BFMeta

White Paper Pre-Release

Brief Version

More technical details will be included in the full version to be released soon.

Table of Contents

1. Abstract

Metaverse is a trusted digital value interaction network based on web3.0 technology system and operation mechanism support, and is a new web3.0 digital ecology with public chain as its core. It provides immersive experience based on virtual reality technology and builds a new social and economic system based on blockchain technology, which makes the virtual world and the real world closely integrated in the economic system, social system and identity system. The blockchain underlying operating system is the core of the metaverse economic system construction. BFMeta, the public chain of BFChain metaverse built on top of the world's leading open source BFChain blockchain operating system, inherits the excellent performance of BFChain public chain underlying high efficiency, security, high scalability and high carrying capacity, and on this basis, for the fundamental characteristics of metaverse, it provides distributed storage, end-to-end communication, distributed digital identity, distributed credit system, large-scale consumer-grade applications, virtual reality technology interface, cross-chain compatibility, etc. Application-level breakthroughs and innovations have been made, aiming to become a global super metaverse public chain system and support 5 billion Internet users worldwide to step into the new metaverse world.

2. Content Overview

BFMeta is a metaverse super public chain built on top of the world's leading open-source Bioforest chain blockchain operating system. Through its series of disruptive innovations at the level of the underlying core components of the blockchain, it realizes a strong support for the construction of the metaverse economic system. Core features of BFMeta include:

1) High Scalability

Adopting energy-saving, secure and efficient consensus algorithm, combined with native cross-chain, distributed storage, distributed computing and other functions, it can support billions of users worldwide to develop and experience Web3.0 and metaverse applications.

2) Mobile terminal direct connection to the chain

BFMeta breaks the constraint that traditional blockchain technology must run on x86 PC, and extends the application scope of blockchain to mobile terminals, such as Android terminal, IOS terminal, Windows terminal, Linux, Unix and other terminals support direct connection to the chain, and the terminal is the node and the node is the service. Therefore, each individual can directly enter the metaverse economic network system through mobile terminals.

3) Support distributed digital identity DID

Through BFMeta distributed infrastructure, a new type of self-sovereign and verifiable distributed digital identity can be realized. Unlike the centralized platform where the platform controls the digital identity, the digital identity on

BFMeta is in the hands of the user. the advantages of BFMeta distributed digital identity include: highly secure, the user's identity information will not be leaked and the information is held by the user; autonomous and controllable, the user can manage the identity independently and control the sharing of their identity data; portable, the identity identity owners are able to use it wherever they need it.

4) New support for Defi

BFMeta supports both DeFi for various digital assets (Tokens) and DeFi for digital products DPs. DPFi is BFMeta's unique liquidity protocol for digital products DPs, which allows DP owners to obtain secured asset loans from peer-to-peer liquidity providers in a fully de-trusted manner, increasing the liquidity of the DP assets they own. DP liquidity providers use DPFi to earn attractive returns or have access to DPs at a price below their market value in the event of loan default.

5) Digital Asset Interoperability Across Chains

Through homogeneous cross-chain technology as well as heterogeneous cross-chain technology, it enables the issuance and management of digital assets and digital products/NFT, and supports parallel chain, evolutionary chain, sub-chain, multi-chain concurrency and cross-chain asset interoperability.

6) Support for XR and other digital interaction technologies

The unique mobile terminal chain technology can access any XR hardware device to realize the high integration of metaverse immersion experience and economic system and value flow system, and truly realize the leap of XR technology from virtual reality game to metaverse social economy.

7) Complete development tools

It has complete development documents, interfaces and SDK support, and has complete tools for blockchain configuration management, service hall, industry

collaboration and transformation, which can meet the development needs of global developers based on the underlying blockchain.

3. Overall Design

3.1 Open Source Mobile Blockchain System

Traditional blockchain technology applications are generally applicable to the service side with large data computing capacity and high asset. With the gradual exploration of blockchain technology, it is found that the current blockchain applications as well as blockchain technology solutions generally have a problem: it does not support mobile. The upper layer of blockchain applications all require third-party service nodes to provide mobile services. For a blockchain technology based on decentralized design and solving credit problems, it is against the original design intention that a third-party credit intermediary needs to be introduced to provide terminal services.

The future blockchain needs to support mobile terminals. First of all, blockchain is a new generation infrastructure built on top of the Internet to solve the credit problem, and it brings progress. If the applications built on blockchain still require the use of traditional PCs to carry out access, then this is somehow a retrograde step. If blockchain is only used on the server side to solve the problem of inter-institutional credit, it will limit the development of blockchain to a great extent. If the application of blockchain cannot face the end-user and let the end-user perceive the existence of blockchain, then this application is not essentially different from the traditional centralized application.

When the blockchain does not support mobile terminals, it is still possible to build the application on the blockchain and provide mobile terminal services, but this requires additional third-party nodes for providing data transit services between the blockchain and the mobile terminal. This way of introducing third-party nodes

essentially introduces third-party central credit, which is against the original design intention of blockchain decentralization. In a sense, when users cannot directly participate in the blockchain, there is no essential difference from not using the blockchain, and they still rely on central credit to provide services, and this form of blockchain can only be called distributed chain database in a sense.

Therefore, blockchain must support mobile terminals, otherwise when users do not perceive the positive meaning brought by blockchain, its will largely restrict and limit the development of blockchain.

However, there are many problems encountered in the process of doing the design of blockchain for mobile terminal, such as:
(1) Insufficient computing power of mobile terminals
(2) Unstable mobile networks and inability to stay online
(3) limited storage space of mobile terminals

Therefore, how to improve the overall architecture of blockchain has become a challenge that must be solved to address mobile blockchain.

BFMeta has built a mobile-based blockchain system - BFS, known as Bioforest Chain System, which breaks the constraint that traditional blockchain technology must run on x86 PC and extends the application of blockchain to mobile terminals, such as cell phones, tablets, smart wearable devices and IoT devices platforms.

BFS includes: core application components, core blockchain underlying technology components, platform infrastructure and developer community.

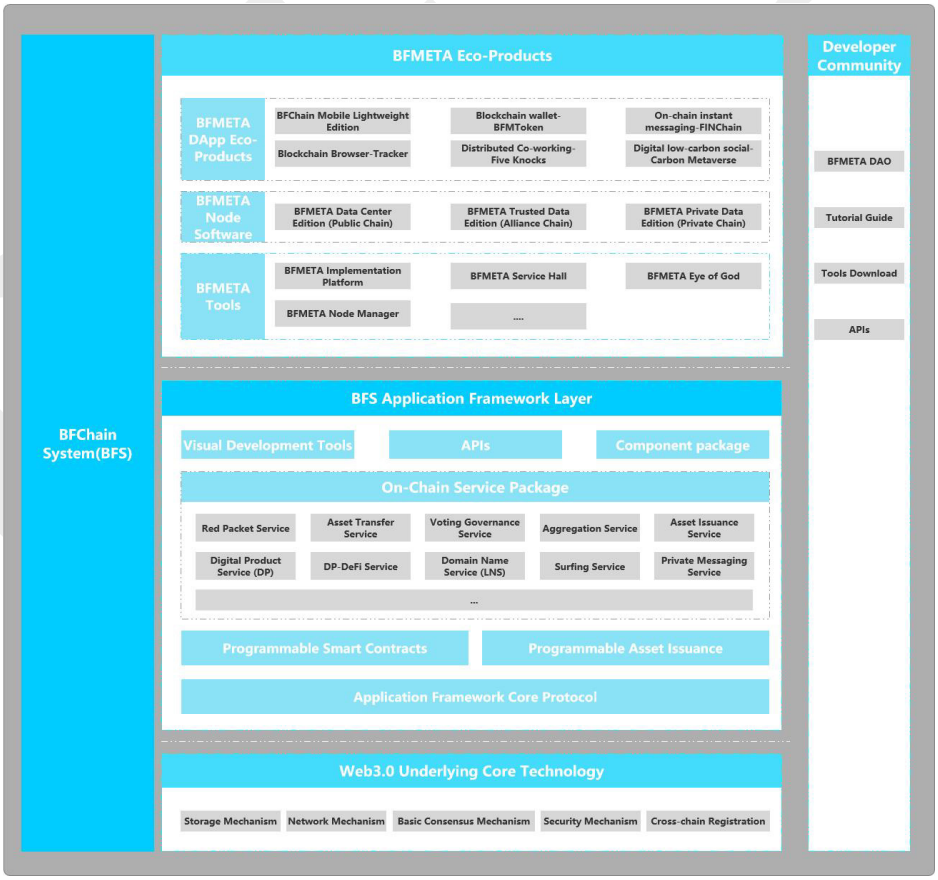
1) Core blockchain underlying technology components: delegated DPOP consensus mechanism, security mechanism, blockchain storage, chain duplex

communication network, multi-chain architecture (cross-chain transactions & blockchain evolution), etc.

2) Core application components: programmable contracts, programmable assets, application framework core protocols, on-chain services, APIs and visual development tools, etc.

3) Platform infrastructure: used to carry the basic requirements for the operation process of blockchain systems, including Android, Windows, Linux, UNIX, macOS and other operating systems.

4) Developer community: BFMeta DAO governance, toolkit download, tutorial services, etc.



BFS Architecture

Next, we will introduce the innovations of each technology of BFMeta mobile blockchain system in detail.

3.2 Blockchain Storage Mechanism

3.2.1 RSD Storage Mechanism

Data storage capacity is the foundation of blockchain landing. Due to the special nature of blockchain, it is impossible to pre-determine and require participating nodes to provide large-capacity and high-throughput data storage devices, so compared with traditional centralized IT construction, the following considerations need to be added during the design.

1) Storage capacity

Traditional IT construction can require the provision of large-capacity disk arrays to increase capacity or separate business and historical data to reduce storage pressure and system performance pressure, but blockchain networks cannot be accomplished by the above two methods.

2) Throughput Performance

In traditional IT construction, nodes can be required to adopt faster storage devices such as high speed and SSD, or adopt Raid1 array disks and increase the number of disks, or spread the throughput pressure to multiple nodes, or preload data to memory, etc. to effectively improve the throughput performance, but it does not work in blockchain scenario because most participating nodes do not have these devices and conditions.

3) RSD Mobile Storage Mechanism

In the traditional IT construction, the system architecture is designed according

to the centralized server, and in order to take into account the cost of the terminal, the terminal does not participate in the business logic calculation, and the terminal also does not store business data, so there is almost no need for mobile storage in the terminal, and the small amount of storage is also user files and personal data information that do not participate in the business logic operation. The traditional storage design approach cannot be applied in the blockchain scenario.

Because of the practical difficulties, most blockchains still follow the traditional IT construction idea in storage design, where data are stored centrally in the participating nodes, and the terminal participants (such as mobile wallets) are relayed through other participating nodes.

Problems of traditional storage ideas

1) Pseudo-decentralization

Because the terminal cannot directly access the blockchain, the actual data request and business process are completed by other nodes, and the trustworthiness of the data is guaranteed by whether the participating nodes cheat or not, which is against the original intention of decentralization of the blockchain.

2) Pseudo-node

If the terminal cannot access the blockchain network, or if it accesses the blockchain network but the data is incomplete, it will not be able to effectively participate in the governance of the network (under the traditional consensus mechanism). Although it looks like a node and can complete indirect communication by means of bridging to a certain extent, it is still not considered a qualified participating node.

3) Inability to participate in consensus

When there is a missing terminal network and data, it will cause the terminal to lose the basis of participation in consensus. Not being able to participate in consensus often means not being able to participate in block building, and this means that the rewards that accompany block building will not be available, which is unfair in a sense to such terminals that contribute the same level of participation.

BFMeta's proprietary RSD mechanism

We have redesigned the way we store blockchain data, and we call it Relational Object Storage (ROS).

For the future development of the BFMeta ecosystem, to encourage more actors and endpoints to access the BFChain network and receive fair rewards, we need to design a solution that takes into account the access problems faced by different endpoints and provides an effective and reliable solution. To this end, we first proposed the concepts of The R-Node (real-time node), The S-Node (service node), and The D-Wallet (distributed wallet) to support high-performance network nodes and distributed service nodes, respectively, and to ensure that they can also participate in the consensus mechanism, we redesigned the data storage mechanism of the blockchain. In order to implement these concepts and ensure that they can also participate in the consensus mechanism, we redesigned the data storage mechanism of blockchain.

Multidimensional sharded storage

BFMeta metaverse data storage not only adopts multiple disks, but also adopts the patented storage technology of "multi-dimensional slice expansion", which allows data to be stored in large quantities while maintaining logical uniformity in

the application layer. The data can be stored in huge amount, but it can keep the logical unity in the application layer. The scaling technology enables qualitative improvement of storage performance and lays a solid foundation for future comprehensive commercial applications.

Memory image storage

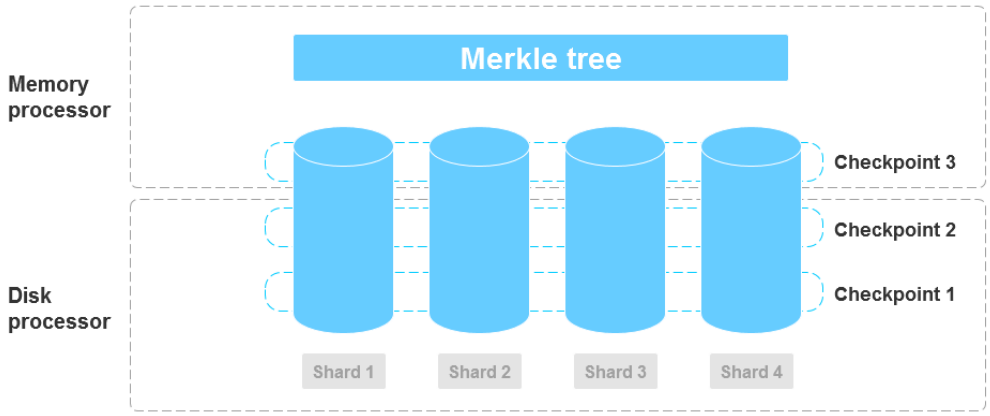
In order to solve the problem of data retrieval efficiency, we introduced Mongodb, a NoSQL database that is currently in common use, mainly using its fast retrieval capability in the case of long chained data. Since it is not small in storage size and not suitable for mobile, we adapted and integrated it with SQLite to allow nodes to participate in consensus while remaining light in size, providing us with a basis for fair rewards.

Checkpoint storage

In the RSD mechanism, two kinds of databases have their own division of labor, and the modified one; for the mobile terminal that needs to participate in the consensus, it needs to store part of the complete block data locally and participate in the consensus mechanism through this part of data. In order to minimize the amount of data stored in the terminal and the length of synchronization, we have established the patented technology of "key checkpoint storage". The terminal no longer needs business calculations and consumes additional calculation time in the synchronization process, but only needs to store the block data after the checkpoint. At the same time, it provides fast start-up capability in case of business node failure. For the establishment of key checkpoints, we use the same consensus mechanism as the block to complete.

Hash tree storage

We use Mongodb to store block hash trees to provide fast identification of forking issues during the execution of the consensus mechanism. The hash tree storage is designed to allow the business to discard burden data during the computation process, allowing direct computation without losing historical information.



