PalletOne Yellow Paper

V1.0 beta

May 2018

Contents

Abstract

PalletOne aims to be the IP protocol for the blockchains, provide the operating environment for multilingual smart contracts via the abstract interfaces of blockchains and totally decouple the underlying chain and smart contract to build a general and Turing-complete blockchain platform for smart contracts across chains. The overall structure of PalletOne is shown in Figure 1.1.

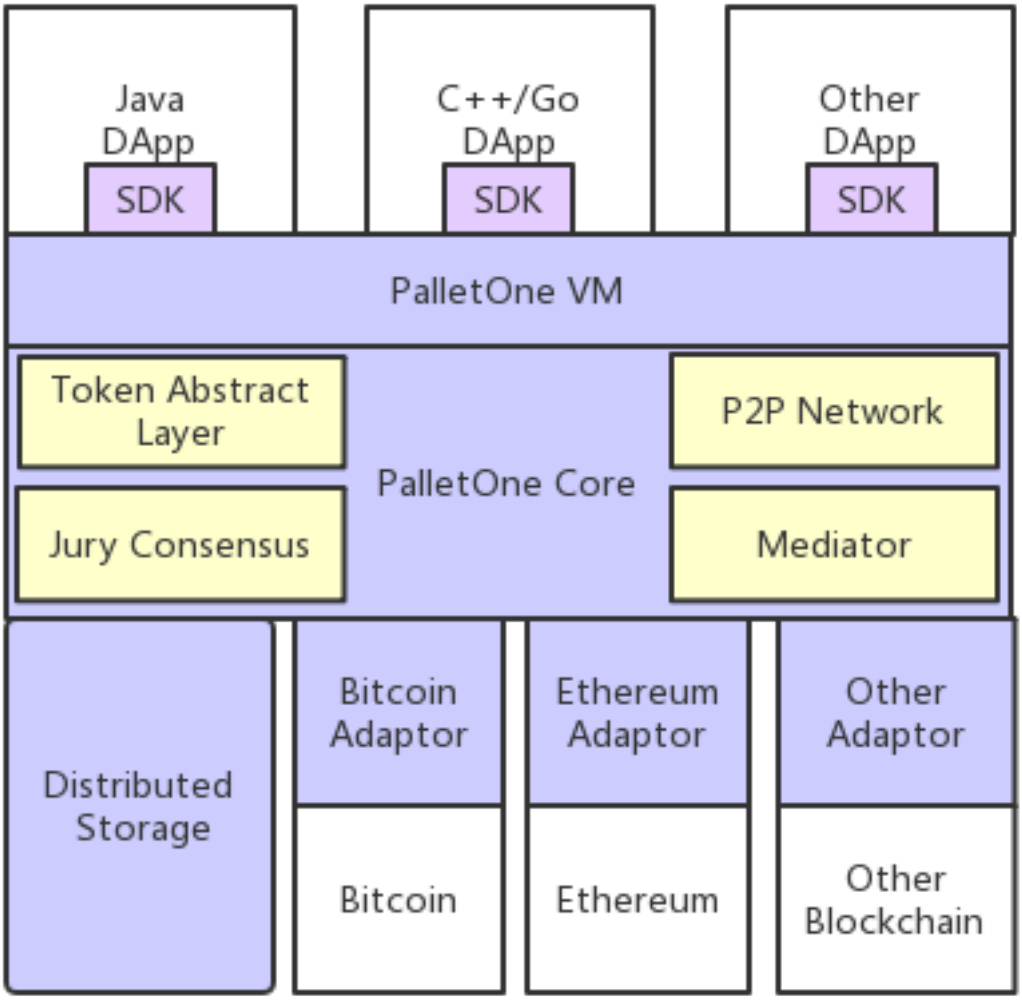

Figure 1.1 Overall structure of PalletOne

PalletOne features multi-chain, multi-language, high capacity, abstract tokens and complete ecology.

1. Multi-chain

PalletOne, as a cross-chain platform, is not bound to any specific blockchain, but aims to achieve the value exchange among chains by building adaptors for corresponding chains based on the abstract general interface.

2. Multi-language

2. PalletOne supports Turing-complete smart contracts. Unlike any other blockchain with smart contract that adopts independent programming language, PalletOne adopts the world's most popular languages (e.g. Java, C/C++, JS) for the development of smart contracts. PalletOne virtual machine provides a sandbox with safe host for various languages in contract execution, which decreases development

difficulty for smart contract developers, thereby enabling the developers to use their proficient language.

3. High Capacity

Different from the serial block generation of general blockchains, the Jury Consensus and DAG Distributed Storage of PalletOne can be operated concurrently on calculation and data storage, thereby achieving the high Capacity.

4. Token Abstract Layer

Unlike Ethereum that users needs to writetheir own ERC20 o ERC721 smart contract codePalletOne comprises Token Abstract Layer in the core, which improves the efficiency and simplicity of token issue and circulation. On PalletOne, users only need to input a few parameters (without editing any contract) to issue their own tokens, which avoids attackscaused by contract bugs.

5. Stable and Healthy Ecosystem

The excellent token economy design of PalletOne lays a solid foundation for the formation of its stable ecosystem. Since PalletOne Application Store adopts the model similar to AppStore, developers, consumers and miners can all be benefited from the PalletOne platform.

The overall network of PalletOne is shown in Figure 1.2.

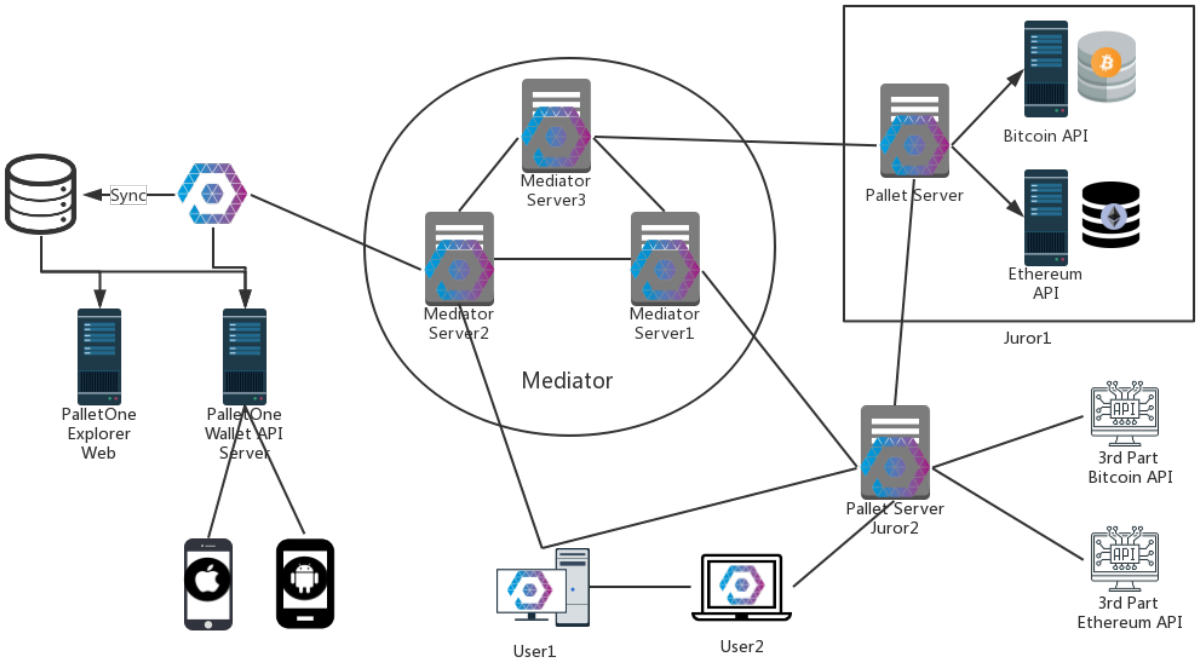

Figure 1.2 Overall network of PalletOne

The network in Figure 1.2 is mainly composed of the following parts:

1. Mediator

2. Mediator is the core of the whole network and comprises 21 independent nodes. Mediators, selected by users via DPOS algorithm, are responsible for token circulation, contract deployment, jury creation and DAG unit witness, and are connected in pairs for rapid communication.

