CXC

Design Instructions for CXC Technology

V 1.0

Table of Contents

1. Introduction

Originally CXC is designed to be a new incubator of blockchain applications, and a computing, development and application platform, which surpasses Ethereum and is Turing sound.

CXC account system adopts a UTXO model to separate account

transactions from data via an independent ChainDB under the premises of safety and security, so that the retrieval mode of block transaction is driven by database rather than memory, and the system throughput and scalability are significantly improved thanks to a message-driven asynchronous architecture and ChainDB-based scenarized interaction mode via smart contract. A single node at the bottom layer can reach 2500+TPS and deal with 1 million transactions per section by centralized means. Each transaction and the largest block can be 16M, and supports unlimited DB and chaining data, to satisfy large-scale commercial applications' requirements on high throughput, low delay and scalability.

In addition to built-in native token and native assets, CXS also supports users to issue his own digital assets, and completely decentralized chaining assets exchange functions. For the long-term development of CXC, CXC software store is designed for developers to develop and issue Dapp based on CXC, forming a benign ecological development situation of developer, user and CXC.

CXC is an integrated application of technologies such as distributed data storage, point-to-point transmission, consensus mechanism, and encryption algorithm. Relying on new features such as POA consensus mechanism, fusion of main chain and side chains, power mining, N*Raid5 fragmented blockchain storage, on CXC, users can relocate existing applications at the minimal cost while issuing assets and developing applications just as the Ethereum supports.

Any unit and individual may leverage CXC technologies to rapidly develop Dapp, issue digital assets, complete token-token exchange which is completely decentralized. The main technical features of CXC are as follows:

Feature Ⅰ: bottom layer with a transaction speed of 2,500 transactions per second, which may be further improved to millions of transactions per second by multiple nodes and zoning.

Feature II: rapidly and freely issue digital assets and develop Dapp.

Feature III: completely decentralized atom transaction and chaining assets exchange.

Feature IV: 15s to form a block, single transaction of data up to 2M, and

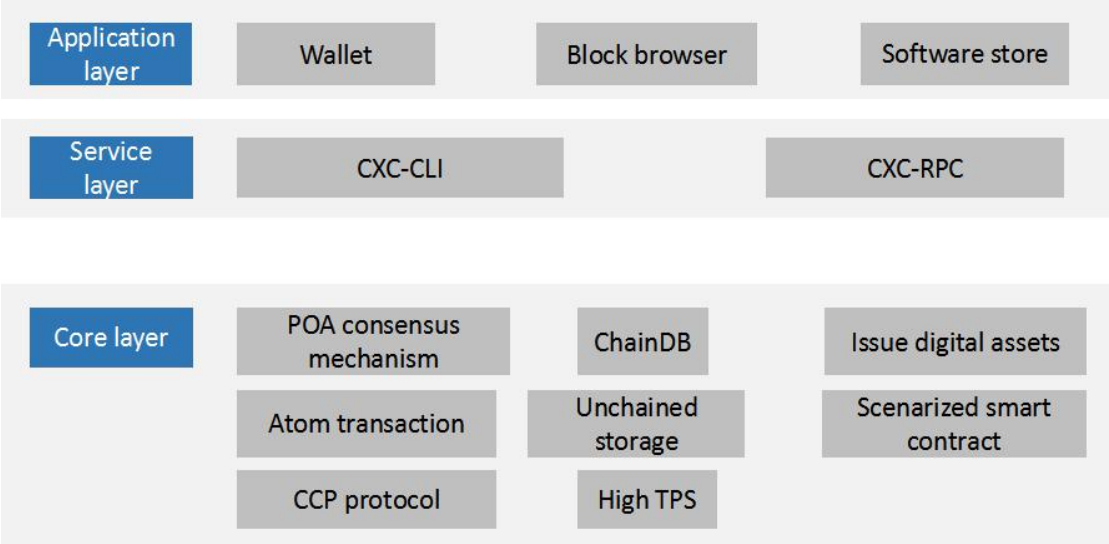
single block up to 16M.

Feature V: unique ChainDB, and native support to scenarized smart contract, directly open the chaining data interface, and free customization and development of CXCDapp based on known mature development tools.

Feature VII: support cross-chain assets transfer and exchange, data synchronization of currencies out of chain, and direct transactions with all currencies on CXC after synchronization.

Feature VIII: first commercial-grade unchained and extended storage which is completely decentralized and similar to N*RAID5.

2. Overall architecture



Overall Architecture of CXC

2.1 Core layer

2.1.1 Account assets system

CXC supports native blockchain assets, and adopts UTXO model. In CXC, assets can be cited by any of the following three methods:

- Assets name, selected upon issuing, 32 bytes maximally, case insensitive.
- selltxid, including the process of transacting assets issued.