3.2.2 Private Data Storage

Using photos and videos to record life is almost one of the necessary parts of people's working life in modern society, and this numerous photo data we often can't carry completely using portable devices, we generally use a cloud device to store these photos and videos of us, but some of our photos and videos may involve the confidentiality of our work and the privacy of our life, so that we are not so comfortable to store in the public In this context, some solutions propose to use offline encryption tools to encrypt photos and videos, but this in turn increases the complexity of sharing these data within a specific range. In addition, since these tools are still provided by the original organizations and the data is still stored by the original organizations, to a certain extent, people can't completely trust whether these tools are really encrypted or whether they have secretly stored an

extra copy before encryption, which makes it difficult to promote the service of private storage when there is a real demand for private photo and video storage. There is still no real credible tool or platform for storing and sharing private photos and videos, and other data such as financial data, diaries, memos, contracts and other data with the same kind of needs are facing the same problem. So, how to store and share private photos and videos securely without increasing the complexity of storage and sharing is an urgent problem.

BFMeta provides a method and its device for secure storage and sharing of private data, by creating group information and permissions, obtaining the group number assigned to the group by the blockchain and assigning member account addresses to the group number, encrypting the public keys of group members using ring signatures and transferring the encrypted data to blockchain transactions, completing the creation and assignment of group permissions, and when storing When the private data is stored, the ring signature string of the group is obtained, and the data to be stored is encrypted using the said ring signature string, and the encrypted data is converted into blockchain transactions and submitted to the blockchain, and when the data is read, the visitor's private key is used to decrypt the data, and the decrypted data is restored according to the original data type, which realizes the role of private storage and limited sharing of data. It solves the problems of data leakage and data custodian's guarding and stealing.

3.2.3 Distributed Storage

In this era of information technology, data is everywhere, and it has become the basis of our modern life and work, but with the development of information, there are also a lot of negative incidents, such as unscrupulous service providers who

sell users' privacy data professionally, and hacker gangs who specialize in stealing users' data from major information platforms, and so on. In the face of such a data crisis, there are quite a number of programs in the world that provide data security protection, such as encrypted storage, and hosted storage; in the encrypted storage program, since the data does not leak depends on the private key not to be leaked, and the private key comes from the service provider, the service provider to provide private key services in the process of strictly do not leak the private key is a new data security problem; in the data hosted storage program, the security of the data depends on the hosted storage In the data hosting storage program, the security of data depends on the technical strength and moral quality of the hosting party, and the repeated data leakage incidents of data hosting platform make people gradually reduce their trust in the technical strength of the hosting party, and the problem of custodian's guarding and stealing from time to time adds to people's worries. The reason behind people continuing to use these services despite the continuous data leakage incidents is that individuals cannot provide such large storage capacity and cannot maintain a data service device that can provide services anytime and anywhere. So how to provide a storage solution that does not require personal maintenance, does not require third-party hosting, and can provide very large scale storage capacity becomes an urgent problem to solve.

BFMeta builds a decentralized distributed data storage method by creating an account on the blockchain and saving the private key, filling in the description information of the uploaded resource, attaching the resource ready to be uploaded as an attachment to a blockchain transaction, completing the upload of the resource by processing the blockchain transaction that is reserved and placing the transaction into the block; starting a node that is in the said blockchain, the node obtains the local file distribution table, detects the blockchain network according to the file distribution table, synchronizes the data resources on the network with higher scarcity than local ones to the local file repository, and repeats the

above steps to synchronize resources every day to complete the synchronization of node resources; accounts that need to download resources select the needed data resources for downloading by looking up the data resources in the network, solving the problem of unlimited and reliable storage of data in the decentralized environment. This solves the problem of unlimited and reliable data storage in a decentralized environment.

3.3 Blockchain Network Mechanism

3.3.1 Full-Link Duplex Communication Network

The blockchain network is the foundation for establishing the whole BFMeta, and the traditional blockchain network basically adopts the socket method.

(1) Socket has the following characteristics

a. Rich component library, supported by programming languages from long ago, such as cobol and c++ in the 1960s and 1970s.

b. Simple logic and easy development, only need to care about the content of the transmission, not the underlying communication logic.

c. Different regional networks cannot interoperate directly and need additional components to support them, such as GRPC.

(2) Problems of Socket

a. Weak autonomous control of network connections means that the data is real-time and the efficiency of the transmission is difficult to control.

b. Socket-based applications cannot communicate directly with browsers, which means that applications with Webkit as a core cannot directly access the blockchain network.

c. The active communication capability for service-to-end is weak, and it is difficult to establish an effective real-time push mechanism.

Higher standard BFMeta network

To achieve direct participation of mobile terminals in the consensus mechanism, we have redesigned the communication domain. We redesigned the blockchain P2P network, which we call it - Full Link Duplex Communication.

For the future development of BFMeta ecosystem, we need to design it in such a way that any endpoint type can easily access us, while most of the current blockchain networks still need a centralized server (e.g. mobile wallets) to provide external services, so we put forward higher design requirements for BFMeta --We redesigned the P2P network because there is no ready-made P2P network to support this.

(1) Introduction of WebSocket mechanism

We introduced the WebSocket mechanism for the scenario of high real-time requirements. It provides the basis for us to provide highly reliable and high-performance BFMeta.

(2) Combination of "HTTP protocol" and WebSocket protocol

In addition, we have introduced the most widely used protocol on the Internet, "HTTP protocol" (we will gradually upgrade to HTTPS later), which is combined with the WebSocket protocol, allowing BFMeta's network capabilities to provide

efficient interoperability not only between nodes, but also across The network capability of BFMeta not only provides efficient interoperability between nodes, but also provides effective interoperability across regional networks and terminal types, supports NAAS, and provides the basis for developing a truly distributed application DAPP.

3.3.2 Node Addressing

With the emergence of Bitcoin, blockchain technology is increasingly recognized as a new layer of infrastructure built on top of the Internet. In the future, there will be countless network nodes in this facility, which are important components to support business operations on the blockchain, and indicating specific operational nodes for business is one of the necessary capabilities in a blockchain network. However, in the actual business operation environment, we need to adjust the actual running nodes of our business due to the limitation of IP allocation by network operators, change of business usage, server scale adjustment, etc., and after this adjustment, it will lead to the nodes that can find our business on the blockchain can no longer find us. If we rely on a third party to find us, it will break the peer-to-peer model of the blockchain and reduce the security; then how to achieve a dynamic addressing capability within the blockchain network without relying on any other third party, so that the business nodes can keep the business continuity no matter how they change and always let the business nodes only need to remember a name to always find us, becomes An urgent problem to be solved.

BFMeta builds a blockchain-based dynamic addressing method and its system, by defining a name protocol for the blockchain, and then handling the business by transferring the ownership and manager of the region, and when addressing is needed, by updating the region where the business node is located and resolving the corresponding region location, and then resolving the region address of the

originating business through this region, the business node can By using the inter-chain protocol module and intra-chain protocol module in the protocol manager to process the data of the blockchain nodes, and through the cooperation of the name management module, ownership management module, sub-domain management module and location resolution module in the region manager, the business nodes can be dynamically addressed, which keeps the continuity of business and improves the business. It maintains business continuity and improves the processing performance of business nodes in the blockchain.

3.3.3 Bluetooth, NFC, AIRDROP Network Transmission

Blockchain is a new distributed infrastructure and computing method that uses block-chain data structure to verify and store data, distributed node consensus algorithms to generate and update data, cryptography to secure data transmission and access, and smart contracts consisting of automated script code to program and manipulate data.

Traditional data connection methods include Bluetooth connection, NFC connection, AIRDROP connection, etc., but they can only exist between the device that sends the data request and the device that is the target of the specific request, and it is not possible for other nearby devices to establish a connection with third-party devices and send data.

Therefore, it is an important research direction in the blockchain field to establish the connection between devices other than the device sending the data request and the third-party device and send the data successfully.

BFMeta builds a blockchain network transmission technology based on Bluetooth, NFC, and AIRDROP, and builds a blockchain network transmission channel

based on Bluetooth, NFC, and AIRDROP transmission, where any node on said blockchain network is connected to the node that needs to send messages to other devices by establishing a topological combination of Bluetooth, NFC, and AIRDROP, but cannot When said node receives a request for sending data, said node calculates the optimal data sending path based on the sending target address and the local topology map and sends it; when the data finally reaches the sending target, the data is sent, and the connection between other devices and third-party devices other than the device sending the data request can be established and the data is successfully sent, solving the problem of two devices that are not directly connected in an Internet-free environment. This solves the problem that two devices that are not directly connected to each other cannot exchange data.

3.4 Blockchain Evolution

With the emergence of Bitcoin, the application scenario of blockchain technology is becoming more and more extensive. Most of the existing blockchains do not support the structure of main chain and evolutionary chain, and the few that do support evolutionary chain are actually different data of the same node. There are some problems with this blockchain structure: for example, the absence of evolutionary chain structure will likely lead to all future businesses being placed on the main chain, which will result in all transaction bottlenecks being backlogged on the main chain, thus leading to a low performance of the actual performance assignable to each business; and the evolutionary chain structure of the same node as the actual storage and processing are in the same node, which will lead to the node becoming more and more massive in the future, and the storage and transaction These problems will become especially prominent when the blockchain carries a larger volume of business.

BFMeta constructs a complete set of independent evolutionary chain operation structure, that is, the main chain makes an exact copy of the node exactly like itself, but does not copy the data and the genesis block, and then generates the genesis block that has a dependency on the main chain according to specific business rules, and sends all such business to the evolutionary chain for processing thereafter, and the evolutionary chain is also deployed on an independent node to run independently. This directly shares part of the business computation and storage consumption to several different nodes, thus greatly improving the business processing capacity and providing a basis for the unlimited expansion of the total amount of computation capacity transactions of the whole network in the future.

With the authorization of BFMeta Evolution Chain, enterprises can quickly customize and develop different business evolution chains according to their business needs and application scenarios to ensure the collaborative development of their multiple businesses. For businesses that do not need to be completely on the chain, enterprises can also selectively on the chain and independently develop DAPP or embedded development on the main chain of BFMeta Biochain.

Each evolutionary chain on BFMeta is interoperable with the main chain, parallel chains and other evolutionary chains, and combined with BFMeta's unique cross-chain technology to realize cross-chain asset transactions between each biological chain. These biological chains together form a living biomass - the BFMeta multi-chain ecogroup. This group realizes that each evolutionary chain is independent of each other and interconnected with the main chain, ensuring that each chain thrives in the ecological soil (BFS mobile blockchain architecture). At the same time, this group will continue to evolve and iterate and update with the participation of developers and users all over the world to solve social problems.

3.5 Cross-Chain Transactions

3.5.1 Cross-Chain Network Interconnection

With the emergence of Bitcoin, the application scenario of blockchain technology is becoming more and more extensive. In the current blockchain design, all kinds of chains are running independently, and chain A and chain B do not intersect with each other and do not communicate with each other, even though a side chain type of technology is used to achieve logical interoperability between chains, there is actually no real data communication between the two chains, but the third C chain is used to establish relationships with A and B respectively to achieve This design has two problems, one is the waste of resources and the other is the inefficiency.

BFMeta constructs a cross-chain network interconnection method, which is mainly used to solve the interoperability problem between different chains. The different chains referred to here include two completely different types of chains, and a main chain, evolutionary chain chain, and side chain of the same type of chain; the interoperability referred to here is the network-level interoperability.

The core design idea of cross-chain network interconnection is to create a generic network layer adapter for different chains and establish the processing logic for each chain in the adapter. This network adapter allows nodes of the same chain to keep their existing work unchanged among themselves, but when it explicitly specifies the need to interoperate with other chains, it sends data directly to the nodes of other chains directly. In addition to explicitly specifying the data sending direction, the network adapter can also play the role of an invisible network highway, because when the first node of the first chain needs to send data to the second node of the first chain, it may happen that these two nodes cannot interoperate directly in different networks, then with the use of the present

invention, the first node of the first chain can send data to the second node of the first chain with the help of the With the present invention, the first node of the first chain can relay data with the help of a node of the second chain that has a common connection with the second node of the first chain, and it does not care about the existence of the second chain in the processing logic of the first chain.

BFMeta not only realizes cross-chain network interconnection, but also solves the problems of low probability of network interoperability, poor efficiency, and duplication and waste of equipment.

3.5.2 Cross-Chain Decoupling

Blockchain is the most important infrastructure in the future credit era, and this infrastructure will be composed of many blockchains, then cross-chain data interaction and cross-chain asset transfer among many blockchains will be a necessary and important part. A single blockchain is originally independent and fully autonomous, but once cross-chain is involved, it may need to rely on the reliability of other chains. chain, it will reduce the reliability of the cross-chain data or assets of the other chain, and if it relies on the other chain, it will reduce its own reliability. The trade-off of reliability in the process of cross-chain interaction of blockchain seems to be an unsolvable problem, and in the future there are bound to be numerous blockchains, and the demand of cross-chain becomes rigid, so how to ensure the reliability of cross-chain interaction data or assets without decreasing its own reliability becomes an urgent problem to be solved.

In order to solve the problem of conflict between cross-chain reliability and its own reliability, BFMeta introduces an in-chain granting mechanism, which is to confirm whether overdraft is allowed or not based on the number of confirmed

blocks of cross-chain data in the chain, and wait for the recovery of cross-chain network reliability during the overdraft period, and if the overdraft exceeds the waiting time, then all transactions down the link will be prohibited, but the transactions of its own chain If the overdraft exceeds the waiting time, then all transactions down the link will be disabled, but the transactions of its own chain will continue to be processed normally until the cross-chain network resumes confirming the current overdraft transactions and then resumes the transactions down the link.

BFMeta's in-chain granting mechanism is to assume trustworthy processing of data that has been confirmed in more than 1 block when the other party is down or gone, and to negatively account for assets that are assumed to be trustworthy overdraft, and to list the outstanding debt. Waiting for the other party to finish rebuilding itself and the outstanding debt is greater than 0 starts waiting for the other party to buy the bill. When the other party has not finished buying the bill, all the new cross-chain transactions added by the other party are queued up after the buy transaction, and when the other party confirms the bill one by one, then starts processing the new transactions. In this way, no matter whether the other party is in normal status or not, it does not affect its own past verification and future transactions, and at the same time, it does not reduce the reliability of cross-chain data. For if there is still a need to continue cross-chain transactions, it is only necessary to clear all the previous outstanding bills list. That is, it can guarantee the reliability of both, and release the interdependence of both, and solve the coupling between cross-chains.

3.5.3 Cross-Chain Asset Interchange

Digital assets are an important part of blockchain, and different blockchains often carry different digital assets. In actual business scenarios, it is often necessary to

exchange assets between different blockchains, and this exchange often becomes a difficult problem; in order to minimize the exchange of assets between chains, the current common solution is to establish a third blockchain to establish exchange relationships with the parties that need to exchange assets respectively, and then only indirectly exchange assets. In order to minimize the asset exchange between chains, the current common solution is to establish a third blockchain to establish an exchange relationship with the parties that need to exchange assets respectively before the asset exchange can be carried out indirectly, which increases the steps of exchange, prolongs the time of transaction, increases the complexity of related business, and prevents the further development of upper layer applications due to the additional third chain, and the probability of the three chains working consistently and stably in the actual operation is much lower than the original two chains, so there are almost no real applications landed so far; later on Some new chains, in order to avoid similar problems to the greatest extent, tend to adopt some kind of best practice standard, and the standard also leads to a group of chains with almost the same overall structure, and there should have been a better way to exchange and circulate the assets on these chains, but it has not appeared yet either; Then how to design an asset exchange between different blockchains without relying on a third chain becomes an urgent problem to be solved.