2. Jury

Jury is an execution node for smart contract without restriction in number. A node may join several juries to reach consensus for contract execution. As cross-chain operation is necessary for Jury, the query and operation should be made for the other chains under the following modes:

- The juror (e.g. Juror1 as shown below) builds a full-node server for the other chains, and PalletOne directly connects to the self-built API service where cross-chain operation is needed.

This mode features fast and secure operation and high cost in deployment.

- API of the third-party chain, such as blockchain.info, should be invoked. This mode features low cost, possible high network delay and external independence that leads to vulnerability.

3. PalletOne Explorer provides the function on query of all historic ledgers and current network status. Users can view contracts, tokens, network, statistical analysis and statements via PalletOne Explorer.

3. Full-node wallet

Similar to Bitcoin Core, users can set up their own full node to locally synchronize ledger data and then conduct query and trading.

4. Mobile wallet

Like Imtoken Wallet, PalletOne will provide mobile wallet on Android and iOS platforms. Users can complete token and contract management with APP. In addition, for the ease of cross-chain operation, PalletOne mobile wallet will support Bitcoin and Ether management.

The overall system of PalletOne is shown in Figure 1.3.

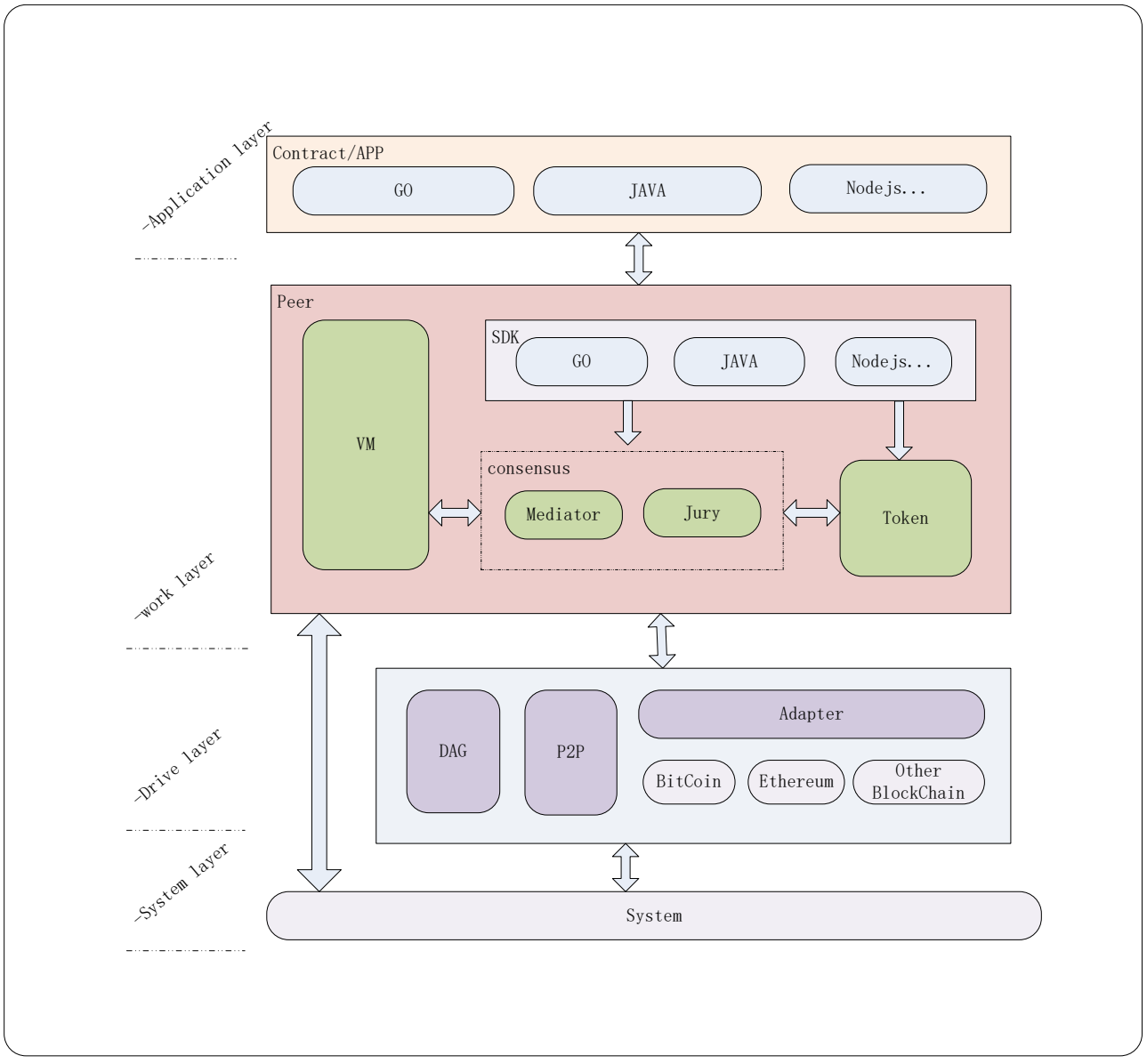

Figure 1.3 PalletOne system

As shown in Figure 1.3, PalletOne system is composed of four layers:

1. System: A specific system provides fundamental service.

2. Drive: It mainly contains DAG Storage, P2P network and adapter modules for functions of different blockchains and aims to satisfy the support to the business layer.

3. Business: It is mainly composed of PalletOne Mediator and Jury Consensus Mechanism modules, supports virtual machine of smart contract and token economy module and provides corresponding SDK module for the development with different programming languages.

4. Application: It comprises the SDK-based APP and smart contract implementation to satisfy the development of different businesses.

PalletOne Core

PalletOne Core, the principal component of PalleOne, as the name suggests, functions as the bridge of communications between the various modules of PalletOne. The PalletOne Core consists of five parts, namely, a Mediator module, a Jury Consensus module, a token abstract layer, interaction interfaces with other modules and P2P network module.

With user friendliness and use easiness in mind, PalletOne Core encapsulates within many application scenarios for users at the top layer, such as abstraction of the token definition and universal contract. At the same time, it also wraps the exchange interfaces of all the modules so as to achieve the decoupling between layers of users, distributed storage and blockchain adaptors. The jury consensus and mediator modules together constitute the security and accountability mechanism of PalletOne to ensure the normal operation of PalletOne.