- Assets id, created upon issuing as a unique identification.

An asset can be created as opening or close. If it is opened, the more assets can be issued subsequently.

2.1.2 Atom transaction

The atom transaction of CXC supports native assets, token assets and cross-chain assets transfer, direct exchange of chaining assets. During atom transaction, commission charge will be deducted from the native assets.

2.1.3 Consensus mechanism

CXC simplifies and optimizes traditional POW, and adopts POA consensus mechanism, which will be introduced hereinafter in details.

2.1.3ChainDB

To improve the performance of blockchain, the concept of ChainDB is put forward and designed by us as the first example in the industry. ChainDB have account transactions separated from data to maintain the TPS while guaranteeing large-capacity chaining data. ChianDB provee API de nivel superior para almacenamiento universal de datos, and CXC, to a certain degree, can be used for large-scale commercial information system as a general distributed database.

2.1.4 Cross-chain protocol

To support the cross-chain value transmission of digital assets, CXC design a cross-chain protocol (CCP).

Targeting at each asset to be transmitted across chains on the target chain, in CXC, a corresponding token will be issued for circulation of target assets in CXC. Such a token is identified as TAT (Third-party Asset Token). In the following, with ETH as an example, the whole process of cross-chain digital

assets transaction will be described in details.

1. Preparation: issue a TAT with ETH as the identifier, whose initial value is 0, and open is true (increased issue model), and transactions on both sides are controlled by CCP that no user can use it;

2. Transfer-in of cross-chain digital assets: when an Ethereum user A intends to transfer his tokens on Ethereum to CXC, he will obtain an Ethereum address in CXC wallet. When the CCP finds that the user is recharging the associated address, it will generate an ETH increased issue transaction in correspondence, and issue ETH tokens of the same amount to user A;

3. Transfer-out of cross-chain digital assets: when a user withdraws digital assets, a transaction will be issued to the CCP black hole address, a certain amount of ETH will be transferred in, and the receiving address E of Ethereum will be designated in the transaction. When CCP finds this transaction, it will make transfer to the address E, and the transaction succeeds when it is confirmed by Ethereum network.

In the whole process, CCP will verify each transaction. In case of rolling back or hard branching in the target network, transactions which have taken place in CXC system will not roll back with it.

2.2 Service layer

2.2.1 CXC-CLI

Command line interface of CXC used to call out CXC-RPC to execute commands of CXC.

2.2.2CXC-RPC

WEB interface service of CXC to provide CXCRPC service.

2.3 Application layer

- Wallet: CXC wallet for assets transaction, etc.
- Block browser: to check block information of CXC.
- CXC store: to run official and third-party DAPPs.
- Cross-chain assets transfer Dapp: CXC users can have assets out of the chain transformed into CXC assets, and have them transacted in CXC.
- Assets exchange Dapp: as one side locks assets and creates exchange code, the other side, namely the user, carries out completely decentralized chaining atom transaction with his own assets for the first users' assets via the exchange code.

3. Consensus mechanism

3.1 Node

According to the functions of different links in the transaction process, CXC divides the roles of nodes into four categories logically, so that different types of nodes can focus on different types of workloads.

- General node. A general wallet user constitutes a general node, and supports basic functions such as send, query, trade, and asset exchange. All blocks are synchronized;
- Consensus node, which participates in network consensus and generates blocks;
- Transaction node, which takes major charge of token-token transactions while participating in consensus.
- Light node, which does not synchronize all block data, but supports basic functions such as transfer and receive.

3.2 POA (Proof Of Alternate) consensus mechanism

In traditional blockchain system, such as Bitcoin system, to determine the attribution of accounting rights, Proof of Work is sued: each node in the

network is engaged to solve a moderately difficult cryptographic problem by following methods: SHA256 Hash operation of all data in the block to obtain a Hash value smaller than the given target value. The block also contains a Nonce, which is gradually increased to find the correct Hash value. The cryptographic puzzle is designed in such as way that every 10mins, a puzzle answer will be found. Once the correct Hash value is found, it will be broadcast by the node in the network and is easy to be verified by other nodes in the network. Node can carry out SHA256 operation for data in the blocks received to determine the Hash value. In this process, POW consumes CPU. So to modify a block, we need to redo it, and POW for all blocks afterward, which means that the Bit network is more prone to the most honesty chains as long as most nodes in the network are honest.