BFMeta builds a method to circulate assets among multiple blockchains, including the steps of exchanging new assets across chains, circulating the transferred new assets in the local chain, and transferring the transferred new assets back to the original chain again. The transfer of new assets back to the original chain also includes the sub-steps of establishing connection, confirming transactions and validating assets.

The core of BFMeta cross-chain transaction is to establish a micro-asset issuance channel between different blockchains. When there is a need for cross-chain asset

transactions, the counterparty assets will be issued as new assets of this chain and the assets of this chain will be frozen, so that the assets of other chains can circulate in the original chain and in the new chain to achieve real cross-chain asset circulation, and at the same time, the total amount of original assets can be kept unchanged.

1) The core function of the asset approver is to confirm the validity of the assets and manage the total amount of assets. There are two core modules in the asset approver, the asset detection module and the asset accounting module. The asset detection module is used to verify the availability and authenticity of assets based on local configuration; the asset accounting management module is used to manage all cross-chain assets of this chain and the allocation and usage.

2) The core role of transaction manager is to help two chains reach a common asset cross-chain status. There are two core modules in transaction manager, asset credential management module and transaction status synchronization management module. The asset credential management module is used to confirm and keep the source of cross-chain assets; the transaction state membranes management module is used to collaborate with both parties to complete the issuance confirmation of new assets.

3) The destruction of the freezer is used to ensure that the global total of the assets can be maintained after the assets go to other chains. The core function of the destruction freezer is to keep the global total of the chain intact. There are two core modules in the destruction freezer, the asset freezing module and the asset destruction module. The asset freezing module is used to freeze the assets of the chain when new external assets enter the chain, so as to ensure that the total amount of the chain's assets remains in the original state after they go out; the asset destruction module is used to destroy external assets when they are

transferred back, and at the same time release the chain's assets, so as to ensure that the total amount of the chain's assets remains in the original state when external assets are transferred out.

3.6 Three-Layer Blockchain Architecture

With the emergence of Bitcoin, blockchain technology is increasingly recognized. The current blockchain structure is inconvenient to use for business scenarios and business authorization with huge data volume, especially for retrieval and verification, which often takes a lot of time to retrieve, and many of them are doing useless retrieval. The current blockchain architecture is to carry these data in one blockchain, so the speed of verifying and retrieving lottery tickets will be very slow, because it needs to retrieve data of all outlets, all lotteries and all issues in the country to get the information to be verified and retrieved.

BFMeta provides a three-layer blockchain architecture that is fast in retrieving and verifying data, and easy and fast to use to solve the tedious retrieval and verification. It consists of.

(1) building a chain for product-level authorization of issuance, where the issuance chain signs and authorizes each product.

(2) constructing an authorization chain for participant authorization, where the issuance chain authorizes the generation rights of the corresponding products to the participants in the authorization chain; the authorization chain performs signature verification for each participant.

(3) Construct a production chain for recording actual production operation data,

and piggyback the signed participants' production operation in the production chain.

By separating the technical responsibilities of different steps through a three-layer blockchain, the verification and retrieval of correctness are taken out from the massive data, thus significantly improving the performance of blockchain when verifying and retrieving on massive data applications, increasing the speed of retrieval and verification, and making it easy and fast to use.

3.7 Big Block

The blockchain based on the Bioforest chain system is able to forge blocks containing rich information, thanks to the algorithms we use such as memory image type storage, multi-process processing of events, and matrix broadcasting, which enable massive events to be processed in a short time, and the forger will sign each event during the process. After the successful forging, we will broadcast the block header to the blockchain network, and other nodes will enter the synchronization process after receiving and verifying the block header information. Since the synchronization of big blocks will consume more traffic and arithmetic power, we have designed some strategies to ensure the stability of the nodes when they are synchronized, following the principle of first request first synchronization to ensure the stability of the nodes while providing efficient services to the outside world. When some nodes have already synchronized to the block, they will also broadcast to other nodes, so that other nodes can share their resources as well.

3.8 Address Private Key Management Mechanism-My Secret

We have redesigned the management mechanism of address private key, which we call My Secret.

The private key is the bottom line to ensure the rights and asset of users. In most blockchains, each user has a pair of public key and private key, and because the private key string is irregular and also long, resulting in almost no user will directly remember this private key, more often it is saved to an album in the form of a picture QR code, or directly by a third-party wallet service provider for unified storage, which can obtain some direct benefits.

(1) It is convenient for users to transfer and keep it in the form of QR code.

(2) The advantage of unified storage by the third-party wallet service provider is that the user only needs to remember the password set by the service provider. Along with these benefits, there are some hidden dangers:

a. Images can be easily lost.

b. Third-party service providers may face problems such as security loopholes, closing down, and supervisory theft, which essentially exchange the credit of the wallet service provider for the wallet key, defeating the original purpose of blockchain decentralization and de-credit intermediation.

In order to provide convenient key management on the basis of key security, we designed "MySecret", which uses a custom long secret message as a seed (it can be a favorite line, a song, a poem), and then the key will be stored in the key. The key

is encrypted and put into the on-chain key safe. It solves a series of problems such as traditional cipher with few bits and low strength that can be easily cracked, the original key characters are messy and meaningless and hard to remember, and the third-party central custodian (such as blockchain.info) is unreliable, so that users can really use keys safely and conveniently in a decentralized environment without using other third parties.

3.9 Automatic Upgrade

In the era of Web 2.0, it is common for an application to be upgraded and updated due to certain bugs or new features. Like all applications, blockchain also needs to be upgraded to keep up with the times. However, blockchain upgrades are much more difficult than regular software upgrades:

a. Upgrading a traditional blockchain requires forking the network, such as Ethernet switching the consensus mechanism from PoW to PoS, which can only be achieved through a hard fork.

b. At the same time, the upgrade work requires preparations ranging from months to years to complete.

BFMeta revolutionizes this process, enabling blockchains to be able to upgrade themselves without the need for forked chains. These forkless upgrades are achieved through BFMeta's open and transparent on-chain governance in which everyone participates. With this feature, BFMeta enables projects to remain agile, adapting and evolving with technology. It also significantly reduces the risks associated with a controversial hard fork.

3.10 Fork Merging

In the BFMeta, when legitimate blocks with different hashes appear at the same height, the blockchain will create a temporary fork, at which point the network is able to quickly identify and perform a rollback merge based on consensus rules. The confirmed transactions in the rolled-back blocks are also rebroadcast to the network. The block fork consensus rules follow the block participation > accumulated fees in the block > block signature to select the block to be applied. The participation is determined by the transactions confirmed in the block, and is calculated as follows: changes in asset within the block * consensus asset weight + number of transactions confirmed in the block * consensus transaction weight, where the consensus weight is specified in the Genesis block.

3.11 Distributed Computing

With the development of the Internet era, the requirements for information technology are getting higher and higher, and more and more scenarios require the use of computers for computing, while the computing capacity of a single computer is always limited, in practical applications often use the form of clusters for computing, but this often requires the institution or individual building the cluster to have a certain initial economic strength, using economic strength to turn to the scale of the cluster, thus improving the overall computing capacity But not all organizations have a certain scale of economic strength at the beginning of the business, in addition even if a certain scale of cluster computing capacity is built, but does not always need such a large computing capacity, often in some sudden business only need a very high peak computing demand, in most of the time are only low-load computing needs, so in the actual business scenario, build a

large-scale computing capacity Therefore, it is not a cost-effective solution to build large-scale computing capacity to meet the occasional peak computing demand in real business scenarios. Currently there are some solutions on the market to solve the problem, such as cloud computing, the demand side can increase or decrease the cloud server at any time according to the business and performance needs, this way to a certain extent to solve the problem of large initial investment, performance can be flexibly configured, but still only simplifies the problem of building a convenient and flexible computing capacity, and does not solve the problem of flexible allocation of computing resources according to the demand for computing capacity. Then how to provide a truly flexible allocation of computing resources without a one-time large-scale investment and elastic computing resource allocation scheme has become an urgent problem to solve.

BFMeta invents a blockchain-based distributed computing method, including computing task definition, computing task distribution and computing task execution, where computing task definition includes task information entry and processable task type registration, computing task distribution includes data disassembly and node connection, computing task execution includes task information acquisition and task execution. The beneficial effect of this invention is to realize decentralized elastic distributed computing, which solves the problem of large-scale distributed computing and waste of idle resources.

4. Consensus Protocol

4.1 Consensus Algorithm of BFMeta

4.1.1 TPOW+DPOP

Consensus mechanism is the soul of blockchain, which is the necessary means for blockchain network to reach agreement in a decentralized and distributed environment.

Advantages of current consensus mechanisms

(1) Proof-of-work mechanism POW

The node with the strongest arithmetic power generates the block, which can effectively increase the cost of evil; the difficulty enhancement strategy reduces the probability of any multiple blocks on the blockchain being rewritten at the same time by technical means to negligible.

(2) Proof-of-Stake mechanism POS

The competition of playing blocks by the nodes with the largest stake can avoid the waste of computing resources and make the cost of evil-doing directly related to its stake, which reduces the probability of evil to a certain extent by means of business.

(3) Practical Byzantine Fault Tolerance Mechanism PBFT

By all nodes in the network to participate in voting, voting less than (N-1)/3 nodes opposed to reach agreement and generate the block, this mechanism is practical, efficient, less waste of resources and scalable.

As time goes by and the business diversifies deeper, these consensus mechanisms

with obvious advantages start to show symptoms of overwhelm and exhibit obvious drawbacks in specific scenarios.

Problems of various current consensus mechanisms

(1) Wasted computational power

In the proof-of-work mechanism POW, only the nodes with the most computational power can generate the blocks, which leads to a huge waste of computational power and prevents the general public from truly participating in the consensus of the nodes.

(2) Concentration of stake to the top

In the proof-of-stake mechanism POS, the larger the stake is, the higher the probability of being eligible to generate the block, and generation the block means a reward, which leads to the mutual promotion of "getting a reward to increase the probability of generation the block" and "increasing the probability of generation the block to get more stake", resulting in small This leads to the marginalization of the nodes with small stake and the loss of the right to participate in consensus.

(3) Low cost of evil

In Byzantine fault-tolerant mechanism, since all nodes can participate in consensus voting, this will lead to the weakening of the business properties represented by their votes, and a node without stake will have almost no cost of evil in the consensus process.

Consensus mechanism specific to BFMeta

We redesigned the participation-based TPOW (Transaction Proof of Work) + DPOP (Delegated Proof of Participation) + DPOS + PBFT consensus mechanism.

We redesigned the TPOW+DPOP+DPOS+PBFT consensus mechanism based on the degree of participation, which not only inherits the business attributes of POS, the efficient attributes of DPOS, and the full participation attributes of PBFT, for the sake of the sustainable development of BFMeta ecology, but also for the sake of higher reliability of BFMeta data and effective avoidance of the problems arising during the development of the existing consensus mechanism. In addition to effectively inheriting the business properties of POS, the efficient properties of DPOS, and the full participation properties of PBFT, it can also effectively avoid the problem of low cost of collective evil by nodes with no stake and high stake, and also provides a basis for The D-Wallet terminal to participate in consensus.

In this consensus mechanism, participating nodes not only need to provide proof of stake, but also proof of participation, where The R-Node obtains participation by providing highly reliable network performance, and The S-Node obtains participation by providing terminal services, and each participating node's activity on the network increases its participation to a certain extent, and the increase of participation is based on obtaining the service signature of the served This ensures that participants of different dimensions can participate in the consensus and governance of the network, thus effectively avoiding the shortcomings of the single-dimension consensus mechanism.

DPOP Consensus Algorithm

The DPOP consensus mechanism is determined by the amount of stake and participation, and the forger is determined by the number of votes in each round of voting. The forger for the next round will be selected in the end-of-round block based on the number of votes, online rate, and public key order, where the number of votes is the first element.

Example of consensus mechanism code (partial):

```
 Calculate the number of votes (number of stake, number of transactions) for an address {
 const participation = number of transactions * participation ratio * end-of-round constant
 const total number of votes = number of stake * proportion of stake + participation
 return total number of votes;
}
```

The above formula is able to derive the number of votes for the address. With the votes, the address can participate in the on-chain governance and provide votes for the nodes, and the address can also receive governance rewards.

TPOW Consensus Algorithm

TPOW (Transaction Proof of Work) refers to the proof of workload of a transaction. It means that a transaction is on the chain, not with zero computational cost, and needs to be supported by a certain amount of arithmetic power to be submitted to the chain. Usually, when we talk about proof of work, we think of it as a big "power consumer", but in TPOW, this is not the case. Generally, TPOW does not affect the daily on-chain transactions of ordinary users, but through the configuration of parameters, its threshold is often dynamically deployed based on the number of asset in the address and the number of activities in a period of time. The more stake an address has, the higher the trigger threshold for TPOW.

Its presence will only have an impact on hackers who want to launch attacks on the blockchain in a short period of time, solving unlimited low-cost DDOS attacks. Ordinary users more often than not do not really need to provide the

arithmetic example, but only need to wait online to the TPOW algorithm in the time parameter as time passes and slowly back down the difficulty can be. So also indirectly will allow users to have more online time proof to contribute part of the distributed network, making normal use and a large contribution to the ecology of the user to obtain governance priority.

4.1.2 Miner Protocol

4.1.2.1 Authorized Creation Protocol

Authorized Genesis Protocol, refers to the protocol corresponding to the Genesis block, which is signed and generated by the generator of the Genesis block. The authorization protocol is mainly to mark the capabilities that this node can obtain, which covers the maximum number of nodes, the maximum TPS, the range of events allowed to be processed, and the assets allowed to be circulated. The authorization is not verified when the block is synchronized, while it is validated when the block is forged. It also means that this authorization file will be able to customize the capabilities of the nodes. In the public chain, we provide full authorization for all nodes by default.

4.1.2.2 Genesis Basic Protocol

Genesis basic protocol refers to the basic information of this chain, including chain MAGIC, chain name, master asset name, Genesis address, etc.

4.1.2.3 Consensus Incentive Agreement

The Consensus Incentive Agreement, refers to the revenue generated by each block forging as the blockchain advances and the distribution rules for voting and

forging, which we define in the Genesis block. In BFM, the block forging reward is 40 for the first 20 years, and 20 for each subsequent block; the ratio of allocation and forgers to voters in each block is 4:6

4.1.2.4 Block Forging Agreement

The block forging protocol refers to the forging interval and the number of blocks per round. BFMeta has a forging interval of 64 seconds and a round of 57 blocks. The last block of each round is the end-of-round block. In the end-of-round block, we calculate the required checkpoint data to be generated for each round by the algorithm.

4.1.2.5 Contract Execution Protocol

The contract execution protocol means that some of the event parameters in this chain will be determined by the protocol of the creation block. For example, the minimum asset value required to issue assets, register the chain, the maximum number of bonus events that can be issued, and other configurations.

4.1.2.6 Event Processing Protocol

The event processing protocol refers to the protocol about the processing capability of events in this chain, including the maximum validity, maximum TPS, maximum size of a single event, and the minimum handling fee consumed per byte.

4.1.2.7 Proof of Algorithm Protocol

The proof-of-algorithm protocol specifies the rules for the TPOW parameters of

the BFMeta. The higher the difficulty of the TPOW, the more difficult the events processed per unit block. The difficulty of TPOW can be reduced by increasing the address participation, thus processing more events per unit block.

4.1.2.8 Network Communication Protocol

The blockchain consensus port for this chain is defined here. Other ports are configured in the configuration file.