Mediator

Mediator is responsible for the security of the PalletOne network. Mediator, similar to a traditional block chain, is actually a trust machine, As a result, the Mediator needs to ensure that all decisions in PalletOne are correct. The mediator shall be elected through the Delegated Proofs of Stake (DPoS) mechanism, and within it the Byzantine Fault Tolerance (BFT) shall be employed by nodes to reach the consensus. In order to prevent Mediator from becoming the bottleneck of PalletOne, most of the security work in PalletOne shall be completed solely by Jury and without invoking the Mediator. The following are the main tasks of Mediator:

1. To hold the deposits of jurors;

2. To deploy smart contract templates;

3. To organize a jury with randomly selected jurors;

4. To provide a signature for multi-signed wallets;

5. To testify DAG units in the distributed storage.

Deposit Managerment

The deposits of the pool of candidates for Mediator and Jury shall be managed in accordance with the provisions of a PalletOne Deposit Contract. Unlike an ordinary smart contract, the deposit contract is fixed in the PalletOne Core and shall be executed by Mediator. It has the functions to deliver, refund, and confiscate a deposit.

In order to prevent the mediator and jury from taking evil actions, users who are willing to become Mediator and a member of Jury must first deliver a certain number of PalletOne Tokens as deposit and thus shall be approved to enter the pool of candidates.

Deposit Delivery

In PalletOne, every user can pay a certain number of PalletOne Tokens at his/her will to enter the pool of Mediator and Jury candidates.

Deposit Refund

```
//handle witness pay
void deposit_witness_pay(const witness_object& wit, token_type amount)
```

```
//handle cashback rewards
void deposit_cashback(const account_object& acct, token_type amount,
bool require_vesting = true);
```

In the exit of the pool of Mediator and Jury candidates, PalletOne Smart Contract Module will execute to return the deposit to the original registered account.

Deposit Forfeiture

For any user makes malpractice as a mediator or juror, the PalletOne Smart Contract Module will execute to confiscate his/her deposit.

When PalletOne is launched online, the Foundation can set up all Mediator nodes as the stating nodes

```
void forfeiture_deposit(const witness_object& wit, token_type amount)
```

and the other users and organizations can replace such staring nodes by voting upon deposit payment. The formal nodes will be operated by the community.

Deployment of Smart Contract Template

The smart contracts are executed by the Jury. However, before the creation of any smart contract, the developers need to deploy the contract template into PalletOne network and Mediator will verify and recognize the template.

The detailed procedures for the development of smart contract template are specified in the chapter “Life Cycle of Smart Contract” hereinafter.

Random Selection of Jurors

When any user starts contract creation, Mediator will randomly select jurors to form the jury as the parameters of contract template and then hand over the contract instantiation and execution to the jury. The procedures are summarized as follows:

1) A user sends a request of contract creation in Mediator P2P network;

2) Mediator super node authenticates the data at a certain time;

3) The current Mediator rotating node randomly selects a algorithm to list jurors and sends the contract execution results and signatures to the P2P network;All Mediator nodes authenticate the jurors and store the data in DAG. The flows of jurors random selection are specified in the chapter “Algorithm for Random Selection of Jurors”.

Signature Provision for Multi-Signed Wallets

Since Mediator needs more deposits and features high online rate and stability, stable signatures can be provided for multi-signature wallets of cross chains

As for the multi-signature wallet of Bitcoin, if a 7/12 multi-signed wallet is created and each User A and User B holds a key, the rest 10 keys can be held by the 10 Mediators winning the highest vote. If the Jury makes a consensus on multi-signature, Mediator directly trusts the consensus results of the Jury

and provides his/her own signature at the right place without the need for contract execution.

Witness of DAG Unit

In PalletOne, Mediator needs to complete the witness of DAG unit. The generated witness unit is composed of trasaction confirmation unit, contract creation & authentication unit and contract invocation & authentication unit. The flows of Mediator to witness DAG unit are shown in Figure 2.1.

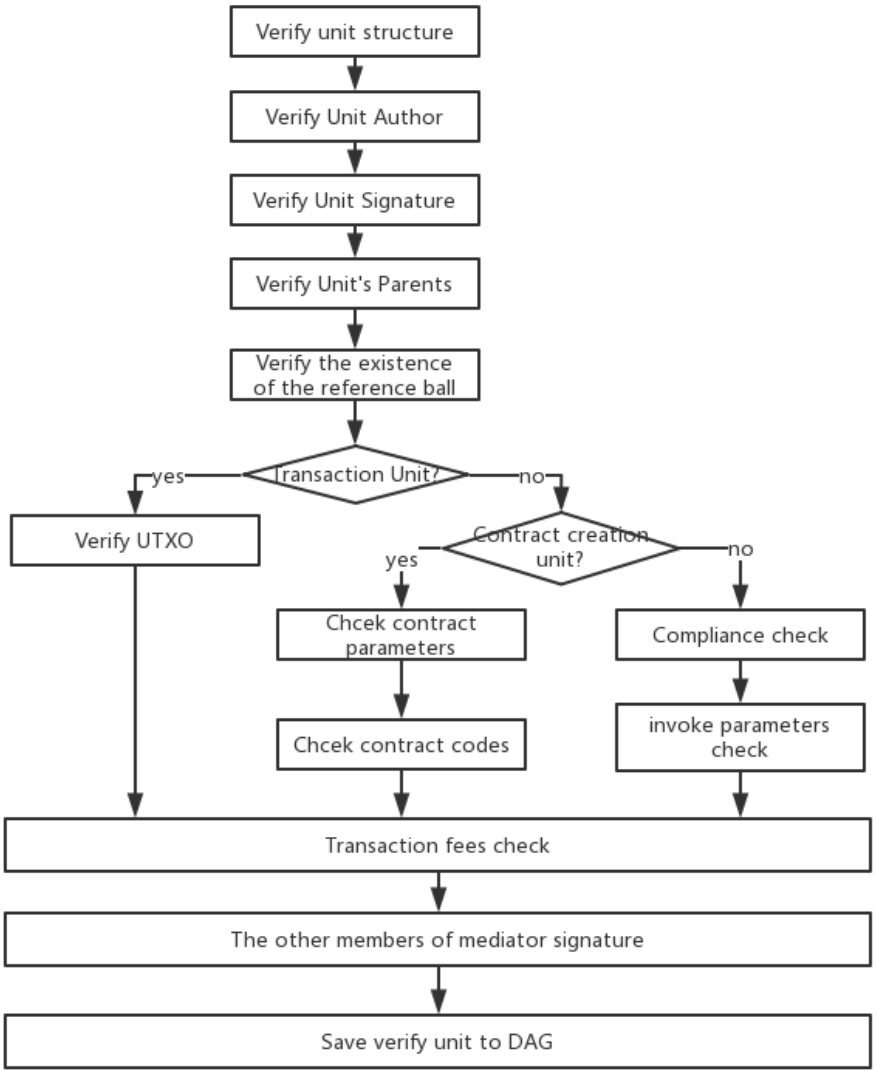

Figure 2.1 Mediator's witness of DAG unit

Jury

Jury is a basic unit to maintain the security and integrity of PalletOne. In specific, Jury is entrusted to operate smart contracts and multi-signature accounts. In order to achievesecurity and decentralization, Jury is composed of a number of participants ("jurors") as designed. Each juror pays a deposit for security assurance. Jury adopts VRF+BFT consensus algorithm to achieve consensus. The random generation and internal consensus algorithm of Jury are detailed in the chapter "Jury Consensus Algorithm".