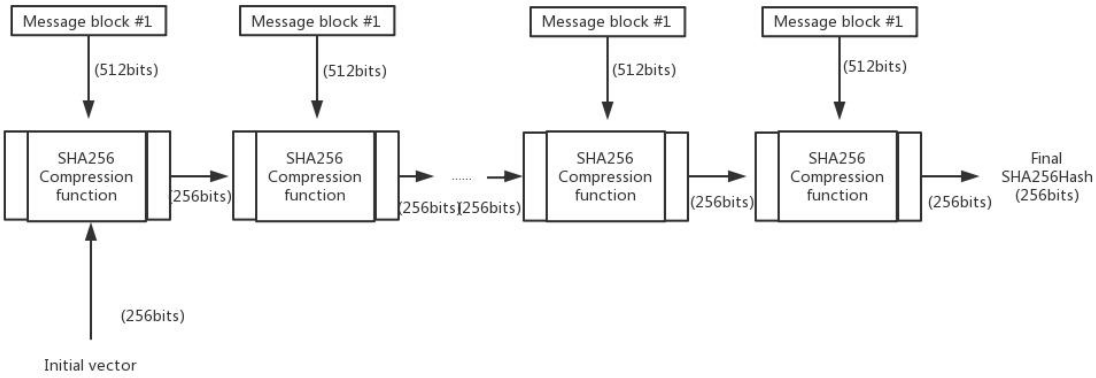
But traditional POW consensus mechanism has a higher requirement on computing power and network environment, consumes a lot of resources while the high computing power may attack network. Therefore, CXC makes improvements against POW algorithm mechanism and forms POA consensus mechanism.

In POA consensus mechanism, the traditional POW function SHA256 is reserved:

SHA256 comes from the famous SHA family. SHA (Secure Hash Algorithm) is a kind of cryptographic Hash function issued by NIST. The first member officially named SHA was issued in 1993, and 2 years later, the famous SHA-1 was issued, followed by another 4 variants, including SHA224, SHA256, SHA384 and SHA512. Those algorithms are also named as SHA2. SHA256 algorithm is a kind from SHA2 algorithm cluster. For messages shorter than 264 digits, SHA256 will produce a 256-digit message abstract. SHA256 is designed with the general features of cryptographic Hash function and serves as the major cryptographic Hash function to structure a blockchain. Whether it is the header information or transaction data of the block, this Hash function is used to calculate the Hash value of related data, and make sure data is integral. Meanwhile, in CXC system, a POW consensus mechanism is designed based on SHA256 Hash value searching for given prefix; SHA256 is also used to structure the address and identify different users.

SHA256 is an iterative Hash function in Merkle-Damgard structure, whose

calculation process consists of two stages: message pre-processing and main circulation. In the stage of message pre-processing, message filling and extended filling are completed by transforming the input original message into n message blocks of 512 bits, and processing each message block with SHA256 compression function. The calculation flow of SHA256 is indicated in the figure below. It is an iterative calculation process that when the last message block (the nth block) is processed, the final output value shall be the SHA256 value of the input original value.



Calculation Flow of SHA256

Compared with traditional POW consensus algorithm, we select consensus nodes randomly and crosswise to pack transaction, sign and issue block, and the one who issues the block is not allowed to continuously participate in the packing and signing (unless there is only online consensus node) to make sure the blockchain operates at high speed, to save resources and guarantee safety. The blockchain issuer maintained by POA consensus algorithm is required to be online 100%, and blocks issued will be verified by the next 6 block issuers. All block issuing awards will be locked in the gain pool of computing power and shared by users according to their contributions to computing power (coin holding computing power +dual-peak computing power +bonus computing power for invitation + bonus computing power for duration of coin holding).

3.3 Improvement of TPS

3.3.3 Causes of failure to improve TPS in traditional POW mechanism

Traditional theories hold the ground that POW is unable to improve TPS remarkably. This is the not the fact. CXC has found following major causes leading to the failure of remarkable improvement of TPS in POW mechanism:

- All wallet transactions are saved in memory, which shortens start-up time but memory is consumed increasingly;
- The adoption of UTXO model causes many operations “comprehensively scanning” each transaction in the wallet at a low efficiency, whether it is a new or old transaction.
- All transactions in the wallet are completely saved, including random “metadata” without any significance from the perspective of node, though it has been saved to the blockchain on disc. For this reason, wallet slows down when 20,000 transactions are stored on the node. After millions of transactions, the wallet may stop working in fact.

