, and a third party can find the corresponding contract through the address and interact with it. On the contrary, there is no global state in MVC, so all state data is stored under UTXO. The UTXO-based contract will generate an entirely new contract after a single execution, so it requires a new method to confirm the contract ID, enabling third parties to safely find the corresponding contract and interact with it.

- **Contract ID**

Each contract is created with a unique input, combining which with the code hash of the contract is the input ID for the contract.



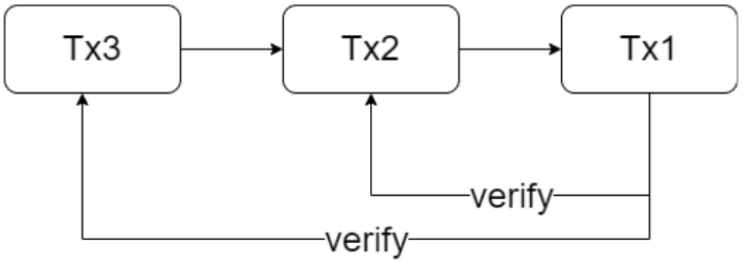
As shown in the figure above, UTXO1 has a unique identifier, TXID1 + OutputIndex1, and Contract1 is the newly created contract. By combining the identifier of UTXO1 with the contract code of Contract1, the ID of Contract1 is created. In practice, the hash (TXID1 + OutputIndex1) + codeHash will be generally used as the ID.

This is the logic of contract ID generation. For fungible and non-fungible tokens（FT or NFT）, there are some additional constraints on this basis due to special requirements (minting).

- **Contract's Back-to-genesis Problem**

Using the solution to the back-to-genesis problem proposed by sCrypt, we can effectively validate a contract's ID to prevent others from forging the same contract ID.

In short, to prevent contract forgery in MVC, the validation towards TX1's parent transaction TX2, and TX2's parent transaction TX3 are required. The validation includes the ID and Code of the contract. The definition of parent transactions is that if the input data of TX1 uses the output of TX2, TX2 is considered as the parent transaction of TX1.



As shown in the figure above, TX1 needs to validate its parent transaction TX2 and grandparent transaction TX3 to check whether or not the contract code and data are consistent with itself.

- **Data Expansion**

During the verification, TX1 needs to obtain the transaction data of TX2 and TX3, which will lead to the size swelling of TX1's transaction data. To prevent the accumulation of TX1's transaction data, MVC implements MetaTxID to hash the transaction input data. As a result, the data needed for this verification is always kept at a small constant number of size, which prevents the infinite expansion of the transaction data.

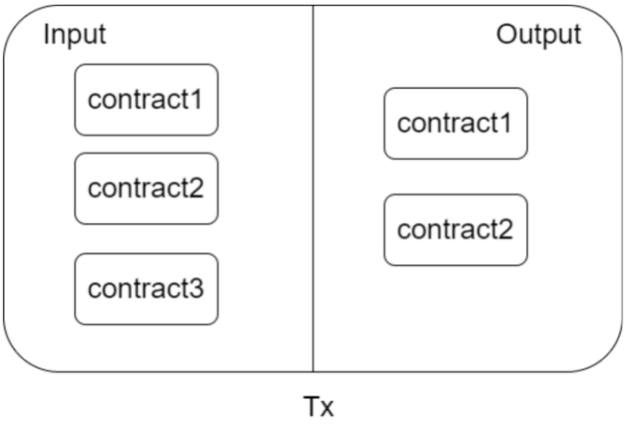
Contracts Composability

The composability of different UTXO contracts can be achieved by validating the contract code and data differently.

The contract in MVC consists of two parts, Code and Data, which are separated by the OP_RETURN operator. The part of Code records the execution logic of the contract, and the part of Data carries the newest state data of the contract.

For a contract, we only need to confirm the execution logic of the contract and its current state to confirm the functionality of the current contract.

Therefore, in the same TX, we can add multiple contracts to the input and read its own TX data, thus to obtain the TX's other input contract code and data and identify the ID of the contract. Based on different contracts, we implement different logic to achieve the combination.



More information about MetaContract:
https://www.microvisionchain.com/development/metacontract

MVC Mining Economy

Miners are providers and guardians of hash power, validators of transactions, and as well as the executor of MVC smart contracts. They are key roles to MVC's decentralization and security. They are also responsible for validating every transaction and smart contract, and play a part in the data storage on-chain. Therefore, they are in a crucial position in the MVC ecosystem.

The MVC network is open and permissionless to miners around the world. Its Mining economy is coherent to that of bitcoin, which means all the bitcoin miners can mine in MVC blockchain directly.

MVC Block Generation

Block Time: About Every 10 minutes

Difficulty Adjustment Algorithm：DAA

Mining Algorithm：SHA-256 (BTC\BCH Compatible)

Block Size Limit: Initial Limit 4G (Scale Dynamically)

Block Rewards

In MVC, the block reward is an incentive for early miners' investment when there are less applications and users. This subsidy will be reduced over time with the flourishing of the ecosystem. In MVC's mining economic model, a majority of the later miners' income shall be transaction fees in the future, rather than the block rewards.

Transaction Fee

In addition to the block rewards obtained from the MVC mining competition, miners can receive fees for validating and confirming transactions. The fee is linked to the data size of the transaction. The larger the data size of the transaction is, the higher fee will be.

MVC aims to carry massive Web3 applications and users. Therefore each block will bring substantial transaction fees to miners in the future even though the average transaction fee is less than $0.01. With the growth of MVC ecology, the transaction fee of each block will surpass the block rewards, becoming miners' main source of income.

Conclusion

In short, MVC is a sufficiently decentralized blockchain with the same level of security as Bitcoin. With innovations including MetaTXID, MetaID, and MetaContract, MVC is taking the advantage of the POW + UTXO model.

We envision that it will be an unlimited scaling blockchain by pruning the data efficiently and elegantly;

We envision that it will bethe blockchain infrastructure for hundreds of thousands Web3 applications;

We envision that it will be the smart contract platform to execute practical, turing-completed smart contracts of any kind..

More importantly, MVC is capable of achieving all these innovations with extremely low fees given the billions of users adoption.

It is the reason why MVC will be the best blockchain to satisfy massive future Web3 applications for global users.

Reference

Satoshi Nakamoto (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.

https://bitcoin.org/bitcoin.pdf

Wenshuai Zhang, Jing Li (2021). Method and system for hierarchically cutting data in blockchain transaction, and storage medium. The University of Science and Technology of China USTC.

https://patents.google.com/patent/CN113360578A/zh?oq=202110682927+.8

Xin Yu Feng, Yu Wang, Qi Ming, He (2022). MetaID v.1.1 Protocol Documentation.

https://www.metaid.io/protocol.html

Jiang Jie, Gu Lu (2021). Sensible Contract, A Feature-Recognition Based Backward-Traceable and Collaborative Contract Model.

https://sensiblecontract.org/files/sensible-contract-v0.2.0-en.pdf