Locked & Un-locked Mode of Jury

Jury is formed in two forms:

- Locked mode

- Non-locked mode of Jury

The following describes the implementation of the two modes.

Locked mode of Jury

A jury is created at the same time as a contract is created and bounded to the contract. The members of the jury are fixed, meaning that such jurors will execute all the subsequent invocations of the contract and will not be allowed to be replaced. This mode will require most jurors to be online for a long time and only allow a small number of them to stay offline. For example, in a jury of four jurors, only one juror is allowed to go offline. It will apply to contracts with a shorter lifecycle. As for contracts that are to be executed for a few months or more, it cannot be guaranteed that the members of a locked-mode jury can always be online for a long term to carry out what a juror should do.

Non-locked mode of Jury

The non-locked mode of Jury means that a pool of jurors more than actually need ones is selected when the contract is created, and for every invocation of the contract, the needed number of jurors shall be randomly selected from the pool, based on some random selection algorithm, to execute the contract. The non-locked mode will be applicable for long-running contracts that do not require multi-signature.

Take the example of a contract of "purchasing game currency with BTC and ETH". This contract does not require multi-signature, and is long-running, and thus suits to the non-locked mode. Provided that 10 jurors are required to reach an consensus for the contract when it is created, the mediator will at first select 50 jurors, who are willing to execute the contract from the list of all candidate jurors, to form a jury pool, and then randomly select 10 ones from the pool to form a jury to execute the invocation for this time. And for each invocation of the contract, the mediator will randomly select 10 jurors to execute the contract from the jury pool.

The more nodes in the jury pool, the more jurors are elected at one time, and the slower to reachconsensus on contract execution, but the more secure it is, and the less likely it is to be attacked. Therefore, smart contract developers shall weight between the efficiency and security in reaching consensus and find a balance according to specific business scenarios.

Token abstraction layer

In PalletOne, a token shall be issued in accordance with the principles similar to issue a colored coin. When a token is issued there, it will be at the same layer with the PalletOne Token, so the user can operate them as he/she do with the PalletOne Token. And in the distributed data storage, we can see that the data of the token is isolated from the one of the status of its contract, which is different from Ethereum where the former is treated same with the latter.

Token Types

PalletOne will have the following token abstraction modes built in the early days:

Homogeneity tokens

Similar to the tokens issued by ERC 20 in Ethereum, users only need to designate the total volume, accuracy, token name and abbreviation of the token in issue. PalletOne creates and issues the tokens at one time without additional issue.

1. Non-homogeneity token

It is often said tokens are homogenous, meaning that tokens held by us are the same. However, there are a lot of non-homogenous tokens in the real world. For instance, if the artworks (e.g. paintings) become tokens, each token represents a unique work of art and cannot be combined or divided. ERC721 in Ethereum defines the non-homogeneity token. PalletOne supports non-homogeneity token. Non-homogeneity token is not created at one time but based on designated address, which may be user's address or contract address. If it is created on a user's address, only the user can create the non-homogeneity token; if it is created on a contract address, the token can only be created by the logic in the smart contract.

Non-homogeneity token contains the following data:

Token name, unique token code, additive attribute, creator address and time of creation

The data above cannot be altered after creation.

In addition to the two common models mentioned above, more token models will be introduced in the future to adapt more application scenarios:

2. Semi-homogeneity token

Semi-homogeneity token means that tokens are different and different tokens may not be unique. For example, if token is applied to the stamp collecting, each stamp is expressed by a token, but the number of stamps of the same kind is at least one. Only the same kind of stamps can be integrated and divided. Like the homogeneity token, semi-homogeneity token requires designated address instead of one-time creation.

Semi-homogeneity token contains the following data:

Token name, unique token code, number of tokens, additive attribute, creator address and time of creation

3. Mining token

Similar to the economic model of Bitcoin, users do not premine or do not fully premine in token issue, and tokens will be gradually issued along with the time and block generation rate.

4. Fixed-denomination token

Similar to the paper currencies in real life, users can define tokens with denominations like 1, 2, 5, 10, 20, 50 and 100, and the tokens cannot be divided in use.

Token Operation

1. Token initialization

PalletOne provides the templates of homogeneity token and non-homogeneity token. Users can fill in parameters via the templates to create their own tokens. Users need to provide the parameters as specified in Table 2.1 in token initialization.

Table 2.1 Parameters for token initialization

Parameter	Description
initTotalSupply	Total amount of issued tokens in initialization
decimals	Accuracy (divided to several decimal places)
name	Full name of token (within 500 characters)
symbol	Abbreviated symbol (within 5 characters)

tokenType	Type of token: Homogeneity/Non-homogeneity
issueAddress	Which address has the right to issue the token? Personal account address or contract address. If it is null, reissue is prohibited.
extProperties	Extended property. Users can define some properties (e.g. JSON format).

2. Issue of tokens

In defining tokens, users can designate the address, which can be personal wallet address, multi-signed wallet address or a contract address. Once the address is designated, tokens can only be issued from the said address. For the fully premined homogeneity tokens, if the issue is prohibited after creation, no issue address is provided. All the tokens are under the issue address after completion of the issue.

```
function issueToken(string _uname,json _properties,uint256 _value)
returns (bool success);
```

For the issue of non-homogeneity tokens, the unique name of the newly issued tokens and property of JSON format can be designated (number: 1). The homogeneity tokens, in general, will not be issued after initialization; if the issue is allowed, it is only necessary to designate the number of issuance instead of the unique name and JSOPN property.

3. Transfer of tokens

```
function transfer(UTXO[] _from, (address _to, uint256 _value)[]) returns
(bool success);
```

PalletOne adopts the operation mode similar to that of Bitcoin with UTXO model at the substructure to record users' balance, and supports multiple inputs & outputs and P2PKH & P2SH for operations (i.e. single-signed transfer and multi-signed transfer).

Limit of transfer of the authorized address. The authorized address can repeatedly invoke transfer function to replace the original address for transfer (total amount not exceeding _value).

4. Authorization of token

The token authorization in PalletOne is similar to the approval method in ERC20/721 – an address is allowed to withdraw a given amount of tokens from the authorizer.

```
function approve(address _spender, uint256 _value) returns (bool
success);
```

Since PalletOne adopts UTXO model, the essence of approve is to transfer a certain number of tokens to 12/ multi-signed address and both the authorizer and authorized roles can use the tokens under the multi-signed address.

5. Token destruction

A special address is defined in PalletOne as the destruction address. To destroy a token, the user involved only needs to transfer such token to the destruction address. The destroyed tokens cannot be recovered, and the total number of issued tokens will be reduced accordingly.

```
function destroy(uint256 _value) returns (bool success);
```

6. Transfer of the right to issue tokens

Any user who holds the right to issue tokens can transfer the right to any other user or contract, and

```
function transferIssue(string _address)  returns (bool success);
```

such right will be lost to the former user after transfer.

Interactions with Modules

Interaction with Smart Contract Virtual Machine

In PalletOne Core, the following operating interfaces of virtual machine have been defined, and it only needs to implement such interfaces in the virtual machine. For the core, it is unnecessary to care about which method is taken to realize the virtual machine and achieve the loose coupling with Docker for the further support on more virtual machines.