3.3.2 Causes of failure to improve TPS from the perspectives of Bitcoin and Ethereum

The failure of Bitcoin node to improve TPS shall not be attributed to the POW consensus mechanism (essentially, POW will complete consensus in blocking time), but the three reasons we mentioned above. Though Bitcoin network can process and store millions of transactions or more which are unrelated with it easily, those transactions are carried out in the disc instead of memory. But when specified to a single node, when processing transactions related to itself, it has to browse over and “compressively scan” all files on the block at a low efficiency because of the adoption of UTXO model, n order to obtain unspent output as the input. The comprehensive scanning covers all transactions in the memory, metadata, and block files, which are unstructured

data resulting in low retrieval speed. So, to a single node, the TPS is only 3 TPS.

Processing mechanism of Ethereum node: the design of Ethereum has sacrificed reliability and doesn't adopt UTXO model taken into account all its shortcomings, but its related transactions are still unstructured data. In addition, as the smart contract is designed to be global, all nodes are required to browse over all block files. Therefore, regardless of slight improvement compared with Bitcoin, its TPS is still unable to achieve remarkable improvement and remains between 20 and 30.

3.3.3 Methods of CXCTPS improvement

CXC improves TPS significantly via following methods:

- Memory-driven to database-driven for structured storage and rapid retrieval.

The CXC core wallet is structurally designed as a local dual database, which divides the original node single database into block database and wallet transaction database, of which, the block database still stores all the block data, and the wallet transaction database stores the transaction and transaction information related to the local wallet address. Any transaction metadata larger than 256 bytes will not be stored in the wallet transaction database, but the wallet transaction database contains a pointer indicating the data position of the blockchain. When a wallet transaction is carried out, transactions satisfying preset standards can be quickly located by directly searching the block database with the pointer or index, instead of a complete wallet scanning, which significantly saves inquiry time and accelerates transaction.

- A unique ChainDB which improves large-capacity chaining data under the premise of guaranteeing performance;

ChainDB allows blockchain being used as a general additional database. Blockchain offers time stamp, notarization and invariance. CXC contains any number of ChainDB, and data issued by each DB is stored on each node, of which, each data module is open to all users for writing, or supports writing from a specific address. If one node subscribes ChainDB, it will establish a

contents index for this data module in the wallet transaction database to facilitate high-efficient retrieval in various means.

- Adoption of optimized POA consensus algorithm as the auxiliary means to improve TPS

POA consensus algorithm reduces consumption of resources and accelerates the process of reaching network consensus. Shorter blocking time and larger block capacity helps to improve TPS.

In general, through the three methods mentioned above (also there are other methods such as automatic collection of unspent output to improve retrieval efficiency), CXC can achieve higher TPS under the premises of safety and reliability by adopting POA mechanism and supporting large-capacity chaining data. On a single node of terminal equipment, TPS can reach 2,500 transactions per second, or tens of thousands to millions of transactions can be processed per second when the high-end equipment is connected to a LAN or partially centralized (such as super node). But in WAN, affected by bandwidth and other network factors, the actual TPS may drop.

3.3.4 Throughput

After application of above means to improve TPS, we may have a check of the TPS in CXC. Following is the throughput of transactions processed on each single node. The average transaction volume per second includes API expenditure, connection, signature, mining and validation of transaction blocks. Via jmeter, send commands of multiple intercurrent http requests are executed with computer specification of Intel Core i7, 4 cores, 8 GB RAM, 2 TB 7200 RPM SATA, Ubuntu 14).

Transaction Times	TPS
100	2900
1000	2766.6
10000	2536.11
100000	2308.6

1000000	2181.5

3.4 51% attack of computing power

51% attack refers to an attack on blockchain after controlling 51% of the network's computing power to recalculate blocks which have confirmed, and branch the blockchain and make profits.

For 51% attack of computing power, CXC reduces its possibility again and significantly via POA (cross consensus mechanism) which confirms the consensus of all nodes participating in consensus in batches, and when a 51% attack of computing force is initiated, as the cross consensus mechanism is random to consensus node, the 51% attack of consensus power fails to be remained after attacking. Therefore, through several times of confirmation after attacking, it can be rolled back to the original blockchain.

4. Transaction model

The structural model of CXC transaction data adopts UTXO model. UTXO is a data structure containing transaction data and executing code. It can be informally understood as the encrypted digital currency an address has received but not spent.

For following reasons UTXO model is adopted:

- Bitcoin network has been safely used for years, which supports that UTXO model is able to stand the test, and a financial transaction model different from account.
- It has natural protection against double spend attack and replay attack.
- It is easy to be scaled, highly parallel and strongly dormant.

But UTXO model has obvious shortcomings as well. In any transaction there will be multiple inputs and outputs, which have to be browsed over for inquiry of balance or transaction, significantly reducing performance. To this end, CXC makes improvements and puts forward solutions:

- Unique cross-chain molecular storage structure.