4.2 Block Forger (Miner Node) Rotation

4.2.1 Multi-Node, Multi-Process Block-Generating Method

We have improved the mechanism for competing to generate a block, which we call it - CABP (Competitive Accounting Based on Participation based competitive bookkeeping mechanism). Blocks are the basic units that make up blockchain data, and the block generation strategy will directly affect the performance of the blockchain network and the rights and asset of participating nodes. Traditional blockchain's block generation blocks have some characteristics.

a. Participating nodes with complete data can only participate in block generation.

b. The participating node with the most arithmetic power or the highest asset can only participate in the block generation.

c. In most consensus mechanisms, only the block generation can gain.

These features pose some immediate problems.

a. Solving the data reliability problem in terms of data integrity also constrains the

participation of lightweight nodes.

b. The use of arithmetic power and asset alone as the basis for generation blocks will lead to long-term development hindrance, and there will be significant resource concentration and stratification.

c. The fact that the only way to gain revenue is to participate in blocking will discourage nodes from contributing to participation in other ways.

By summarizing the problems caused by the traditional blocking method, we have improved the mechanism of blocking based on a longer-term consideration, in which arithmetic power and asset are no longer the only criteria for judging, and we have introduced more dimensions, such as stability, activity and transaction volume, so that all types of participants can participate in the blocking process, which will facilitate BFMeta to attract all kinds of players and thus enrich the participant ecology. We have improved the mechanism of playing block rewards, we call it - BPIM (Based on Participation Incentive Mechanism).

Rewards are necessary to sustain the blockchain network. The science of reward mechanism design will promote the prosperity of the blockchain network and in turn will constrain the development of the blockchain network. There are roughly two ways of rewarding in traditional blockchain networks:

a. Receiving block rewards by competing to generate blocks.

b. Receiving transaction fees by participating in transactions.

These two ways are simple in logic and easy to implement, but have some problems in later development:

a. Participating nodes with low competitive ability cannot get block rewards, which may be unfair to participating nodes that have contributed other participation.

b. Transactions by fees will bring about fee discrimination, and the generation block nodes may prioritize transactions with high fees in order to ensure their own revenue.

In order to solve the problems caused by the traditional reward mechanism, BFMeta introduces a multi-dimensional reward mechanism, we design the R-Node (real-time node), the S-Node (service node) and the D-Wallet (distributed wallet). The R-Node and the S-Node can switch between each other or work simultaneously according to the network environment or participants' wishes, so that contributors of different dimensions can be rewarded. The R-Node and the S-Node can switch between each other or work simultaneously depending on the network environment or participants' wishes, allowing contributors of different dimensions of participation to be rewarded.

In the reward distribution, both the rewards earned through asset and the participation provided will be rewarded and distributed when the block is played (the block will also be distributed when it contains a fee). Each service node will increase the weight of the service node to obtain the reward, so as to encourage the access of service nodes when there are few service nodes, and encourage real-time nodes to provide more efficient service nodes when there are enough service nodes, thus dynamically balancing the BFMeta network from multiple dimensions through a multi-dimensional reward mechanism.

4.2.2 Block Forger Election Algorithm

Block forgers, those nodes in the BFMeta network that are responsible for

collecting event information and packing it into blocks (i.e. miners). In addition to having the characteristics of a normal node, a block forger is responsible for:

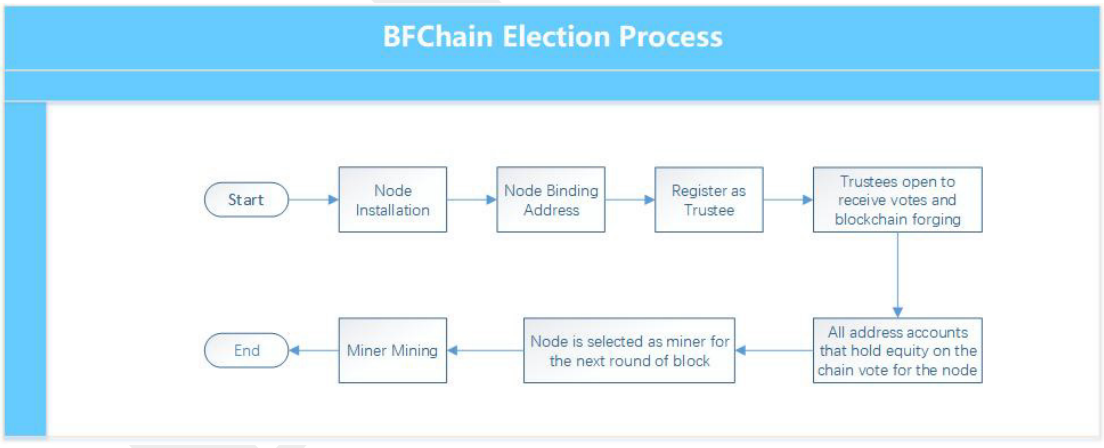
a. Collecting events in the network.

b. Validating events and packing them into blocks.

c. Broadcast the block to other nodes and add the block to their own local block chain after passing the validation.

In the BFMeta network, there are 57 block forgers in each round, and the BFMeta block forgers are selected based on a combination of parameters such as the number of votes received by the trustees and their online rate.

Flow Chart of Campaign



The process involves three main parts in the generation of block forgers:

a. Nodes participate in the campaign

a) Becoming a trustee

b) Opening to receive votes and block forging

b. Address accounts holding asset vote on trustees

c. Becoming a forger

a) Enter the candidate zone

b) Become a Forger if the campaign is successful

4.2.2.1 Becoming a Trustee

In order to improve consensus efficiency, in the BFMeta blockchain ecosystem, address accounts can apply to become trustees as long as they hold asset in the main chain. All address accounts can vote for the trustees who have opened to receive votes, and the system will elect 57 trustees from them according to certain algorithmic rules. These 57 trustees, or block forgers, will be responsible for forging the next round of 57 blocks.

In the BFMeta ecology, there are two main types of trustees as follows:

a. Ordinary trustees: Ordinary address accounts become trustees by application.

b. Genesis Trustees: Address trustees bound to the Genesis block, currently there are 114 Genesis Trustees.

On-chain rules

a. Registering as a trustee is an event that requires payment of a certain on-chain fee, so only accounts holding asset in the main chain can apply to become trustees.

b. By default, a maximum of 10 "registered trustee" events are confirmed in each round; if there are more than 10 "registered trustee" events in that round, the

events after the 10 confirmed events will be queued until the next round block forging begins before they can be confirmed.

c. Only when a node becomes a trustee can it enter the candidate zone, so that it has the chance to be selected and become a forger.

d. Support a node to bind multiple trustees at the same time.

4.2.2.2 Receiving/rejecting Votes

After the address account successfully applies to become a trustee, it also needs to be configured whether to participate in the forger campaign (i.e., receive votes), and the default is to reject votes. Participating in the forger campaign means that other nodes on the chain will potentially vote for this trustee, then, it is possible that it will be selected to become a block forger.

It should be noted that if a trustee is lucky enough to be selected as a block forger and is unable to forge a block because the node is not ready when the block is forged (e.g., the block height is not finished synchronizing, the node network is unstable, the node is in the process of upgrading and updating, etc.), it will affect the trustee's next campaign and greatly reduce the probability of being selected subsequently. In order to reduce the dropout rate of trustees, it is recommended to first refuse to participate in the forger campaign (i.e., refuse to vote) when the node is not ready, until the node is ready, and then turn on receiving votes.

4.2.2.3 Address Account Voting on Trustees

Voting means that all address accounts on the chain holding asset in the main chain can vote on trustees in various supported DApp (e.g. BFMeta App) or node programs. The more node votes a trustee gets, the higher the probability that it will become a forger.

4.2.2.4 Automatic Node Voting

The voting-related rules are detailed in the chapter on consensus voting mechanism, and will not be repeated here.

4.2.2.5 Entering the Candidate zone

The nodes on the chain vote for the trustees involved in the election by participating in voting (automatic and manual voting) with the votes obtained in their hands. The bottom layer will count all the trustees that were voted in this round at the end of the block of this round. Those Trustees who have been voted for and recommended will enter the Candidate Zone, and the system will select the 57 Block Forgers for the next round from the Candidate Zone according to the election rules of the bottom tier (see "Becoming a Forger" section below).

How to improve the probability of entering the candidate zone?

a. Increase the online rate
b. Increase the number of blocks forged
c. Increase the number of packaged deals
d. Increase the percentage of votes received
e. Re-register as a new trustee

4.2.2.6 Becoming a Forger

Campaign Consensus

a. Only trustees (hereafter called candidates) who enter the candidate zone may be selected to become forgers for the next round of blocks.

b. The system will select 57 from the candidates according to the established consensus of the bottom layer as the block forgers for the next round.

c. The underlying rules, which depend mainly on the following two parameters:

a) Number of votes received: i.e., the number of votes received by the candidates in the current round of voting.

b) Online rate: online rate = number of forged blocks / (number of forged blocks + number of drops); once a node has had one drop, the online rate will not return to 100%, but as the number of forged blocks keeps increasing, its online rate can be infinitely close to 100%.

d. Judgment method

a) Priority comparison of votes: the number of votes received is compared first: the higher the number of votes received, the more likely the candidate will be selected.

b) If two candidates have the same number of votes, then compare the online rate: the higher the online rate, the more likely to be selected.

For example:

Candidate	Number of forged blocks	Number of dropped lines	Online rate	Number of votes received
A	70	30	70%	100
B	90	10	90%	99
C	85	15	85%	100
D	100	0	100%	50

In the above table, the probability of each candidate being selected in descending order is: C > A > B > D.

How to increase the online rate

a. Reduce the number of dropouts: Try to avoid being selected to forge a block when the node is in the following states: not finished synchronizing, the node is in the upgrade period, the node itself is not ready. If a node is found to be in one of the above states, the trustee can turn off receiving votes by triggering a "reject vote" event. After the "reject vote" event takes effect, the node bound trustee will no longer be voted, and its probability of being selected will be greatly reduced.

b. Increase the number of forged blocks: Nodes should participate in block forging as much as possible to get more forged blocks. Once the number of blocks forged by that node is much larger than its dropouts, then its online rate will remain high, even infinitely close to 100%.

Initial Forger

The BFMeta initial forgers (i.e., the block forgers in the first round) are elected from the 114 Genesis trustees according to the underlying campaign consensus (as above). Each subsequent round of block forgers will be elected through the campaign process described above.

4.2.3 Consensus Motivation

4.2.3.1 Consensus Incentive Mechanism

The consensus incentive mechanism is an incentive for miners who provide arithmetic, network, storage, verification and packaging services to the BFMeta network, as well as for all address accounts that provide arithmetic, network and storage to participate in the BFMeta voting governance.

Currently, the incentive for successfully forging a block in the BFMeta consensus incentive mechanism can be divided into two components.

a. Granting a certain amount of asset in the main chain

b. The uplink fee generated by all events in the currently forged block

Those who can receive this incentive include:

a. The forger of the block, who can receive an incentive of 40% of the total incentive of the block.

b. All address accounts that voted for that forger in this round, which can receive an incentive of 60% of the total incentive for that block.

4.2.3.2 Total Incentive Received for Forged Blocks

In the BFMeta network, after each successful forging of a block, the bottom layer will calculate the total incentive it should receive, as follows

totalReward = rewardPerBlock + sum(eachtransactionfeeOftheBlock)

where:

a. rewardForPerBlock: that is, the main chain asset incentive granted by forging this block. In the BFMeta consensus, the number of asset for this incentive will vary with the block height interval, as follows:

Blockh eight	Number of asseti ncentivized (BFM)
1- 9855000	40
9855000 -	20

b. sum(eachtransactionfeeFortheBlock): where eachtransactionfeeFortheBlock is the upload fee of each event packed into this block; sum(eachtransactionfee FortheBlock) is the sum of the upload fee of each event packed into this block. eachtransactionfeeFortheBlock.

The total incentive of the forged block will be shared between the block forger and all address accounts that voted for the forger in this round. The consensus rules for sharing are:

a. The block forger will share 40% of the total incentive to forge the block, i.e. totalReward * 0.4

b. All address accounts that voted for the forger in this round will share a total of 60% of the total incentive to forge the block, i.e. totalReward * 0.6.

c. The incentive to be shared by each address account that voted for the forger is: totalReward * 0.6 (the number of votes the address account voted for the forger in the previous round / the total number of votes the forger received in the previous round)

4.2.4 Voting Algorithm

The BFMeta block forgers are selected based on the number of votes received by the trustee and the online rate and other parameters according to the underlying consensus rules (see Block Forger Campaign for details).

The number of votes received by the trustees is obtained by voting for the trustees who have turned on receiving votes through the accounts of addresses holding asset in the main chain across the network.

All address accounts on the chain that hold main chain asset can vote for trustees in various supported DApps (e.g. BFMeta APP) or BCF node programs. The more votes a trustee receives, the higher the probability that he or she will become a forger.

This chapter will introduce the voting rules of BFMeta in five parts:
- Getting votes
- Voting automatically
- Manual voting
- Voting incentive allocation
- End-of-round calculation

4.2.4.1 Getting Ballots

A ballot is a ticket used by an address account on the chain to vote for a trustee. An address account can only vote for a trustee if it obtains a ballot.

In order for an address account on the chain to obtain a ballot, it needs to ensure that

a. the address account holds an asset in the main chain

b. Voting governance is turned on and has been online for 6 consecutive hours (i.e., online for three consecutive rounds)

The consensus formula for the number of votes (for the next round of voting) that an address account can obtain in each round is:

The number of votes an address account can receive in each round = balance

balanceWeight + numberOfTransaction numberOfTransactionWeight * Rate

Where:

a. balance: the main chain asset owned by the address account as of the previous round.

b. balanceWeight: the weight of the main chain asset, constant, the Genesis consensus default is: 5000.

c. numberOfTransaction: the number of events that have been confirmed in the last round for the address account.

d. numberOfTransactionWeight: event volume weight, constant, Genesis consensus default: 1.

e. Rate: max(mainchain asset of each address account participating in the last round)/max(number of confirmed events for each address account participating in the last round).

BFMeta will count the number of votes each account has in the next round at the end of each round block.

4.2.4.2 Automatic Voting

Auto-voting means that BFMeta provides the recommendation algorithm in the bottom layer and nodes configure the algorithm parameter values independently. When a node turns on auto-voting, the bottom layer will automatically vote for trustees that meet the conditions based on the algorithm and the configured

parameter values.

4.2.4.3 Automatic Voting Recommendation Algorithm

The configuration parameters involved in the underlying recommendation algorithm are listed in detail in BFMetaPC.Config.AutoVoteModel and will not be repeated here. The following examples illustrate the auto-vote recommendation algorithm.

Parameter	Definition	Set value
enable	Whether to open automatic voting for nodes	true
priorRecommendedNumber	Whether to prioritize the number of referrals	true
maxNumberofRecommended	The maximum number of candidates elected by this node	57
productivityPercent	Percentage of online rate	30%
forgedBlocksPercent	Percentage of the number of forged blocks	30%
applyTxPercent	Percentage of the number of packaged transactions	20%
votePercent	Percentage of votes received in the previous round	10%
newDelegatePercent	Percentage of new trustees	10%

The above table is set up to indicate that this node needs to elect a maximum of 57 candidate trustees through automatic voting, and the composition of these 57 candidate trustees is:

- The 17(57*0.3) trustees with the highest online rate within the selected block range that are open to receive votes.
- The 17 (57*0.3) trustees with the highest number of blocks forged within the selected block range that are open to receive votes
- 11 (57*0.2) trustees with the highest number of packaged deals in the selected block range who opened to receive votes
- 6 (57*0.1) trustees who received the highest number of votes in the previous round
- 6 newly registered trustees in the selected block with the following rules: Number of votes > Number of primary asset > Public key.