Deploy virtual machine

```
string Deploy(string templateId, string config,byte[] program)
```

1. Start virtual machine

```
ContractResult Start(string vmId, string instanceId, string[]
parameters)

struct ContractResult
{
 TokenPayment[] TokenPaymentSet,
 KeyValue[] ReadSet,
 KeyValue[] WriteSet,
 ChainTrans[] InterchainSet
}
```

- Transfer operations related to tokens on TokenPaymentSet and PalletOne. What data have ReadSet read?
- What data have WriteSet written?
- What transactions have been initiated by InterchainSet if it is a cross-chain contract?

2. Call the functions in virtual machine

```
ContractResult Invoke(string vmInstanceId,string function,string[]
parameters)
```

3. Shut down virtual machine

```
Stop(string instanceId)
```

4. Destroy virtual machine

```
Destroy(string vmId)
```

The core provides cross-chain access interfaces for virtual machine operation, and the invoking of such interfaces will be finally transferred to the relevant adapter that will complete execution.

Interaction with Distributed Storage

In PalletOne Core, we have defined the abstract interfaces for the operation of distributed storage. The core only operates the data in the distributed storage via the interfaces without caring about the specific implementation. DAG is a default distributed storage solution with high performance in PalletOne and will support more storage modes (e.g. IPS or other distributed databases) in the future. In terms of PalletOne Core, the distributed storage is a database for read-write transactions and supports common trading data operations and query of contract status and token UTXO.

1. Establish the connection with distributed storage

```
OpenConnection()
```

2. Read the stored data according to the trading ID

```
Transaction GetTxData(string txId)
```

3. Query the data of contract status

```
KeyValue[] QueryStateData(string contractId)
```

4. Query the historic data of contract status

5. Query the data of token UTXO

```
Utxo[] QueryUtxoData(string address)
```

6. Query historic transactions at token addresses

```
Transaction[] QueryAddressHistory(string address)
```

```
bool CheckTxConfirm(string txId)
```

7. Write in trading data

```
UnitId WriteTxData(string txId,Transaction trans)
struct Transaction
{
 string TransactionId
 Datetime CreateTime
 TokenPayment[] TokenPaymentSet
 KeyValue[] ReadSet
 KeyValue[] WriteSet
 ChainTrans[] InterchainSet
 PubkeySign[] AuthorSign
}
```

8. Check if the trading data is confirmed

The function is used to check if any transaction is confirmed; the “confirmed” means that the data will not be altered.

Disconnect from the distributed storage

```
CloseConnection()
```

Interaction with Blockchain Adapters

In PalletOne core, we have designed three operating interfaces in line with the features of the other blockchains - general digital currency adapter interface, UTXO model that based adapter interface and smart contract adapter interface. The smart contract enables cross-chain operations through these interfaces provided by the SDK.

General digital currency adapter interface

1. Wallet

- Generate public/private key pairs
- Generatel wallet address
- Generate the public key for corresponding private key
- Authenticate the legality of wallet address

2. Query

- Check account balance as per the wallet address
- Check historic transactions as per the wallet address
- Check trading objects as per trading ID
- Check trading status as per trading ID

3. Transaction

- Generate a transaction
- Sign a transaction
- Send a transaction to the network

UTXO adapter interface

Digital currency adapter proposes interface functions exclusively for UTXO based on the general abstract adapter interface:

- Check the existing UTXO based on wallet address
- Initiate a transaction with a given cost of UTXO

Smart contract adapter interface

Smart contract adapter proposes interface functions related to contract based on the general abstract adapter interface:

- Deploy the contract
- Initialize the contract
- Call the contract
- Destroy the contract

P2P Network

PalletOne has miner node and user node, of which the former contains the juror candidate list after payment of deposit and is divided into Mediator node and Jury node. Miner nodes are all full nodes with complete ledger data. Contract execution and consensus operation are required for miner nodes. Meanwhile, miner nodes must stay online for long and request a satisfactory calculation environment (CPU, memory and hard disk) and the network environment (high bandwidth and low delay). User nodes are those for browsing and transaction initiating. Full ledger is not necessary, and a longer delay is allowed for user nodes. The P2P network structure of the miner node and user node in PalletOne is shown in Figure 2.2.

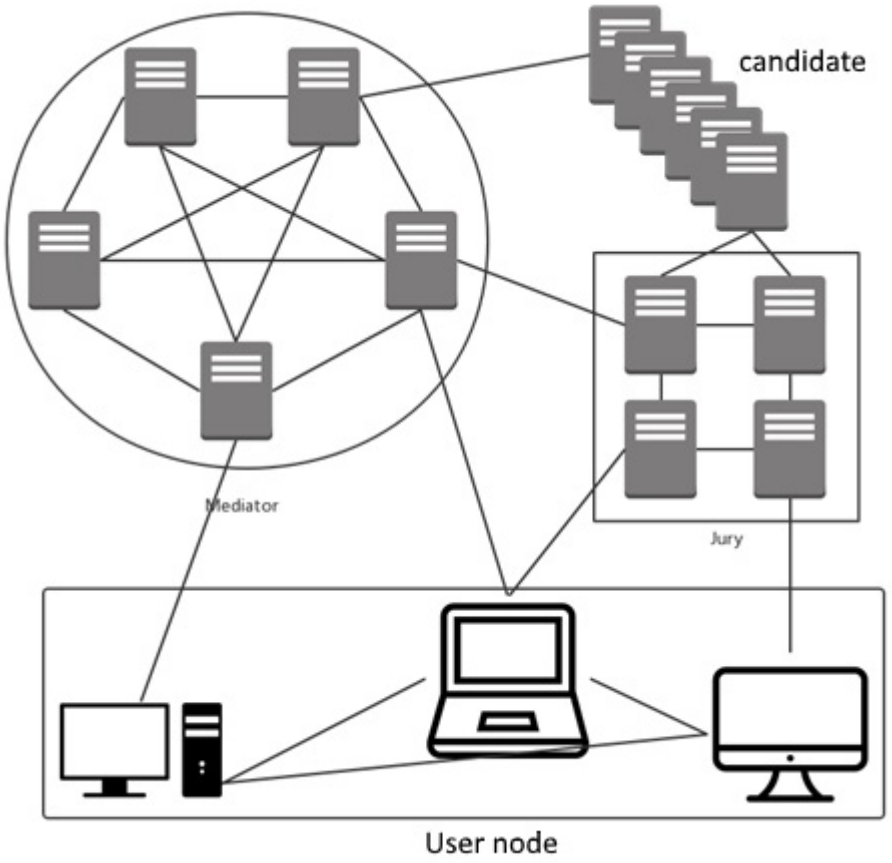

Figure 2.2 P2P network structure of miner node and user node

We assume that miner nodes stay online for long while user nodes may get online or offline at any time, but the situation that a large number of user nodes get online or offline at the same time will not happen.

The data in stored in all the nodes are ideally consistent. Miner node requires low delay; while timeliness is not necessary for the user node (all the data should be synchronized after a long time).

Mediators in PalletOne will set up P2P connection between each other, with an aim to lower the delay of data transfer among Mediators. Jury is a temporary team, meaning that the jurors are not required to be connected in pairs.

Information type in PalletOne

There are three types of network information in PalletOne:

1. Request – various requests made by users. If a user node receives a request, the request validity will be tested before forwarding. If a miner node receives a request, it will check if such request is related to itself after the request validity is tested and deal with the request if so or directly forward the request if not.