CXC adopts a cross-chain molecular structure for storage which consists of local two main databases and divides the single database of original node into block database and wallet transaction database, of which, the block database still stores all the block data, and the wallet transaction database stores the transaction and transaction information related to the local wallet address. Any transaction metadata larger than 256 bytes will not be stored in the wallet transaction database, but the wallet contains a pointer indicating the data position of the blockchain. When a wallet transaction is carried out, transactions satisfying preset standards can be quickly located by directly searching the block database with the pointer or index, instead of a complete wallet scanning, which significantly saves inquiry time and accelerates transaction.

- Collection and transmission.

During operation of CXC, the user can automatically have all unspent outputs (UTXO) in the same address combined into an unspent output, in order to improve wallet performance, and reduce the consumption of native assets during transaction.

5. Smart contract and ChainDB

5.1 ChainDB

To improve the performance of blockchain, the concept of CXC ChainDB is put forward and designed as the first example in the industry, which has account transactions separated from data to maintain the TPS while guaranteeing large-capacity chaining data. As a general data storage function on CXC, ChinaDB provides advanced abstract and API, and can be used on chain to realize three different types of database:

- NoSQL key value database;
- Identity driven database, enquired and classified according to specific

sender and receiver;

- Stationary sequence model database, with data changing stationarily to eliminate long-term trend and differencing, and for item sorting.

ChainDB allows blockchain being used as a general additional database. Blockchain offers time stamp, notarization and invariance. Any number of DB can be created and data issued by each DB is stored on each node, of which, each is open to all users for writing, or supports writing from a specific address. If one node subscribes DB, it will index its contents to facilitate high-efficient retrieval in various means. Otherwise, it has no need to pay any computing power. Each data in each DB is a sequenced item list.

Each data has following characteristics:

- One or more items which have been digitally signed by senditemers.
- One or more key value with length between 0-256 bytes, which can be used for indexing.
- data, which can store several M of data.
- JSON object supporting storage structuring, easily to write and read.

If one node subscribes this data module, it can perform index and inquiry via any of following methods:

- Indexing of key value.
- Indexing of senditemers, namely, the creator.
- Indexing of txid, blockindex, blockhash.

Bridging different times, ChainDB simplifies and facilitates the interaction between chains, and solves many problems such as token-token transaction on chain, cross-chain assets exchange, and scenarized smart contract which can’t be solved through technical design.

5.2 Scenarized smart contract based on ChainDB

5.2.1 Shortcomings of traditional smart contract

The earliest definition about smart contract can be dated back to 1994 by NickSzabo: “Smart contract is a computable transaction protocol which executes contract terms”. A smart contract on blockchain is a section of code

stored therein, which is triggered by blockchain transactions, reads and writes data from or into the database in this blockchain. With Ethereum as an example, it allows anyone to create a smart contract on blockchain. Though it is Turing sound, but in practice, there are many shortcomings which are inevitable:

- All nodes have to execute computation of smart contract, whether interested or not, which slows down the operation significantly.
- The performance of virtual machine for smart contract is far below the performance of given computer architecture running code; for safety, smart contract shall be run in separated sandboxes, and a new virtual machine shall be started when a smart contract is called out, which affects the operating system of smart contract significantly.
- Low concurrent performance. Smart contract is linked to a database, which means that node on the blockchain has no way to know which database shall be called out for the smart contract. Therefore, each transaction may cause interference to others, unable to ensure that multiple progresses can be carried out at the same time and according to their sequence. Though zoning technology and progress calling are applied in blockchain, it results a smart contract modifying its status records in a smart manner, which means that contracts can't be called out for each other. This totally deviates from the design goal.
- The code assigned to a smart contract can't be varied and upgraded, a horrible thing to modern software developers.
- High delay. Due to the nature of general computation, contents sent to the smart contract can't update the status of blockchains until their final sequence is determined, which will result in delay (until transaction confirmed in block) and any possible reversion (branching in chain).
- Unable to stop incorrect transactions from being added into the blockchain.
- External data to be accessed by traditional smart contract shall be provided by a centralized server, which goes against the decentralized

characteristic of blockchain. Data stored in external centralized server can't guarantee data safety, and may result the whole blockchain meaningless.

5.2.2 Scenarized smart contract

CXC develops a technology named smart engine. Differing from smart contract, smart engine never requires all nodes participating in computation and validation of transaction. Only subscribers are required to participate in those processes.

Smart engine can be written in JavaScript and used in Chrome and Node.js. It returns a Boolean to indicate if the transaction is acceptable. If not, the error causes will be provided.