4.2.4.4 Manual Voting

Manual voting means that a node can vote for any trustee on the chain that has been enabled to receive votes, and the number of votes cast is controlled by the node itself, as long as it does not exceed the maximum number of votes currently owned by the node.

4.2.5 Voting Incentive

When a forger has forged a block, the bottom layer will directly calculate the voting incentive due to all the address accounts that voted for the forger in the previous round, and the incentive rights will be directly issued to each voting address account.

4.2.5.1 Incentive Formula

TotalRewardOfVote : {rewardPerBlock + sum(eachtransactionfeeOftheBlock)} * votePercent

a. rewardPerBlock: i.e. block incentive, determined by the underlying consensus, the incentive given for forging the block will be different for different block heights of forging. Current height, the underlying consensus default is 15BFM.

b. sum(eachtransactionfeeOftheBlock): each event generated on the chain needs to pay a certain uplink fee, this uplink fee can be set by the account when initiating the event, or the best uplink fee recommended by the underlying consensus algorithm can be used. where eachtransactionfeeFortheBlock is the uplink fee for each event packed into the block; sum(eachtransactionfeeOftheBlock) is the sum of the uplink fees for each event packed into the block.

c. votePercent: the voting incentive weight, this is a constant, determined by the underlying consensus, the current default is 0.6.

votePercent for each address account that voted for the forger in the previous round = totalRewardOfVote * (number of votes for the forger in the previous round / total number of votes for the forger in the previous round)

For the consensus incentive mechanism of BFMeta ecology, please refer to the chapter on consensus incentive mechanism, which will not be described in detail in this subsection.

4.2.5.2 End-of-Round Calculation

The following data will be calculated at the end-of-round block of each round:

a. Count the number of votes accumulated in the round for the trustee accounts that have opened to receive votes.

b. Count and save the number of votes to be received in the next round for the address accounts with voting governance turned on.

c. Update the account master asset and the number of confirmed events at the end of the round for each address account with voting governance turned on.

d. Update the Rate value (Rate = max(master chain asset for each address account that participated in the previous round)/max(number of confirmed events for each address account that participated in the previous round)).

4.2.6 Distributed Transaction Synchronization

Blockchain is an indispensable and important infrastructure for the digital world in the future metaverse, and on top of this infrastructure, it will carry vertical applications of hundreds of lines and thousands of industries, and many applications will bring massive users, and massive users need the support of blockchain to provide massive transaction processing capacity. The special chain-like block structure of blockchain determines that only one block can become a valid block at the same time, and the block contains the transactions within this unit time, which also constrains that only one batch of transactions can be processed at the same time, which seriously constrains the improvement of blockchain performance. Currently the industry has adopted some methods to

solve this problem, such as removing the block structure, but removing the block structure will lead to a significant decrease in the reliability of transactions, which is a huge cost cost. How to break through the performance constraints caused by the chained block structure without reducing the reliability of transactions has become an urgent problem to be solved.

BFMeta pioneered a method for synchronous processing of distributed transactions in blockchain, which makes it possible to solve this problem: obtain the list of nodes that have completed consensus and count the protocol version with the maximum convention, calculate the transaction range of participating second consensus nodes and send it, check the consensus status, the applicable range of transactions, and Byzantine consistency issues, and when the nodes receive the transactions check the transaction range and out block time, place the result of transaction processing into a new block, wait until the node's block time to announce the result to the public, and complete the parallel processing of the transaction. We also provide a distributed transaction synchronization processing system for blockchain, including a second consensus manager, transaction manager, transaction manager, block forger, etc. The components are connected sequentially, which solves the problem of blockchain processing transactions in parallel by multiple nodes (non-cooperative block-beating nodes) at the same time, thus improving performance.

5. Programmable Contracts

5.1 Smart Contracts

BFMeta builds a method to create smart contracts directly from mobile.

Both parties to a transaction freeze part of their assets to a smart contract in the form of a smart contract, which is submitted to the blockchain for network-wide deposition, and the contract takes effect when a network-wide consensus is reached. After the contract takes effect, both parties only need to sign a supplemental distribution agreement to the frozen agreement for transfers within this asset amount. Since the total amount has not changed at this time, the supplemental agreement only needs to be signed and acknowledged by both parties and no longer needs to wait for network-wide confirmation. This method increases the transfer speed between two accounts that transfer frequently, except for the first freeze and the last unfreeze which are normal speed, the transfer speed will be instantaneous at all other times, and the fee is very low.

Smart contract application on mobile blockchain has both high efficiency, security, simplicity and economic practicality, which is more in line with the requirements of future scale landing application.

BFMeta's smart contracts are the next generation of blockchain smart contracts, unlike the current ones that rely on the execution of virtual machines in nodes, and thus derive various counter-intuitive restrictions to barely maintain the viability of the solution; and run one code repeatedly on each node, which is a backward design for today's energy-constrained reality. distributed network that

relies on on-chain communication for distributed execution of smart contracts, ultimately storing on the chain only the results that are confirmed by multiple signatures and the correct ledger.

Essentially, BFMeta's smart contracts do not define a traditional contract virtual machine, but rather a multi-signature list of variables: it is also the result of the execution of the contract that is agreed to by all addresses involved in the contract, and the total book changes without wrong accounting. This means that the blockchain does not need to execute the contract repeatedly when it is synchronized, but only needs to make sure that all involved addresses agree to the contract result. This does not even require relying on code to execute the contract; it is also feasible for real-life population to agree on the contract outcome and upload the result to the chain. This opens up more possibilities for blockchain ledger operations.

It also means that any programming language can be used for contract development, and it is only necessary to link the addresses involved in the contract based on a distributed network on the chain, execute it according to its content and submit its result with a signature. The fundamental reason why the current blockchain cannot adopt such a design is that BFMeta deeply integrates the distributed network into the blockchain, relying on the highly available distributed network to get an off-chain that is on-chain effect. So we can distribute smart contracts to each node (including mobile nodes) to execute their own part of the code separately, and finally aggregate.

In theory, this model of smart contract can be performed in any programming language, but it involves the idempotence of the contract, in order to allow any node is able to verify its code execution process. Therefore, our recommended programming language of choice is Rust/Typescript. Rust is used because it can

compile WASM small enough and high enough to ensure that all nodes can verify the contract results idempotently. It can also be compiled to Native for some high performance scenarios in environments where the conditions are met. This means that any function and function available in the developer's programming language can be used, and for the executor, it is just a matter of choosing a contract that meets their needs while picking the one with better performance.

Let's take a specific scenario to describe how this contract solution differs from traditional contracts: Suppose there is an "image data collection contract" that requires the collection of photos from a defined number of geographic regions, and the finalist submission address gets the reward. Then in the traditional blockchain, relying only on the contract code, it is impossible to judge whether the photos meet the geographical requirements and quality requirements on the chain; while in BFMeta, after relying on the distributed network to collect the image data, the decision maker uses his own key tools to review the image data locally and manually selects the shortlisted photos, and finally the contract is finalized and uploaded to the chain. If the contributors publish their own review criteria, then they can also let other addresses push each other to review, and the contributors only need to pick the one they want in the end (similar to separating the cake cutter from the cake picker), thus further ensuring fairness.

5.2 Digital Products (DP/NFT)

DP (Digital Products) is an acronym for a non-homogeneous digital asset type, which is a signed proof of a digital product stored on the blockchain that is unique, tamper-evident, and non-detachable.

BFMeta provides a way to mark the ownership of digital products. Each circulation

and change of ownership of a DP work is recorded on the blockchain and a unique digital certificate is generated, making it non-detachable, non-replicable and non-tamperable.

DP is of great significance for building metaverse due to its uniqueness, non-splittability, non-tamperability and replication. It can be used to record and trade digital assets, such as digital works, artworks, property certificates, tickets and game props. At the same time, DP changes the traditional virtual goods trading model, users can directly produce and trade virtual goods as if they were in the real world.

BFMeta connects various assets in the real world with the digital world through DP, continuously enriching the ecological variety of the metaverse, and thus continuously expanding the imaginary boundaries of the metaverse.

5.3 DeFi Support

DeFi (Decentralized Finance) , a decentralized financial system created based on blockchain technology and cryptocurrency, is decentralized, open and transparent, reliable, fair and secure. It has been able to make possible the attribution, circulation, realization of value and authentication of virtual identity in the metaverse.

BFMeta supports both various digital asset (Token) DeFi and digital product DeFi. DPFi is a liquidity protocol for BFMeta's digital product DP, which allows DP owners to obtain secured asset loans from peer-to-peer liquidity providers in a fully de-trusted manner, increasing the liquidity of the DP assets they own. DP liquidity providers use DPFi to earn attractive returns or have the opportunity to

acquire DPs at a price below their market value in the event of a loan default.

6. Programmable Digital Asset Issuance

6.1 Destruction Issuance (Deflation Mechanism)

BFMeta builds a method, system and apparatus for asset issuance based on blockchain token destruction.

6.2 Decentralized Asset Exchange

Assets are an important part of production and life in social activities, which are both necessary elements for production and important driving factors for social development, while their circulation efficiency in society largely affects the forward rate of social development.

How to accelerate the circulation efficiency of assets to speed up the development of society is a unanimous effort of the whole social activities of the relevant institutions and individuals.

In the traditional way, the circulation of assets often relies on a central authority, such as housing agents, notaries, property management centers, intellectual property exchanges, etc. They act as intermediaries to provide services for both sides of the circulation of assets, either by the intermediaries to intervene in holding assets and funds to guarantee the transaction, or by the intermediaries to witness the transaction, which to a certain extent alleviates the problem of asset circulation, but does not well The lack of any kind of intermediary will largely lead to the failure of asset circulation, and the simultaneous participation of intermediaries determines the inefficiency of completing asset circulation;

however, if we let the two sides deal directly, there is a high probability of not reaching agreement because of the trust problem. Then, how to establish a method of asset circulation that is trustworthy for both parties but not affected by the efficiency of the third party becomes an urgent problem to be solved.

BFMeta builds a decentralized asset exchange method, where the issuing party fills in the information of the issuing entity and the issuing asset, the issuing party uses a digital certificate to sign the transaction, submits the blockchain transaction to the blockchain and places the transaction into the block; the participating party uses a private key to encrypt the identity feature information and turns it into a blockchain transaction to complete the real name authentication; when the blockchain extracts the information of the asset After the blockchain extracts the information of the asset, the sender fills in the asset transfer information and uses the signature of the sender to sign and send the blockchain transaction; the counterparty receives the blockchain transaction, signs it and sends it to the blockchain for processing, and completes the asset exchange by verifying the correctness of the transaction, which realizes the role of decentralized and rapid flow of assets and solves the credit risk introduced by the intermediary and the problem of inefficient asset circulation.

7. Chain Services

7.1 Chain Domain Name-LNS

In web 2.0, due to the disadvantages of a string of numeric IP addresses that are not easy to remember and do not show the name and nature of the address organization, domain names were designed and the DNS (Domain Name System) was used to map domain names and IP addresses to each other, making it easier for people to access the Internet without having to remember the number of IP addresses that can be read directly by machines. IP address number strings.

In the blockchain system, node IPs are also not easy to remember. Can we design a corresponding chain domain name for each node on the blockchain, just like the domain name system?

Meanwhile, blockchain can solve the security and privacy problems of each website on web 2.0 Internet, so can we develop a new chain domain name system on blockchain to realize decentralized and distributed access to websites on the chain?

BFMeta has built a new distributed location name service LNS, which is called "Location Name Service". Organizations/users can register or purchase LNS and create a corresponding DWeb site for them, which can be easily accessed by people. The emergence of LNS location name service builds a bridge between the complex computer language of blockchain and the common human language, so that people can easily access the blockchain website by entering the name of a person/organization + .com/cn/org, just like the Internet in the past.

7.2 DWeb

BFMeta provides DWeb blockchain website construction for organizations and individuals using a peer-to-peer protocol invented in-house. These DWeb sites can store web pages, images, media, user data, etc. just like regular Web 2.0 sites.

In the Web 2.0 era, hosting a website was traditionally done by "servers", which could be centralized vendor computers or cloud dedicated computers. to help DWeb sites online, and can even be permanently open for favorite sites, permanently online.

Users can grab an LNS chain domain name at BFMeta and create a corresponding DWeb site, and then share the DWeb link with any other user.

7.3 Dual Offline Payment

The Internet has become an essential component of our modern daily life. What happens if, in some special circumstances, the Internet is not available? We are likely to be unable to order takeout, call a taxi, watch classroom videos, submit work, communicate with people, transact with people, and so on. The inability to transact with people electronically almost prevents more than 80% of our daily activities. So is there an electronic payment method that can be used even when the network is offline? At present, there are products such as the card-carrying coin purse launched by UnionPay Card and the offline payment launched by Alipay WeChat, but these products still require merchants to be online in order to use them, and it is still impossible for both parties to make payments completely off the network. In the current era of advanced Internet, network offline is still a frequent occurrence, such as on the plane, in the ocean region, in the deep forest, power outage and disconnection, network blockage, fiber break, network

congestion, etc., which can lead to network unavailability, so how to provide payment service for both parties of the transaction even when the network is completely offline has become an urgent problem to solve.

BFMeta builds a blockchain-based offline transaction method, including a unilateral offline transaction method and a bilateral offline transaction method. In the unilateral offline state, the online party is allowed to submit a payment request for the offline party; in the bilateral offline state, the payer issues an irrevocable, non-repudiation and non-forgeable payment voucher for the payee, which is directly used by the payee as the basis of arrival, and the payee cashes the voucher from the network when the network is restored.

BFMeta's offline transaction system, including transaction manager, account synchronizer, and voucher manager; it can provide good support for mobile terminal devices and provide application layer services even if the network is unstable, realizing the role of making payments offline and solving the problem of not being able to make payments in case of network disconnection.

7.4 On-chain Red Envelope

Based on the natural attribute of "daily application" in mobile, BFMeta asset bonus is an important innovation of blockchain application on the ground, which is the starting point to help users establish blockchain awareness. digital wallet or the change of book numbers on Alipay/WeChat red envelopes is fundamentally different. At the same time, in the scenario of physical assets on the chain, on-chain red envelopes can also send the digital assets corresponding to the physical assets.

7.5 Service Market

In the service market, BFMeta provides various valuable surfing market (Web Application) and node application (Node Application) to provide comprehensive and diversified services for developers, different types of nodes and users.

Users can visit interested on-chain Web sites on the Surf Market, or select a favorite node (e.g. EOW) in the node search result list, and then download and install it to use. Blockchain application coders can also develop various DApps, DWeb and deploy smart contracts in the "Developer Community". In the physical chain scenario, digital goods issuers can also display their products, brands and after-sales services in the surfing market.

7.6 Crossing the World

On BFMeta, enterprises can customize and develop parallel chains according to their business needs and application scenarios. For the business that does not need to be completely on the chain, enterprises can also selectively on the chain, independently develop DAPP or embedded development on the main chain of BFMeta. Each sub-chain on BFMeta will interoperate with other parallel chains and form a living biomass together. This group will continue to evolve and iterate and update with the participation of developers and users all over the world to solve social problems.

For enterprises, BFMeta presents three main functions:

1) Empowering evolutionary chain development

Through the authorization of BFMeta, enterprises can customize the development of license chains and parallel public chains according to their business needs and application scenarios.