2. Unit Data – the unit data in DAG. All the nodes will test the validity of Unit after receiving Unit data and locally write in and forward it to the other node if valid.

3. Network Data includes the data related to registration, logoff and heartbeat.

Consensus Algorithm

Mediator Consensus Mechanism

DPoS Consensus Algorithm

DPoS algorithm applies witness mechanism to solve the problem of centralization. A total number of N (21 as default in PalletOne) witnesses (or Mediator super nodes in PalletOne) generate blocks by turns (Mediator's functions are specified in the chapter "Mediator"), and such witnesses are selected by voting of subjects in the whole blockchain network.

Advantages of DPoS algorithm

DPoS is more democratic than the other systems due to the decentralized voting mechanism. DPoS still holds some requirements on trust so as to guarantee that all the witness nodes of trusted & signed blocks representing the whole network are correct without bias. In addition, each signed block must have been proved that the previous block is signed by the trustable node. The transactions in DPoS can be confirmed without the authentication of a certain number of non-trusted nodes.

With less requirements on the number of recognized nodes, DPoS algorithm achieves faster trading. By trusting a small number of integrity nodes, DPoS removes the unnecessary steps during block signature. DPoS blocks can accommodate more transactions than PoW or PoS, thereby increasing the trading speed of encrypted digital currency to that of Visa, Mastercard and other central liquidation systems.

Centralization still exists in DPoS still holds, but is under control as each client is able to decide which nodes should be trusted. The DPoS-based blockchain retains plenty of advantages of some centralized systems and can guarantee the decentralization to some extent. Upon fair elections, DPoS ensures that every node is possible to become the agent for the majority.

Rational logic behind DPoS

- Shareholders have the right to select the bookkeeper upon voting.
- The dividend to shareholders can be maximized.
- The amount for cyber safe maintenance can be minimized.
- Network performance can be maximized.
- Cost of network operation (bandwidth, CPU, etc.) can be minimized.

Control power is held by interest owner

The fundamental feature of DPoS is that the interest owners hold the control power for system decentralization. Like the voting mechanism that has defects, DPoS is the only feasible way to manage the company's common property. Fortunately, if you do not like the company operator(s), you can exit by selling your interests. Such feedback mechanism can help interest owners become more rational than citizens in voting. Interest owners decide the signature verifier by voting, and anyone owns over 1% votes is entitled to the Board of Directors. All the agents constitute the "Board of Directors" and sign the blocks by turns. If any director misses the opportunity to sign the blocks, the clients will automatically vote for others. Finally, the directors who miss the opportunities will be disqualified and others will have the change to join the Board of Directors. The member of the Board of Directors will receive a certain number of token monies as rewards to encourage them to stay online and take part in the campaign for election. Each director must deposit 100 times the average bonus per block as a margin to ensure at least 99% of their online time.

Extensibility

Assuming that the recognition cost and service charge for each transaction are fixed, the number of decentralized transactions is restricted. Assuming that the verification cost is equal to the service charge, the whole network is fully decentralized and only supports one verification node. Assuming that the service charge is 100 times to the verification cost, the network will support 100 verification nodes.

Role of witness

1. Witness is given the right to generate and broadcast blocks.

2. The process of block generation includes the collection of transactions in P2P network and use of witness' key for signature.

3. Random assignment of witness for the next round at the end of the previous block (maintenance interval).

Attack inhibition by DPoS

1. If a witness does not generate any block, he may be dismissed and lose the predictable and stable income in the future.

2. The dishonest witness does not quite block generation until he has clear interest demand.

3. The witness cannot sign the invalid transactions as the transactions should be confirmed by all the witnesses.

Mediator Super Node

To participate in PalletOne consensus, it is required to pay a deposit in advance to become the candidate of the super node.

Selection of Mediator super node

PalletOne Token holders can select a random number of Mediator witness nodes from the juror candidates. As for each PalletOne Token account, every Token can cast a vote to a witness node or several witness nodes, and the process is called approval voting.

N (21 as default) witnesses with the most votes are Mediator super nodes. The witnesses failing to become super nodes may request deposit returning to exit the list of super node candidates, or keep the deposit to take part in the contract execution with the Jury and wait for the substitution of any other super node.

Trading fee

In contract deployment, a certain amount of trading fee will be paid to the super node who verifies or executes a contract and generates data unit, and the amount will be determined by PalletOne Token holders upon voting. If any witness node fails to generate any witnessed unit, he/she will not receive any trading fee and may be disqualified by PalletOne Token holders upon voting.

Updating of super node

The online data of super nodes will be updated on a daily basis. Meanwhile, the votes will be also counted every day for super node updating. Super nodes conduct "bookkeeping" by turns – each node verifies the unwitnessed unit and generates witnessed unit within N seconds (5 seconds as default, alterable upon community voting). After all the super nodes complete a round of polling, nodes will be updated once. If any super node fails to generate any data unit within a duration, such duration will be automatically skipped and the next node will continue "bookkeeping".

When the online rate of super nodes is lower than a certain level

Anyone can monitor the health condition of the whole network by viewing the online rate of super nodes. The participation rate of super nodes is required to reach 99% by PalletOne. If the participation rate of a super node is lower than a certain level at a time, PalletOne users can request such node to exit the list of super nodes.

In traditional DPoS consensus mechanism, every witness is requested to broadcast the generated block(s). However, even if the other witnesses receive the new block(s). They are still unable to recognize the new block(s) until they complete block generation. Therefore, the transaction can only be recognized after a long time.

In order to shorten the time of recognition by traditional DPoS algorithm, we have introduced BFT consensus mechanism in Mediator. Under the new mechanism, every witness makes international broadcast in Mediator before block generation, and the other witnesses, after receiving the new block(s), will verify such block(s) and return the block with verified signature to the block witness without waiting for the witness' block generation for recognition. In terms of the current witness generating block, he completes the internal broadcast and then receives the recognition from other witnesses. If makes the broadcast soon after receiving the recognition of 2/3 of all witnesses, the witness unit and trading unit therein will become irreversible. The time of transaction recognition is significantly reduced. This mechanism is also adopted in EOS.

Jury Consensus Algorithm

Algorithm for Random Selection of Jurors

Verifiable random function is abbreviated as VRF.

Why does PalletOne Jury depend on randomness?

In blockchain, only the method of random number generation by cryptology can help manage a large number of clients (miners) in an anti-attack network to build a super virtual machine. Of course, Satoshi Nakamoto designs the method of asking miners to calculate random problems to generate randomness. PalletOne Jury needs a stronger and more efficient random number generating algorithm that cannot be easily manipulated. Only in this way can the jury unpredictability be guaranteed, thereby lowering the risks of hacker attack and miner's malpractice.

VRF can be easily understood as it is used to complete the random selection for Pallet Jury. For this purpose, the returned value of VRF should be unpredictable. For PalletOne, the previous random number (the original random number is given by protocol) and a variant (e.g. hash value for trading) are combined and then signed with a certain private key (or vice versa), and at last the new random number is generated after hash. The random numbers generated in this way can be easily verified by others for algorithm satisfaction, meaning that the verifiability is assured. Meanwhile, the returned hash values are randomly distributed, which also guarantees the randomness.