To achieve a smart contract which is completely decentralized, CXC puts forward the solution of circulating via ChainDB. All external data triggering the smart transaction will be stored to ChainDB of CXC. When the smart engine finds any data change, where conditions are complied with, it will execute transaction or other operations to avoid any slowdown of operation speed as for each operation, all footnotes have to be checked for compliance with execution conditions, and to maintain the design concept of decentralization.

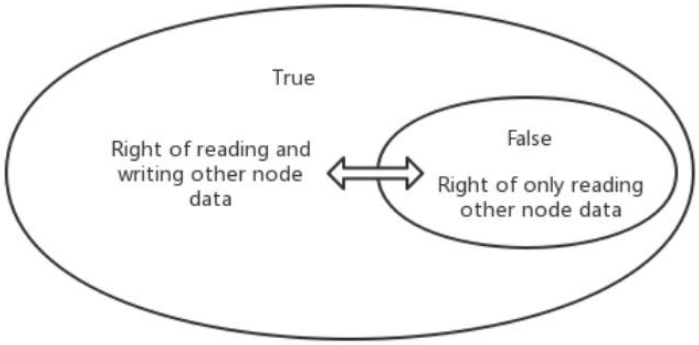
In addition, if an error is found in the code of smart engine, it can be replaced with a new version. Against the risk of infinite loop, CXC puts forward following solutions:

- The creator of smart engine shall perform a validation before forwarding the broadcasting. If it involves in infinite loop, then creation is not supported.
- All nodes subscribing smart engine shall have it validated. If it involves infinite loop, then subscription is not supported.

For large scenarized applications, user may depend on external codes and use any general programming language to write the external application code, namely, key data is stored in ChainDB, smart engine validates transactions, and external codes run on business logics level.

5.3 Safety and privacy mechanism of ChainDB

When creating a DB, we may set the open parameter as true or false, which leads to two different results:



Venn Diagram of Rights of ChainDB

As shown in the Venn Diagram of Rights of ChainDB, if open is false, then all other nodes have the right of reading data only; if it is true, all other nodes are allowed to read and write data.

Through the design above, any primary data stored in the blockchain is visible to each node connected to the chain. In many cases, the loss of privacy is a major issue. Then how should be solve it? CXC gives you following answers:

- Data is written in an encrypted manner.
- Passwords to read data are only available to subset of participants, and by combination of symmetric cryptography and asymmetric cryptography, we can finish this task high efficiently.

  The solution can be created via 3 data modules;
- pubkeys. is used by participants as the public key allocated to them according to RSA public key program.
- items. issues data, and each data is encrypted by symmetric AES passwords.
- access. offers data access rights . For each participant who shall have access to the data, a data is created under the data module which contains

the data passwords encrypted with the participant's public key.

6. Scalability solution

6.1Shortcomings of existing blockchain data storage

General blockchain networks, such as BTC and Ethereum, support incidental storage of HASH and extremely short data through transactions, but are unable to provide a storage for massive and large data on chain.

Against chaining data storage and retrieval, CXC designs ChainDB to build each section of data into the transaction. Each transaction on the blockchain shall be digitally signed by one or more parties, and copied to each node for sorting and time stamping by the consistency algorithm of chain, and stored permanently in a tamper-proof manner. At present, ChainDB has reached the data capacity of 16M for a single data transaction, which can satisfy the small and medium-sized projects' demands on data storage. However, the huge volume makes large data not suitable for chaining storage. Therefore, CXC develops an out-of-chain data storage function similar to N*Raid5 for chaining interaction of large data.

6.2 CXC out-of-chain data storage—out-of-chain fragmented storage similar to N*Raid5

The development of blockchain applications realizes the scattered out-of-chain data transmission. A general option is the adoption of existing peer-to-peer file sharing platform, such as IPFS, for combined use with blockchain. But, IPFS has shortcomings as follows that it can be combined with blockchain at a high efficiency and conveniently;