2) Physical assets on the chain and digital assets (including digital goods, digital consumption points, etc.) issuance

BFMeta provides "digital assets anchored to physical objects, and the digital twin on the chain replaces the physical objects in the chain circulation", as well as digital assets issuance and management services for physical enterprises and institutions.

3) On-chain application development and smart contract deployment

Developers can develop various DApps, DWebs and deploy smart contracts on BFMeta to build a credible foundation platform together.

8. Interface Documentation

8.1 Interface Incoming Parameters and Return Parameters Description

(1) The full name of the interface is the function name of the interface, which is also the full name when called from the command line.

(2) Interface abbreviation is the abbreviated name when the command line call.

(3) Callable method refers to the ways in which the interface is allowed to be called.

(4) The call method is used when http is used, and the string is added in front of /api when websocket is used.

(5) The request and return parameters are described in the syntax of typescript, or in <type definition> if designed for type definition.

8.1.1 Example of Passing/Entering Parameters

The following is a description of the "Get Specified Account" interface for passing and entering parameters.

- Full name of the interface: getAccountInfoAndAssets
- Interface abbreviation: ga
- Callable methods: Http, Websocket, command line, Grpc

- Call method: post
- Interface url address: /api/basic/getAccountInfoAndAssets
- Request parameters

```
interface GetAccountInfoAndAssets { /**account address */

address: string;}
```

- Return parameters

```
interface GetAccountInfoAndAssets extends RespCommonParam {

result: MemInfoModel.AccountInfoAndAsset;}
```

8.2 Basic Interface

This section will briefly introduce the BFMeta basic interface parameters, for more details on the use of each parameter and the re-referencing content, please go to the BFMeta developer community.

8.2.1 Getting BFChain Version Number

- Full name of the interface: getBfchainVersion
- Interface abbreviation: v
- Callable methods: Http, Websocket, command line
- Call method: get
- Interface url address: /api/basic/getBfchainVersion
- Request parameters: None

8.2.2 Getting the Current Latest Block of the Local Node

- Full name of the interface: getLastBlock
- Interface abbreviation: glb
- Callable methods: Http,Websocket,Command Line,Grpc
- Call method: get
- Interface url address: /api/basic/getLastBlock
- Request parameters: None

8.2.3 Getting the Specified Block

- Full name of the interface: getBlock
- Interface abbreviation: gb
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/basic/getBlock
- Request parameters.

```
interface GetBlock {/**block signature* /

signature?: string;/**blockh eight */

height?: number;/**view thep age( 20 recordsp er page)* /

page?: number;}
```

8.2.4 Getting the Specified Event

- Full name of the interface: getTransactions
- Interface abbreviation: gt

- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/basic/getTransactions
- Request parameters.

```
interface GetTransactions {/**event id */

signature?: string;/**event block height */

height?: number;/**The minimum height of the block the event belongs to, can be used with maxHeight to query events in a block. */

minHeight?: number;/**the highest height of the block the event belongs to, can be used with minHeight to query the events of a period */

maxHeight?: number;/**event initiator */

senderId?: string;/**event recipient */

recipientId?: string;/**Event type, if not passed in then event type is not filtered, please refer to */type?: string[];/**The index value of the event, you can query the event based on the index value of the event. The index value may be a value such as assetType or signature or username, and it is recommended to use it in parallel with other conditions to find precisely. */

storageValue?: string;/**View the page (20 records per page)    */

page?: number;}
```

8.2.5 Getting the Last Transaction of an Account

- Full name of the interface: getAccountLastTransaction
- Interface abbreviation: None
- Callable mode: Http,Websocket

- Call method: post
- Interface url address: /api/basic/getAccountLastTransaction
- Description: This interface is used to get the approximate balance of the specified address. According to the return parameter transactionAssetChanges of the accountType to get the assetBalance as the balance
- Request parameters.

```
interface GetAccountInfoAndAssets {/**account address* /

address: string;/**assett ype* /

assetType: string;}
```

8.2.6 Creating an Account

- Full name of the interface: createAccount
- Interface abbreviation: ca
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- interface url address: /api/basic/createAccount
- request parameters.

```
interface CreateAccount {/**account key* /

secret: string;}
```

8.2.7 Getting Node Status

- Full name of the interface: getBlockChainStatus

- Interface abbreviation: gbc
- Callable methods: Http, Websocket, command line, Grpc
- Call method: get
- Interface url address: /api/basic/getBlockChainStatus
- Request parameters: None

8.2.8 Getting the Last Transaction of the Account According to the Transaction Type

- Full name of the interface: getAccountLastTypeTransaction
- Interface abbreviation: galtt
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/basic/getAccountLastTypeTransaction
- Request parameters.

```
interface GetAccountLastTypeTransaction{/**account address* /

address: string;/**transactiont ype* /

transactionType: string;}
```

8.2.9 Getting the Event Type

- Full name of the interface: getTransactionType
- Interface abbreviation: none
- Callable mode: Http,Websocket
- Call method: post
- Interface url address: /api/basic/getTransactionType

- Request parameters.

```
interface GetTransactionType{/**event base type */

baseType: {

    //asset transfer

    TRANSFER_ASSET = "AST-01",

    //secondary password

    SIGNATURE = "BSE-01",

    //register forger

    DELEGATE = "BSE-02",

    //govern voting

    VOTE = "BSE-03",

    //set username

    USERNAME = "BSE-04",

    //start receiving votes

    ACCEPT_VOTE = "BSE-05",

    //stop receiving tickets

    REJECT_VOTE = "BSE-06",

    //create DAPPID

    DAPP = "WOD-00",

    //DAPPID payment
```

```
    DAPP_PURCHASING = "WOD-01",

    //data storage certificate

    MARK = "EXT-00",

  //create asset

    ISSUE_ASSET = "AST-00",

    //destroy asset

    DESTORY_ASSET = "AST-02",

    //initiate a gift of asset

    GIFT_ASSET = "AST-03",

    //accepting a gift of asset

    GRAB_ASSET = "AST-04"};}
```

8.3 Event Class Interface Usage Description

8.3.1 Transfer Events

8.3.1.1 Creating a Transfer Event

- Full name of the interface: trTransferAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trTransferAsset
- Request parameters.

```
interface TrTransferAsset extends TrCommonParam {/**Numbero ft ransferred assets,
0-9a nd without decimal point,m ustb eg reater than 0* /
```

```
amount: string;/**The type of assett ob et ransferred,i n uppercasel etters,3 -5 characters* /

assetType?: string;/**The name of thec hain to whicht he asseti st ransferred,i n lowercasel etters,3 -8 characters* /

sourceChainName?: string;/**Network identifiero ft he chaint ow hich thea sseti s transferred, in uppercasel etters or numbers,5 characters, last biti sa checkd igit */

sourceChainMagic?: string;/**The addresso ft he receiving account of thee vent,b ase58 encodedh exadecimals tring */

recipientId: string;}
```

8.3.1.2 Creating a Transfer Event (with Security Key)

- Full name of the interface: trTransferAssetWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- interface url address: /api/transaction/trTransferAssetWithSign
- Request parameters.

```
interface TrTransferAssetWithSign {/**bufferg enerated by event body without signature, generatedb yT rTransferAsset* /

buffer: string;/**event signature* /

signature: string;}
```

8.3.1.3 Sending a Transfer Event

- Full name of the interface: transferAsset

- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/transferAsset
- Request parameters.

```
interface SendTrCommonParam {/**Buffert ob es igned, converted to base64 string
*/

buffer: Buffer;/**signature of thet ransaction* /

signature: string;/**the security signatureo ft he transaction* /

signSignature?: string;
```

8.3.2 Setting Up a Secure Password Event

8.3.2.1 Creating a Set-Security-Password Event

- Full name of the interface: trSignature
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trSignature
- Request parameters.

```
interface TrSignature extends TrCommonParam {/**new security password* /

newSecondSecret: string;}
```

8.3.2.2 Creating a Set-Username Event (with Security Key)

- Full name of the interface: trSignatureWithSign

- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trSignatureWithSign
- Request parameters.

```
interface TrSignatureWithSign {/**bufferg enerated by event body without signature, generatedb yt rSignature */

buffer: string;/**event signature* /

signature: string;}
```

8.3.2.3 Sending a Set-Security-Password Event

- Full name of the interface: signature
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/trSignature
- Request parameters.

```
interface SendTrCommonParam {/**Buffert ob es igned, converted to base64 string */

buffer: Buffer;/* /

buffer: Buffer;/**signature of thet ransaction* /

signature: string;

 /**the security signatureo ft he transaction* /

signSignature?: string;}
```

8.3.3 Setting the User Name Event

8.3.3.1 Creating a Set-Username Event

- Full name of the interface: trUsername
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trUsername
- Request parameters.

```
interface TrUsername extends TrCommonParam {/**usernames tring, uppera nd lower casel etters, numbers, underscores, 1-20 characters, cannot containt hen ameo f thec urrent chain* /

alias: string;}
```

8.3.3.2 Creating a Set-Username Event (with Security Key)

- Full name of the interface: trUsernameWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- interface url address: /api/transaction/trUsernameWithSign
- Request parameters.

```
interface TrUsernameWithSign {/**bufferg enerated by event body without signature, generatedb yt rUsername* /

buffer: string;/**event signature* /

signature: string;}
```

8.3.3.3 Sending a Set-Username Event

- Full name of the interface: username
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/username
- Request parameters.

```
interface SendTrCommonParam {/**buffer to be signed, converted to base64 string */

buffer: Buffer;/**signature of the transaction */

signature: string;

 /**the security signature of the transaction */

signSignature?: string;}
```

8.3.4 Registered Trustee Events

8.3.4.1 Creating a Registered Trustee Event

- Full name of the interface: trDelegate
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trDelegate
- Request parameters.

```
interface TrDelegate extends TrCommonParam {}
```

8.3.4.2 Creating a Registered Trustee Event (with Security Key)

- Full name of the interface: trDelegateWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trDelegateWithSign
- Request parameters.

```
interface TrDelegateWithSign {/**buffer generated by the event body without signature, generated by trDelegate */

buffer: string;/**event signature */

signature: string;}
```

8.3.4.3 Sending a Registered Trustee Event

- Full name of the interface: delegate
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/delegate
- Request parameters.

```
interface SendTrCommonParam {/**buffer to be signed, converted to base64 string */

buffer: Buffer;/**signature of the transaction */

signature: string;

 /**signature of the transaction */

signSignature?: string;}
```

8.3.5 Receiving Polling Events

8.3.5.1 Creating a Receive-Vote Event

- Full name of the interface: trAcceptVote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trAcceptVote
- Request parameters.

```
interface TrAcceptVote extends TrCommonParam {}
```

8.3.5.2 Creating A Receive-Vote Event (with Security Key)

- Full name of the interface: trAcceptVoteWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trAcceptVoteWithSign
- Request parameters.

```
interface TrAcceptVoteWithSign {/**buffer generated by event body without signature, generated by trAcceptVote */

buffer: string;/ **event signature */

signature: string;}
```

8.3.5.3 Sending and Receiving Vote Events

- Full name of the interface: acceptVote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/acceptVote
- Request parameters.

```
interface SendTrCommonParam {/**Convert the buffert hatn eeds as ignature into a base64 string.* /

buffer: Buffer;/**signature of thet ransaction* /

signature: string;

 /**the security signatureo ft he transaction* /

signSignature?: string;}
```

8.3.6 Rejecting Votes

8.3.6.1 Creating A Reject-Vote Event

- Full name of the interface: trRejectVote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trRejectVote
- Request parameters.

```
interface TrRejectVote extends TrCommonParam {}
```

8.3.6.2 Creating a Reject-Vote Event (with Security Key)

- Full name of the interface: trRejectVoteWithSign
- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trRejectVoteWithSign
- Request parameters.

```
interface TrRejectVoteWithSign {/**bufferg enerated from event body without signature, generatedb yt rRejectVote */

buffer: string;/**event signature* /

signature: string;}
```

8.3.6.3 Sending a Reject-Vote Event

- Full name of the interface: rejectVote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/rejectVote
- Request parameters.

```
interface SendTrCommonParam {/**Convert the buffert hatn eeds as ignature into a base64 string.* /

buffer: Buffer;/**signature of thet ransaction* /

signature: string;

 /**the security signatureo ft he transaction* /
```

```
signSignature?: string;}
```

8.3.7 Polling Events

8.3.7.1 Creating a Voting Event

- Full name of the interface: trVote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trVote
- Request parameters.

```
interface TrVote extends TrCommonParam {/**the numbero fa sset cast, 0-9a nd without decimal points, 0a llowed */

asset: string;/**the addresso ft he receiving account fort he event, base58 encoded hexadecimals tring */

recipientId: string;}
```

8.3.7.2 Creating a Voting Event (with Security Key)

- Full name of the interface: trVoteWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- interface url address: /api/transaction/trVoteWithSign
- Request parameters.

```
interface TrVoteWithSign {/****bufferg enerated from event body without signature, generatedb yt rVote* /
```

```
buffer: string;/**event signature */

signature: string;}
```

8.3.7.3 Sending and Receiving a Polling Event

- Full name of the interface: vote
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/vote
- Request parameters.

```
interface SendTrCommonParam {/**Convert the buffer that needs a signature into a base64 string. */

buffer: Buffer;/**signature of the transaction */

signature: string;

 /**the security signature of the transaction */

signSignature?: string;}
```

8.3.8 Publishing DApp Events

8.3.8.1 Creating a Release-DApp Event

- Full name of the interface: trDApp
- Callable methods: Http, Websocket, command line
- Call method: post

- Interface url address: /api/transaction/trDApp
- Request parameters.

```
interface TrDApp extends TrCommonParam {
  /**DAppidw ithout checksum, upper caseo r numeric,7  characters */

  newDAppid: string;

  /**The type of the DAppid, can onlyb e0  or 1,0  meanst he DAppidi sp aid, 1 meanst he DAppidi s free. */

  type: number;

  /**The numbero fb enefitsn eededt o purchaset he right to uset he DAppid( must be carried if the DAppidi sa  paid app, no need to carryi ti fi ti sa  free app),0 -9 andn o decimal points, must be greatert han0 .* /

  amount: string;

  /**the recipient account addresso ft he event, base58 encodedh exadecimal string */

  recipientId?: string;}
```

8.3.8.2 Creating an Issue-DApp Event (with Security Key)

- Full name of the interface: trDAppWithSign
- Callable methods: Http, Websocket, command line
- Call method: post
- interface url address: /api/transaction/trDAppWithSign
- Request parameters.

```
interface TrDAppWithSign {
```

```
/**buffer generated from event body without signature, generated by trDApp */
buffer: string;
/**event signature */
signature: string;}
```

8.3.8.3 Sending an Issue-DApp Event

- Full name of the interface: DApp
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/DApp
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64 string. */
  buffer: string;
  /**event signature */
  signature: string;
  /**the security signature of the transaction */
  signSignature?: string;}
```

8.3.9 DApp Purchase Events

8.3.9.1 Creating a Purchase-DApp Event

- Full name of the interface: trDAppPurchasing
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trDAppPurchasing
- Request parameters.