There are two aspects should be noticed in this process in order to reduce the possibility of result manipulation:

1. The signature algorithm should be unique. The same private key should be applied for signing the same information and only one legal signature can be verified, which is not practicable for the ordinary asymmetric encryption/decryption algorithms like SM2. If the signature algorithm applied is not unique, the repeated signatures may be applied to select the most advantageous one in random number generation, showing a lower security.

2. It is necessary to avoid taking the data of the current block as one of the sources of randomness in generating new random numbers, such as the introduction of Merkle root value of the list of current block trading, as the block generator may try to alter the sequence of packaging transactions or to package different transactions to generate the new random numbers with the highest advantage.

In designing and inspecting the new consensus algorithm, the following two aspects should be noticed. The algorithm we adopt is PalletOne Jury consensus based on cryptology, which can help generate nearly invincible, totally uncontrollable and unpredictable random numbers in a network with a large number of participants. Based on PalletOne Jury consensus, every participant in PalletOne Jury can generate a verifiable random function (VRF), based on VRF, PalletOne Jury is completely independent for self-organization, operation and management.

About cryptology

1. Pseudo-random number generator (PRG) can change the seed number ξ into a sequential value PRG (ξ, i)=0, 1, 2….

2. Pseudo-random number ordering: PRG (ξ, i) is entered as a Knuth-Fisher-Yates Algorithm to generate a randomly ordered PermU (ξ).

Threshold signature

Threshold signature is a paradigm. Keys can be divided into N pieces in different servers to enhance the system practicability and anti-attack performance. In (t,n) threshold encryption system, the operation on the keys should be conducted under coordination of t+1 in N servers (The mechanism can be regarded as the extension of Shamir private key sharing). The public key of the function (signed & verified key) has not changed compared to ordinary format.

The threshold signature is originated from the distributed protocols. Threshold homomorphic encryption is applied in the voting system and multi-party calculation protocols. Threshold signature features higher sensitivity and private key security like the verification subject. It can be also used as a tool for distributed storage system.

Threshold signature solution – BLS signature

BLS signature was proposed by Boneh, Lynn and Sahcham in 2003.

● BLS function

Assuming that we have generated a private key/public key pair (sk, pk), BLS provides the following functions:

1） Sign (m, sk) – sign information with the private key sk and return the signature σ.

2） Verify (m, pk, σ) – verify the signature of information with the public key and return "true" or “false”.

● Threshold BLS

Threshold BLS, also known as TBLS, provides another function:

Recover function – recover the group signature σ from signature sharding.

The Implementation of VRF in PalleOne

In PalletOne, the implementation of VRF adopts threshold signature (t-of-n) and this threshold signature solution uses a DKG (Decentralize Key Generator) (no trusted third parties are needed), which enable several participants to generate keys jointly (e.g. public key in the group and each private key sharding). DKG protocol is not only a private key sharing protocol. In the private key sharing protocol, private key sharding can be used to recover private keys in the group for only one time as the private key sharding cannot be used after each participant obtains the private keys in the group. But in DKG, the private key sharding can be used without any limitation so that no operation is needed to recover private keys in the group.

In the threshold signature solution, n participating nodes are combined to establish a public key (public key in the group), each participant keeps a private key sharding (private key sharding in the group). After the steps above are completed, only t of n of the participating nodes are capable to create a signature that can be verified by the public key in the group.

The threshold signature solution in PalleOne has two characteristics:

（1） Non-interactivity

If a threshold signature solution only conducts one-way communication with each participant for one time during the process of creating group signature, then this threshold signature solution is considered to have non-interactivity. In the non-interactivity solution, each participant use its own private key to

create a part of the signature sent to the third party that can recover the out group signature once it has collected t legal signature parts.

（2） Uniqueness

The uniqueness of signature solution means that for every piece of information and every public key, there is only one signature is legal. But in the threshold signature solution, there is an additional requirement, i.e., no matter which participant signs, the final generated group signature must be the same.

The jury consensus includes two parts in PalletOne:

（1） the establishment process is only performed once (using DKG)

（2） repeat signing without limit and generate output.

Establishment process: run DKG in the group and establish public key and private key sharding in the group, when this step is completed during the process of blockchain system initialization. After DKG is completed, a public verification vector and private key shardings held by each node in the group are generated. The public verification vector can be used to recover the public key corresponding to each private key sharding and the public key sharding corresponding to each private key will be published to each node. The signature sharding generated by each node and the public key in the group can verify the results of Recover Function.

Signature Process: each node generates a signature σ=Sign (r||ξr-1, sk) in the r round, of which ξr-1 is the random value generated in the r-1 round, and broadcast the this signature. Once any nodes have received this message, they can verify this signature with the public key. If the verification result is correct, they will save and broadcast this signature. If one node has received t signatures, it can run Recover Function to calculate out group signature and then the random value ξr in the r round can be generated by hashing this group of signatures.

Initialization and Updating of Grouping in PalletOne

1. Grouping initialization

When the system is initializing, Mediator is responsible for randomly grouping nodes in the entire network

1) Assume the quantity of grouping is n and according to the random seed value ξ, the jury will generate:

```
Group（ξ, j）: = PermU （PRG（ξ, i））（{1.....n}）
```

2) At the beginning of the system, we set this value to be m, then:

ξ can be set as the hash value of PalletOne.

3) After grouping, the grouping information will be sent to all nodes in the whole network by broadcast for caching. Each node in each group identified as Nodeid is the node with its name passing the public signature and performing hash calculation.

2. Update of grouping

Every t seconds (assuming that t=3600), all nodes in the whole network will be re-grouped by the Mediator.

```
Gj=Group（ξ, j）j=1...m
```

ξ can be set as the hash value of the data unit signature that is the last broadcast of super node which

the Mediator is currently "on duty".

3. The random selection of Jury

When Mediator receives a contract request for the rth time, it first broadcasts the contract request message, and randomly selects jurors to form a jury for the contract. The jury selection method is as follows:

1) Assuming the jury selected is Gr, we set:

```
Gr=Gjj=r mod m
```

Wherein ξr is the hash value of the contract ID.

2)The information (including the contract ID, the hash value of the contract ID, and the selected jury) is placed in the data unit for broadcasting.

4. Method of issuing juror nodes

After receiving the data unit, the nodes throughout the entire network can verify whether it is selected as the juror node by the private key. Meanwhile, all the nodes can also verify whether the selection of the jury is correct according to the hash value of the contract ID combined with the jury pre-grouping in the cache.

5. Node ranking

Based on ξr, PalletOne can rank each node in the jury; this ranking can be defined as PermU(ξr), and the data unit with the jury consensus is released by the node with the highest Rank value.

6. Threshold relay

From the perspective of scalability, the above random numbers must be generated within a fixed number of nodes (21 Mediator nodes) rather than within the entire network. Otherwise, the complexity of the information will grow indefinitely as the number of nodes grows.

For the non-locked jury mode, it is the n nodes randomly selected in the whole network. These randomly selected nodes constitute a jury. After the jury is formed, the threshold operation is performed to select the current jury, and then the current jury is relayed to the next jury. This method is called "threshold relay".

Consensus within the Jury

After the jurors are randomly selected by the VRF algorithm to form a jury, the jury will perform the following steps to complete the consensus on the contract execution:

1. All juror nodes execute the contract respectively according to the contract call request, and package and broadcast the contract result hash signature to the network.