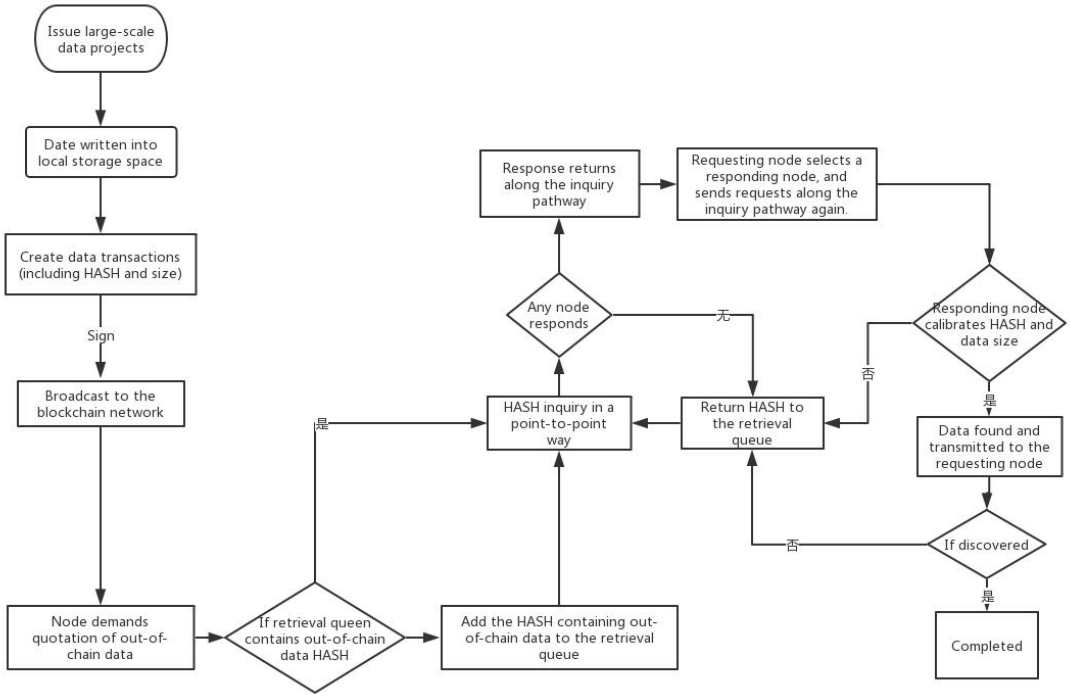
- Each participate shall install, maintain and update three totally independent software (blockchain node, IPFS node and middleware).

Each software has the data stored to different positions.

- There will be two independent point-to-point networks, each with its configuration, network port, identity system and permission.
- IPFS is closely combined with blockchain, which makes the middleware more and more complicated, and faces the risk of centralization.

Against the shortcomings and risks arising from the combination of IPFS and blockchain, CXC achieves interaction and synchronization with out-of-chain data, and avoid any possible centralization risk via following convenient and high-efficient ways:

The issue node writes new data into the its local storage space, and divides large-scale projects into blocks to automatically structure and issue out-of-chain data transactions which will be signed and broadcasted to the network, transmitted between modes and enter the blockchain via general ways. When the data demanding nodes requires quotation of out-of-chain data, it will add the HASH request of this data to the retrieval queue, and use it as a backstage progress. If the HASH is found in the retrieval queue of the node, then it will be enquired and sent to the network to find blocks with the HASH identifier. The inquiry is transmitted to other nodes in the network in a point-to-point manner. Any node with data can respond to it and the response will be relayed to user along the pathway of inquiry. Where no node responds, the HASH inquiry will be returned to the queue for another try later. If no node responds for a long term, the demanding node will sent the request to the network again. Where a node receives the request, it will validate the data size and HASH value according to request, send corresponding data, and as the data is retrieved, it will be written into local storage space by the receiving node for retrieval according to API. Where the request is not received, or fails to match with the required HASH or size, it will return to the queue for retrieval from other sources.



Flow Chart of CXC Out-of-chain Application Data Storage Similar to N*Raid5

In a shortly delayed network, offline data of smaller volume will be transmitted at the moment of quoting the transaction. For high-load applications, CXC supports networking of more than 1,200 out-of-chain projects or retrieval of 30MB out-of-chain data per second, and the normal transmission of off-chain data of 1000MB maximally, with little affect on the CXC network efficiency.

ChainDB can be set to store general DB, application DB and hybrid DB. The latter two can be used to store information such as HASH, publisher and index of out-of-chain data block, realize rapid retrieval and distribute out-of-chain data block.

For each block, the on-the-chain data shall be limited to 16M. When it exceeds 16M, it can be stored as out-of-chain data (a single out-of-chain data block exceeding 1000M is not suggested).

7. Assets issuing

In addition to native tokens, CXC also supports users to issue their own assets. When creating assets, necessary parameters for the assets have to be designated:

Once created successfully, the new assets will be sent to the designated address, of which, if the open parameter is true, it means that increase in later stage is permitted.