```
interface TrDAppPurchasing extends TrCommonParam {

  /**the recipient account address of the event, base58 encoded hexadecimal string */

  recipientId: string;

  /**the DAppid to which the certificate belongs, an uppercase acquisition array, 8 characters */

  DAppid: string;

  /**The type of the DAppid, can only be 0 or 1, 0 means the DAppid is a paid type, 1 means the DAppid is a free type. */

  type: number;

  /**the number of DAppid purchase assets */

  purchaseAsset: number;}
```

8.3.9.2 Creating a Purchase-DApp Event (with Security Key)

- Full name of the interface: trDAppPurchasingWithSign
- Callable methods: Http, Websocket, command line

- call method: post
- interface url address: /api/transaction/trDAppPurchasingWithSign
- Request parameters.

```
interface TrDAppPurchasingWithSign {
  /**buffer generated from event body without signature, generated by trDAppPurchasing */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.9.3 Sending a Purchase-DApp Event

- Full name of the interface: DAppPurchasing
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/DAppPurchasing
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64 string. */
  buffer: string;
  /**event signature */
  signature: string;
```

```
  /**the security signature of the transaction */

  signSignature?: string;}
```

8.3.10 Depositing Events

8.3.10.1 Creating a Deposition Event

- Full name of the interface: trMark
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trMark
- Request parameters.

```
interface TrMark extends TrCommonParam {

  /**the contents of thec ertificate,a sa na rbitrarys tring* /

  content: string;

  /**the type of thec ertificate,a sa na rbitrary string,u sedt od istinguish the certificate* /

  action: string;

  /**the DAppido ft he certificate, in uppercasel etters,8 characters */

  DAppid: string;

  /**The type of DAppid, can onlyb e0 or 1. 0m eans DAppidi sp aidt ype.1m eans DAppidi sf reet ype. */

  type: number;

  /**numbero fa ssets pent to purchase DAppid* /

  purchaseAsset?: number;}
```

8.3.10.2 Creating a Deposit Event (with Security Key)

- Full name of the interface: trMarkWithSign
- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trMarkWithSign
- Request parameters.

```
interface TrMarkWithSign {
  /**buffer generated from event body without signature, generated by trMark */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.10.3 Sending a Deposit Event

- Full name of the interface: mark
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/mark
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64 string. */
  buffer: string;
```

```
/**event signature */

signature: string;

/**the security signature of the transaction */

signSignature?: string;}
```

8.3.11 Asset Issuance Events

8.3.11.1 Creating an Asset Issuance Event

- Full name of the interface: trIssueAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trIssueAsset
- Request parameters.

```
interface TrIssueAsset extends TrCommonParam{

    /**the name of the issued asset, in uppercase letters, 3-5 characters */

    assetType: string;

    /**The total number of new assets issued, the number of assets consists of ten numbers from 0-9, the number of assets does not contain a decimal point and must be greater than 0. */

    expectedIssuedAssets: string;

    /**The address of the creation account of the new entitlement, base58 encoded hexadecimal string, this address must be given to the originating account of this event to transfer the master entitlement of this chain */
```

```
    recipientId: string;}
```

8.3.11.2 Creating an Asset Issuance Event (with Security Key)

- Full name of the interface: trIssueAssetWithSign
- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trIssueAssetWithSign
- Request parameters.

```
interface TrIssueAssetWithSign {
  /**buffer generated from event body without signature, generated by trMark */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.11.3 Sending an Asset Issuance Event

- Full name of the interface: issueAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/issueAsset
- Request parameters.

```
interface SendTrCommonParam {
```

```
/**Convert the buffert hatn eeds as ignature into ab ase64s tring. */
buffer: string;
/**event signature* /
signature: string;
/**the security signatureo ft he transaction* /
signSignature?: string;}
```

8.3.12 Asset Destruction Events

8.3.12.1 Creating an Asset Destruction Event

- Interface full name: trDestroyAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trDestroyAsset
- Request parameters.

```
interface TrDestroyAsset extends TrCommonParam{
    /**The number of assets to be destroyed, 0-9 and without decimal points, must be greater than 0. */
    amount: string;
    /**the name of the destroyed asset, in uppercase letters, 3-5 characters */
    assetType: string;
    /**the issuing account address of the asset, base58 encoded hexadecimal string */
    recipientId: string;}
```

8.3.12.2 Create an Asset Destruction Event (with Security Key)

- Full name of the interface: trDestroyAssetWithSign
- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trDestroyAssetWithSign
- Request parameters.

```
interface TrDestroyAssetWithSign {
  /**buffer generated from event body without signature, generated by trMark */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.12.3 Sending an Asset Destruction Event

- Interface full name: destroyAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/destroyAsset
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64 string. */
  buffer: string;
```

```
    /**event signature */

    signature: string;

    /**the security signature of the transaction */

    signSignature?: string;}
```

8.3.13 Asset Exchange Events

8.3.13.1 Creating an Asset Exchange Event

- Full name of the interface: trToExchangeAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trToExchangeAsset
- Request parameters.

```
interface TrToExchangeAsset extends TrCommonParam{

    /**the network identifier of the source chain of assets to be exchanged, consisting of upper case letters or numbers, 5 characters, the last bit is a check digit */

    toExchangeSource: string;

    /**the network identifier of the asset source chain to be exchanged, in uppercase letters or numbers, 5 characters, the last bit is a check digit */

    beExchangeSource: string;

    /**the name of the asset source chain to be exchanged, in lowercase letters, 3-8 digits */

    toExchangeChainName: string;
```

```
    /**the name of the source chain of the asset being exchanged, in lowercase letters, 3-8 bits */

    beExchangeChainName: string;

    /**the name of the asset to be exchanged, in uppercase, 3-5 characters */

    toExchangeAsset: string;

    /**the name of the asset being exchanged, in uppercase, 3-5 characters */

    beExchangeAsset: string;

    /**The number of assets to be exchanged, 0-9 and without decimal points, must be greater than 0. */

    toExchangeNumber: string;

    /**used as the denominator for the exchange ratio with asset, a positive integer. exchangedAsset = exchangedAsset *exchange ratio */

    prevWeight: string;

    /**the numerator of the exchange ratio with asset, a positive integer. exchanged asset = exchanged asset * exchange ratio */

    nextWeight: string;

    /**cryptographic key set: if the key is filled, the event receiving the asset exchange must carry a signature pair generated by some key.    */

    ciphertexts?: string[];}
```

8.3.13.2 Creating an Asset Exchange Event (with Security Key)

- Full name of the interface: trToExchangeAsset
- Callable methods: Http, Websocket, command line

- call method: post
- interface url address: /api/transaction/trToExchangeAsset
- Request parameters.

```
interface TrToExchangeAssetWithSign {
  /**buffer generated from event body without signature, generated by trMark */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.13.3 Sending an Asset Exchange Event

- Full name of the interface: trToExchangeAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/trToExchangeAsset
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64 string. */
  buffer: string;
  /**event signature */
  signature: string;
  /**the security signature of the transaction */
  signSignature?: string;}
```

8.3.14 Acceptance of an Asset Exchange Event

8.3.14.1 Accepting an Asset Exchange Event

- Full name of the interface: trBeExchangeAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/trBeExchangeAsset
- Request parameters.

```
interface TrBeExchangeAsset extends TrCommonParam{

    /**to event signature, 128-byte hexadecimal string */

    transactionSignature: string;

    /**To exchange the number of assets, the number of assets consists of ten numbers from 0-9, the number of assets does not contain a decimal point and must be greater than 0. */

    beExchangeNumber: string;

    /**The number of exchanged assets consists of ten numbers from 0-9. the number of assets does not contain a decimal point and must be greater than 0. */

    toExchangeNumber: string;

    /**encryption key: if the key is filled in for an asset exchange event, it must carry the key specified for an asset exchange event to generate a key signature pair. */

    ciphertext?: string;

    /**to event's originating account address, base58 encoded hexadecimal string */

    recipientId: string;}
```

8.3.14.2 Creating an Accept-Asset-Exchange Event (with Security Key)

- Full name of the interface: trBeExchangeAssetWithSign
- Callable methods: Http, Websocket, command line
- call method: post
- interface url address: /api/transaction/trBeExchangeAssetWithSign
- Request parameters.

```
interface TrBeExchangeAssetWithSign {
  /**buffer generated from event body without signature, generated by trMark */
  buffer: string;
  /**event signature */
  signature: string;}
```

8.3.14.3 Sending an Accept-Asset-Exchange Event

- Full name of the interface: destroyAsset
- Callable methods: Http, Websocket, command line
- Call method: post
- Interface url address: /api/transaction/send/destroyAsset
- Request parameters.

```
interface SendTrCommonParam {
  /**Convert the buffer that needs a signature into a base64            /
  buffer: string;
```

```
/**event signature */

signature: string;

/**the security signature of the transaction */

signSignature?: string;}
```

8.4 Instructions for Using the Node Management Interface

8.4.1 Safety Close of Node

- Full name of the interface: safetyClose
- Interface abbreviation: sfc
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/safetyClose
- Request parameters.

```
interface SafetyClose {/**verifyType: 001 node owner verification, 002 administrator verification */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**Whether the machine needs to be shut down: true means shutdown and false means no shutdown */

isShutdown?: boolean;}
```

8.4.2 Setting Node Password

- Full name of the interface: setSystemKey
- Interface abbreviation: ssk

- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/setSystemKey
- Request parameters.

```
interface SetSystemKey {/**old password for the node */

systemKeyOld: string;/**new password for the node */

systemKeyNew: string;/**Whether to decrypt the new password in asymmetric way (true: use asymmetric way to decrypt, false: do not use asymmetric way to decrypt, plaintext transmission) */

newKeyDecryptEnable?: boolean;}
```

8.4.3 Verifying Node Password

- Full name of the interface: setSystemKey
- Interface abbreviation: ssk
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/setSystemKey
- Request parameters.

```
interface VerifySystemKey {/**node password */

systemKey: string;}
```

8.4.4 Adding Node Administrator

- Full name of the interface: addSystemAdmin
- Interface abbreviation: asa
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/addSystemAdmin
- Request parameters.

```
interface AddSystemAdmin {/**node password */

systemKey: string;/**node administrator address: please refer to <Node Administrator> for administrator description */

systemAdminAddress: string;}
```

8.4.5 Getting Node Administrator

- Full name of the interface: getSystemAdmin
- Interface abbreviation: gsa
- Callable methods: Http, Websocket, Command Line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemAdmin
- Request parameters.

```
interface GetSystemAdmin {/**node password */

systemKey: string;/**node administrator address: if there is an incoming address, then return the information of that administrator address; if there is no incoming, then return the information of all administrators. */
```

```
systemAdminAddress?: string;}
```

8.4.6 Verify Node Administrator

- Full name of the interface: verifySystemAdmin
- Interface abbreviation: vsa
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/verifySystemAdmin
- request parameters.

```
interface VerifySystemAdmin {/**encrypted administrator address */

cryptoAdminAddress: string;}
```

8.4.7 Deleting Node Administrator

- Full name of the interface: delSystemAdmin
- Interface abbreviation: dsa
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/delSystemAdmin
- Request parameters.

```
interface DelSystemAdmin {/**node password* /

systemKey: string;/**node administratora ddress: please refert o< Node Administrator>
fora dministrator description* /
```

```
systemAdminAddress: string;}
```

8.4.8 Resetting Node Administrator

- Full name of the interface: resetSystemAdmin
- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/resetSystemAdmin
- Request parameters.

```
interface ResetSystemAdmin {/**node password */

systemKey: string;/**node administratora ddress: please refert o< Node Administrator>
fora dministrator description* /
systemAdminAddresses: string[];}
```

8.4.9 Binding Node Accounts

- Full name of the interface: bindingAccount
- Interface abbreviation: ba
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/bindingAccount
- request parameters.

```
interface BindingAccount {/**node password* /
```

```
systemKey: string;/**trustee privatek ey aftere ncryption* /

cryptoSecret: string;/**encrypted TrusteeS ecurity Key* /

secondSecret?: string;}
```

8.4.10 Getting Node Trustee

- Full name of the interface: getSystemDelegate
- Interface abbreviation: gsd
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemDelegate
- Request parameters.

```
interface GetSystemDelegate {/**node administratora ddress: please refert o< Node Administrator>f or administratord escription */

verifyType: string;/**check value: depending on thea uthorityo ft he node visitor, passwordc heck for node ownera nd addressc heck fora dministrator */
verifyKey: string;}
```

8.4.11 Querying All Forgers Registered by the Node

- Full name of the interface: getInjectGenerators
- Interface abbreviation: gsd
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/getInjectGenerators
- Request parameters.

```
interface GetInjectGenerators {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.12 Query Details Of the Forger Registered by the Node

- Full name of the interface: getSystemDelegateDetail
- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/getSystemDelegateDetail
- Request parameters: /api/system/getSystemDelegateDetail

```
interface GetSystemDelegateDetail {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**受托人地址 */

address: string;}
```

8.4.13 Getting Node Details

- Full name of the interface: getSystemNodeInfo

- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/getSystemNodeInfo
- Request parameters.

```
interface GetSystemNodeInfo {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.14 Node Information Query

- Full name of the interface: miningMachineInfo
- Interface abbreviation: mmi
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/miningMachineInfo
- request parameters.

```
interface MiningMachineInfo {/**node administrator address: pleaser efer to <Node Administrator>f or administratord escription */

verifyType: string;/**check value: depending on thea uthorityo ft he node visitor, passwordc heck for node ownera nd addressc heck fora dministrator */

verifyKey: string;}
```

8.4.15 Setting Node Configuration Information

- Full name of the interface: setSystemConfig
- Interface abbreviation: ssc
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/setSystemConfig
- Request parameters.

```
interface SetSystemConfig {/**node administrator address: please refert o< Node Administrator>f or administratord escription */

verifyType: string;/**check value: depending on thea uthorityo ft he node visitor, passwordc heck for node ownera nd addressc heck fora dministrator */

verifyKey: string;/**configuration information: allp arametersh ere can be empty* /

config: AllPartial<Config.ConfigRevisable>;}
```

8.4.16 Getting Node Configuration Information

- Full name of the interface: getSystemConfigInfoDetail
- Interface abbreviation: gsci
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemConfigInfoDetail
- Request parameters.

```
interface GetSystemConfigInfoDetail {/**node administrator address: please refer to <Node Administrator> for administrator description */
```

```
verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.17 Getting Node State (Real-Time Information)

- Full name of the interface: getRuntimeState
- Interface abbreviation: grs
- Callable methods: Http, Websocket, command line, Grpc
- call method: post
- interface url address: /api/system/getRuntimeState
- Request parameters.

```
interface GetRuntimeState {/**node administrator address: please refer to <Node Administrator> for administrator description */
verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.18 Getting Node Access Statistics

- Full name of the interface: getSystemMonitor
- Interface abbreviation: gsm
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemMonitor

- Request parameters.

```
interface GetSystemMonitor {/**node administratora ddress: please refert o< Node Administrator> fora dministrator description* /

verifyType: string;/**check value: depending on thea uthorityo ft he node visitor, passwordc heck for node ownera nd addressc heck fora dministrator */

verifyKey: string;/**Specify thet ypeo f access, including thet raffic of accessing IP, numbero ft imes, numbero f accessing interfaces, numbero fb locks, eventd ata, etc. */

monitorType?: string;/**The numbero f queries,f or example, limit=10, meanst hat1 0 data can be queried.* /

limit?: number;/**Query start position, fore xample,o ffset=  0, meanss tart querying fromr ow 1. */

offset?: number;}
```

8.4.19 Getting Running Log Type of the Node

- Full name of the interface: getSystemLoggerType
- Interface abbreviation: glt
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemLoggerType
- Request parameters.

```
interface GetSystemLoggerType {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */
```

```
verifyKey: string;}
```

8.4.20 Getting the List of the Node Running Log

- Full name of the interface: getSystemLoggerList
- Interface abbreviation: gll
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemLoggerList
- Request parameters.

```
interface GetSystemLoggerList {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**log type */

loggerType: string;}
```

8.4.21 Getting Contents of the Node Running Log

- Full name of the interface: getSystemLoggerDetail
- Interface abbreviation: gld
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getSystemLoggerDetail

- Request parameters.

```
interface GetSystemLoggerDetail {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**log file name */

loggerName: string;/**the number of queries: e.g. limit=10, means 10 data can be queried. */

limit?: number;/**query start position, e.g. offset = 0, means the query starts from the 1st row. */

offset?: number;/**the string to search */

searchString?: string;/**the way to read the file */

readFileType?: {

    readFileAsync = 0,

    createReadStream = 1,};}
```

8.4.22 Deleting the Node Running Log

- Full name of the interface: delSystemLogger
- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/delSystemLogger
- Request parameters.

```
interface DelSystemLogger {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/** log file name */

loggerName: string;}
```

8.4.23 Getting the Node Email Address

- Full name of the interface: getEmailAddress
- Interface abbreviation: gea
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/getEmailAddress
- Request parameters.

```
interface GetEmailAddress {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**the email address to check */

emailAddress?: string;}
```

8.4.24 Setting the Node Email Address

- Full name of the interface: setEmailAddress

- Interface abbreviation: sea
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/setEmailAddress
- Request parameters.

```
interface SetEmailAddress {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**mailbox receive address */

emailToAddress: string;/**mailbox send address */

emailFromAddress: string;/**mailbox configuration */

emailConfig: {

    /**mailbox configuration type: POP3/SMTP/IMAP */

    type: string;

    /**mailboxConfig host */

    host?: string;

    /**mailbox configuration port */

    port?: number;

    /**Whether to enable mailbox security control */

    secureConnection?: boolean;

    /**Whether to enable ssl */
```

```
    ssl?: boolean;

    /**Whether tls is enabled */

    tls?: boolean;

    /**relay mailbox configured information */

    auth?: {

        /**relay mailbox configured username */

        user: string;

        /**relay mailbox configured password */

        pass: string;

    };};}
```

8.4.25 Verifying Node Trustees by Node Private Key

- Full name of the interface: verifysystemsecret
- Interface abbreviation: vss
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/verifySystemSecret
- Request parameters.

```
interface VerifySystemSecret {/**trustee private key after encryption */

cryptoSecret: string;}
```

8.4.26 Setting Node Access Whitelist

- Full name of the interface: setSystemWhiteList
- Interface abbreviation: swl
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/setSystemWhiteList
- Request parameters.

```
interface SetSystemWhiteList {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**white list */

whiteList: string[];}
```

8.4.27 Getting Node Access Whitelist

- Full name of the interface: getSystemWhiteList
- Interface abbreviation: gwl
- Callable methods: Http,Websocket,command line,Grpc
- Call method: post
- Interface url address: /api/system/getSystemWhiteList
- Request parameters.

```
interface GetSystemWhiteList {/**node administrator address: please refer to <Node Administrator> for administrator description */
```

```
verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.28 Deleting Node Access Whitelist

- Full name of the interface: delSystemWhiteList
- Interface abbreviation: dwl
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/delSystemWhiteList
- Request parameters.

```
interface DelSystemWhiteList {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/* *white list */

whiteList: string[];}
```

8.4.29 Getting Network-Related Information About a Node Process

- Full name of the interface: getProcessNetwork
- Interface abbreviation: gpn
- Callable methods: Http, Websocket, Command Line, Grpc
- Call method: post

- Interface url address: /api/system/getProcessNetwork
- Request parameters.

```
interface GetProcessNetwork {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**query number: for example, limit=10 means you can query 10 pieces of data.    */

limit?: number;/**query start position: for example, offset = 0 means the query starts from row 1. */

offset?: number;/**process type */

processType?: string;}
```

8.4.30 Getting Node Process CPU Information

- Full name of the interface: getProcessCPU
- Interface abbreviation: gpc
- Callable methods: Http,Websocket,command line,Grpc
- Call method: post
- Interface url address: /api/system/getProcessCPU
- Request parameters.

```
interface GetProcessCPU {/**node administrator address: please refer to <Node Administrator> for administrator description */
```

```
verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**query number: for example, limit=10 means you can query 10 pieces of data. */

limit?: number;/**query start position: for example, offset = 0 means the query starts from row 1. */

offset?: number;/**process type */

processType?: string;}
```

8.4.31 Getting Node Process Memory Information

- Full name of the interface: getProcessMemory
- Interface abbreviation: gpm
- Callable methods: Http, Websocket, Command Line, Grpc
- Call method: post
- Interface url address: /api/system/getProcessMemory
- Request parameters.

```
interface GetProcessMemory {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;/**query number: for example, limit=10 means you can query 10 pieces of data. */

limit?: number;/**query start position: for example, offset = 0 means the query starts from row 1. */
```

```
offset?: number;/**process type */

processType?: string;}
```

8.4.32 Sending Node Status at Regular Intervals

- Full name of the interface: systemStatus
- Interface abbreviation: ess
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/systemStatus
- Request parameters.

```
interface SystemStatus {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.33 Timed Sending of Node CPU, Memory and Network Information

- Full name of the interface: systemProcess
- Interface abbreviation: esp
- Callable methods: Http, Websocket, command line, Grpc
- Call method: post
- Interface url address: /api/system/systemProcess
- Request parameters.

```
interface SystemProcess {/**node administrator address: please refer to <Node Administrator> for administrator description */

verifyType: string;/**check value: depending on the authority of the node visitor, password check for node owner and address check for administrator */

verifyKey: string;}
```

8.4.34 Getting Information About a Node

- Full name of the interface: getSystemInfo
- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/getSystemInfo
- Request parameters: None

8.4.35 Getting Node Status

- Full name of the interface: getMachineStatus
- Interface abbreviation: none
- Callable methods: Http,Websocket
- Call method: post
- Interface url address: /api/system/getMachineStatus
- Request parameters: None

9. Application Tools

9.1 Instant Messenger-Secret Chat

Secret Chat is BFMeta's first blockchain-based decentralized mobile social tool for intra-group private message exchange. By forming a group of nodes into a logical group or a logical node, the concept of "virtual group" is built, and when you need to send a message to everyone in the group, you only need to send a message to this virtual node, realizing the private message sending and receiving of many-to-many, and building a structural model of group private message exchange under blockchain. It is a fast and secure structure model for message exchange in blockchain, and solves the problem of restricted group sending from point to point in blockchain.

It not only supports sending text, pictures, voice and video on the chain, but also supports sharing and witnessing life to friends through "Square".

9.2 Five Knocks

In the context of the epidemic normalization and the era of economic and technological globalization, the market environment is becoming more and more competitive, and the survival of enterprises is becoming more and more difficult. In order to meet the harsh survival challenges in the modern business environment, finding ways to improve the efficiency of enterprise management and reduce the cost of business operations is the top priority for enterprises. One of the effective ways to solve this problem is to change the office model.

Under the traditional centralized office, all employees have fixed departments, fixed leaders, fixed colleagues, fixed workstations, mature work patterns, and work in a command-based manner. In the office mode with the leader as the central node, everything is designed to enhance the efficiency of management as the first priority. The problems brought by this model are:

1) Decision makers in the organization have limited access to information

2) Individuals in the organization have limited knowledge of information, and in a centralized and hierarchical organizational model, information transfer is easily distorted.

Five Knocks is the first distributed collaborative office tool created by BFMeta, which breaks the centralized office method of the past for a long time and is a distributed collaborative office tool in the Web3.0 value Internet era built with a new decentralized and distributed idea.

This tool, with the blockchain technology of Distributed Autonomous Organization (DAO) as the consensus rules for the autonomy of each organizational department of the enterprise, realizes the digital management of the enterprise; with the Secret Chat as the core technology of collaborative office, realizes the cross-organizational and cross-regional communication of members of each department without barriers.

Although the hierarchy still exists, the organizational structure is no longer centered on management, but on employees and users. In such an organizational structure, the traditional hierarchical relationship no longer exists, and the superior, as the reporting object of the subordinate, is only a way to collect information and only intervene a little when necessary. Although enterprise

management is still hierarchical, user interaction has been decentralized. The responsibilities and asset of all parties are clarified through negotiation, and an open and transparent technical framework and procedural rules are formed in the form of smart contracts, stored on the blockchain, and used as a basis to complete cooperation between individuals.

9.3 Eye of God

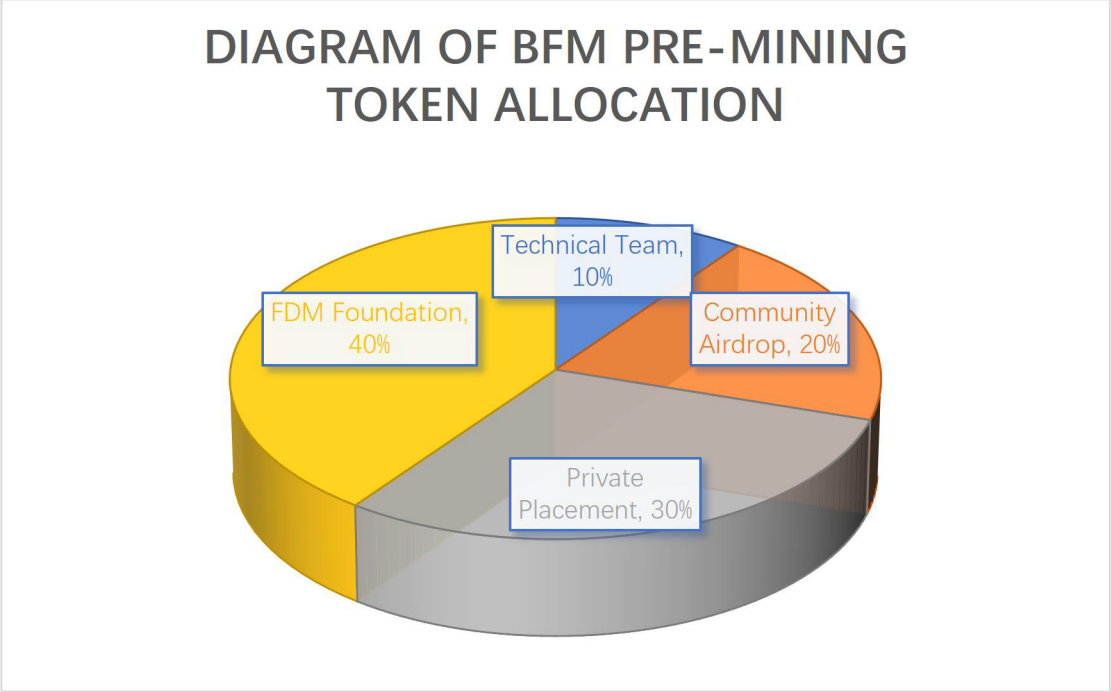
Blockchain technology itself is complex, and even technical bullies need to spend a lot of time learning and deploying it. For example, for Bitcoin and Ether, which are currently mainstream in the market, only people with professional mining machine deployment ability can participate in on-chain consensus and get rewarded. This is unfriendly to white users.

To facilitate ordinary users to create blockchains and deploy individual nodes, BFMeta has developed a visual chain management tool-Eye of God. Through a visual and customizable configuration interface, it is easy for each user to custom build their own blockchain and manage it.

10. FDM Foundation and Token Distribution

BFMeta is distributed globally by the Future Development Metaverse (FDM) Foundation, a Singapore-registered foundation, and BFM is the native Token of BFMeta. The total number of BFMs issued is 1,100,000,000, of which 100,000,000 are pre-mining Token and 1,000,000,000 are generated through mining.

The distribution ratio of pre-mining Token is as follows: 10% for technical team rewards, 20% for global major community airdrop, 30% for private placement, 40% for FDM Foundation to support future ecological development and global developer support program use.



11. Disclaimer

This White Paper is not a recommendation that you purchase any BFMeta, nor is it a document to which you should refer for any contract or purchase. This white paper does not constitute an offer to buy or sell, nor does it constitute a contract or commitment of any kind. BFMeta does not intend to constitute securities or any other regulated product in any country or jurisdiction.

This White Paper is not the basis for a prospectus or any other offering document for securities and is not intended to constitute an offering or solicitation of securities or any other regulated product in any country or jurisdiction. This White Paper has not been reviewed by any regulatory authority in any country or jurisdiction.

You acknowledge and agree that BFMeta does not have the following functions:

1. Represent the asset, control or obligations of BFMeta or any other institution in any jurisdiction, or the right to participate in or control the application of decisions made by the foregoing.

2. Represent any type of investment.

3. Represents any marketable security that has intrinsic value or market price.

4. Represents any commodity or asset that any person is obligated to redeem, or to purchase.

By participating in the Program, the participant acknowledges that he or she

understands and agrees to the terms and conditions set forth in these Terms and Conditions and accepts the potential risks at his or her own risk.

1. Market Risk: If the overall cryptocurrency market is overvalued, then investment risk will increase and participants may have high expectations of price growth for the Program that may not be realized.

2. Systemic risk: This refers to force majeure factors, including but not limited to natural disasters political unrest, etc.

3. Regulatory risk: The trading of cryptocurrencies is highly uncertain, and due to the lack of strong regulation in the field of cryptocurrency trading, cryptocurrencies are subject to the risk of sharp rises and falls, etc. Individual participants who are inexperienced in the market may find it difficult to resist the asset shock and psychological pressure caused by market instability.

4. Project risk: The team will spare no effort to achieve the goals mentioned in the white paper, and now has a more mature business model, however, due to the unpredictable development trend of the industry as a whole, the existing business model may not match well with the market demand, thus making it difficult to achieve profitability. At the same time, as this white paper may be updated with the implementation of project details, if the updated details of the project are not accessed by the participants of this program in a timely manner, the participants will have insufficient knowledge due to information asymmetry, thus affecting the subsequent development of the project.

5. Technical risks: The project is based on cryptographic algorithms, and the rapid development of cryptography also brings potential risks of being cracked; blockchain, distributed storage and other technologies support the core business

development, and the team cannot fully guarantee the implementation of the technology; during the project update process, vulnerabilities may be found to exist, which can be compensated by releasing updates, but the extent of the impact caused by vulnerabilities cannot be guaranteed.

6. Hacking and crime risk: In terms of security, electronic tokens are anonymous and difficult to trace, which are vulnerable to hacking or used by criminals, or may involve illegal asset transfer and other criminal acts.

7. Policy risk: At present, the international regulatory policy for blockchain projects and financing with virtual currency parties is still unclear, and there is a certain possibility of loss to participants due to policy reasons.

8. Unknown risks: With the continuous development of blockchain technology, there may be some risks that cannot be predicted at present.

This White Paper makes no representations or warranties that the information, representations, opinions or other matters described or communicated in it in connection with the Program are correct or complete, nor does it make any representations or warranties as to the results or reasonableness of any forward-looking or conceptual statements, and the absence of representations and warranties is not limited to the foregoing. Nothing in this White Paper shall constitute or be deemed to constitute any promise or representation as to the future.

To the full extent permitted by applicable law, we will not be liable or responsible for any loss or damage arising out of or in connection with any action taken by any person in accordance with this White Paper, whether by negligence, default or lack of care.

Participants are requested to fully understand the background and overall framework of the team before participating and to participate rationally.

BFMeta reserves the right to amend and change the content of this white paper at any time.

BFMeta