8. General CLI commands

Command	Parameter	Description
help		Return the list of available API command.
stop		Close the node on this blockchain
showmem		Return information about the memory pool
addnode	1.ip(:port) 2.command	1. For example: 127.0.0.1:7318 2.add|remove|onetry
shownode	1.true|false 2.(ip(:port))	1. true=return all information about the use of addnode, false=return the list of addnode 2. node containing or omitting ip(:port)
shownet		Return information about network IP and port in connection
showpeer		Return information connected to other nodes

showblock	1.hash|height 2.(0|1|2|3)	Check the designated block information of hash|height 1. hash|height 2. (0|1|2|3|4) 0: block in primary hexadecimal format. 1 (Default): abstract of blocks containing miner address and txids 2|3: contain more information of each transaction and its primitive hexadecimal format 4: contain the complete description of each transaction
showblocks	1.blocks 2.(true|false)	Return information of designated block 1:string|array|object string:heigt|hash|height range array:jason array object:{/"starttime/":1520299771,/"endtime/":1520299771} 2:true|false
showblockhash	height	Return the HASH value of given height block

addnewaddr		Return the new address added to wallet by private key
addmultiaddr	1. Nrequired 2. ["PUBKEY", ...]	Create multiple addresses and add to the wallet.
setupmulti	1. Nrequired 2. ["PUBKEY", ...]	Set up multiple addresses but not add to the wallet.
setupkeypairs	(count)	Set up one or more pairs of public/private keys, 1 in default
showaddrs		Return the detailed address information to the wallet of the node.
dumpprivkey	address	Private key dumping address.

importprivkey	1. privkey(s) (label)	Add import private key (or number of groups) and generate related addresses.
importaddr	address(es)	Add the address to wallet (without private key), create one or more view address.
backupwallet	filename	Back up wallet.dat
dumpwallet	filename	Dump all private keys in wallet to text file.
importwallet	filename	Import wallet.
encryptwallet	password	First encryption of node wallet, with password as the unlocking password. Once encryption is done, the private key in the wallet can't be directly retrieved from wallet.dat. It will stop and restart.
changepass	1.old-password 2.new-password	Change wallet password.
walletpass	1. password timeout	Unlock the node wallet with password, valid within timeout.
showassets	1. (assets=*) 2. (false|true) 3. (count=MAX) 4. (start=-count)	Return information of assets issued on the chain.
showbal	1.(minconf) 2.(false|true)	1. Default: 1 2. Default: false: show local address, true: contains monitoring address
showaddrbals	1.address 2.(minconf=1) 3.(false|true)	Return the list of all assets balance in the address node wallet (include native currency) 1.Address 2.Minimal confirmation false excludes locking
showallbals	1.(addresses=*) 2.(assets=*) 3.(minconf=1) 4.(false|true) 5.(false|true)	Return the list of balance in the node wallet designated by addresses (including native currency). 1.Address 2.Assets 3.Minimal confirmation 4.false excludes monitoring address false excludes locking

showaddrdeal	1.address 2.txid	Output information of transactions related to address txid
showaddrdeals	1. "address" 2. (count) 3. (skip) 4.(false|true)	1.Address 2.Reutrn number of transactions 3.Skip 4.Detail information of true
showwalletdeal	1.txid 2.(false|true) 3.(false|true)	Provide information related to the transactions in the txid node wallet. 1. txid 2. false excludes monitoring address 3.false excludes detailed information
showwalletdeals	1.(count=10) 2.(skip=0) 3.(false|true)	Show information related to the last transactions in the node wallet of this count 1.Count 2.Skip 3. false excludes monitoring address
send	1. address 2. amount 3. (comment) (comment-to)	Send one or more assets (native assets allowed) to the address, and return txid. 1. address 2. {"asset":qty, ...} object amount, asset name, ref or sell txid, each object qty is the quantity of assets sent. 3.additional information Attribution of additional information
sendfrom	1. from-address 2. to-address 3. amount 4. (comment) 4. (comment-to)	send via from-address
sendasset	1. address 2. assetID 3. Qty 4. (Fee) 5. (comment) (comment-to)	Send asset of qty to address, and return TxID. 1. address 2. assetID: name: ref or issue TxID 3.Qty 4.Fee 5.additional information Attribution of additional information
sendassetfrom	1.from-address 2.to-address 3.asset qty 4.(Fee) 5.(comment) 6.(comment-to)	Similar sendasset via from-address

senddata	1. address 2. amount|object data-hex|object	Similar send, add data transaction output 1.address 2.native token |{"asset-id" : qty} data-hex|{"for":datamod,"key":"...","data":"..."}
senddatafrom	1. from-address 2. to-address 3. amount|boject datahex|object	Similar senddata via from-address

For details, please refer to the file for developers.