2. A juror in the current jury is elected as the Leader according to a certain algorithm rules.

3. The Leader is responsible for collecting the result hash performed by all the juror nodes. If the threshold value of the contract is met, the result hash is packaged into a contract call unit, written to the DAG database and broadcast to the network.

4. If the elected Leader does not complete the packaging and broadcasting within the specified time, a new Leader will be elected to perform the packaging and broadcasting.

Contract execution

After the jury is formed, it can start the PalletOne virtual machine to run the smart contract based on the contract call request. There should be no random numbers, time variables, etc. in the smart contract, so no matter which juror node it is executed in, the same output will be generated for the same input. The juror computes the hash with the output result of the contract execution along with the input parameters, signs the hash with its own private key, and then sends the contract input and output hash, signature, execution time, and other relevant information to the P2P network.

Election of jury Leader

When each juror receives the same execution result hash of more than 2/3 of the nodes and their respective signatures, it will judge whether it is the Leader for the current contract execution according to whether its signature hash is the smallest of all hashes.

Generation of contract call unit

Knowing that it is the Leader by comparing all signatures, the Leader packages the input and output of the contract, the jurors' signatures, etc., plus the referenced parent unit, timestamp, signature, etc. to generate a contract execution unit.

Time-out and re-election of Leader

If all jurors still do not receive the contract execution unit packaged by the Leader after waiting for N seconds, the Leader node is considered to be faulty, and the juror node with the second smallest signature hash will be elected as the Leader node to package the contract execution unit. If the unit is still not received after waiting for N seconds, the Leader election and the packaging work will be repeated.

The consensus model of BFT is adopted in the jury, in which the Leader is elected by the signature hash of the contract execution result and finally enjoys the fee for this execution. If the juror's computer configuration is poor, causing slow contract execution and not timely acquisition of contract execution result, then the opportunity of serving as the Leader in packaging unit and charging fees may be missed. Therefore, the jurors are also motivated to configure better computer resources to increase their opportunity of obtaining transaction fee.

Distributed ledger storage

The structure of distributed storage of PalletOne is shown in Figure 4.1.

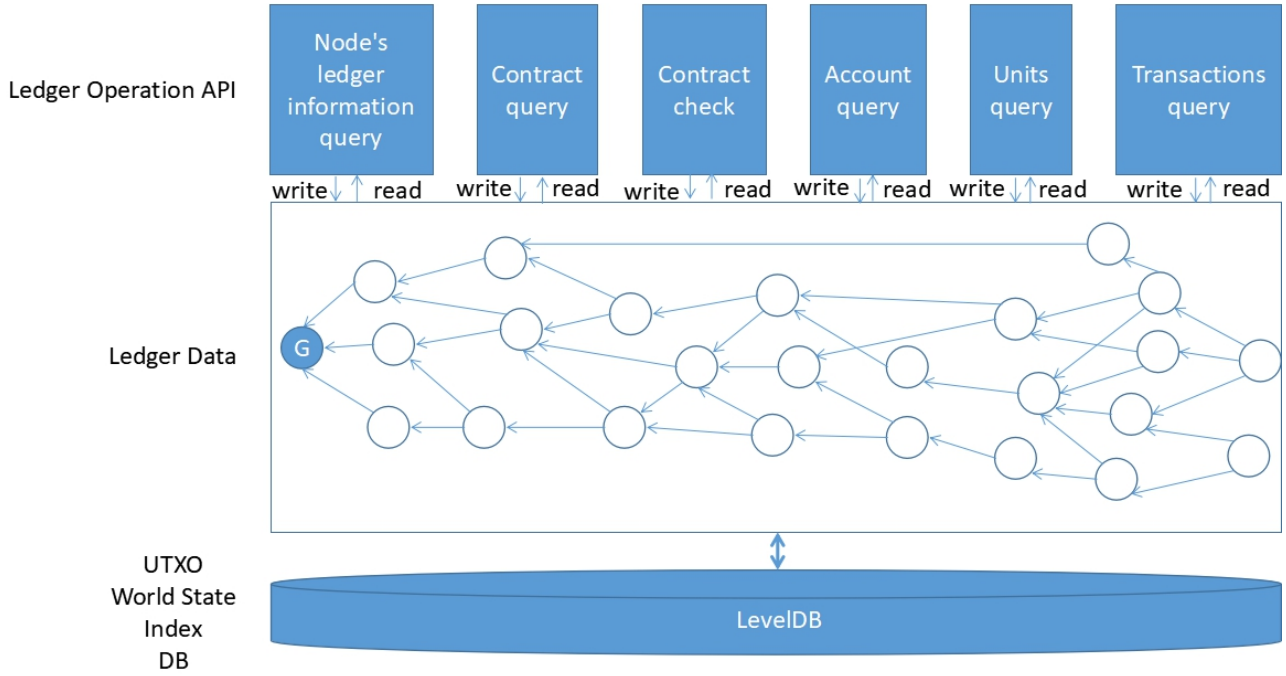

Figure 4.1 The distributed storage structure in PalletOn

Directed Acyclic Graph

Directed Acyclic Graph, abbreviated to DAG, does not feature the concept of Block, meaning that all the data in it is not packaged into blocks and then connected by a blockchain. Instead, it features that each user in it can submit a data unit into which many things, such as transactions and messages, can be wrapped, and then such data units connect with each other by mutual reference, as shown in Figure 4.2. It is characterized by an asynchronism in data unit write operation, meaning that a large number of nodes can autonomously write transaction data into it. The purpose of PalletOne's use DAG is to solve the problem in a traditional blockchain that it has only a main chain and cannot execute contracts in parallel, and to save the time for packing transactions to form blocks.

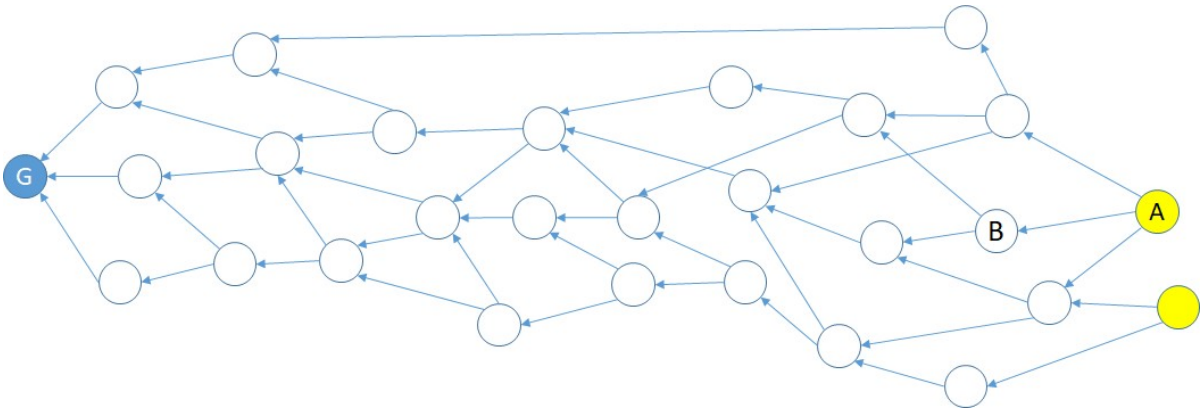

Figure 4.2 DAG structure in PalletOne

In a traditional blockchain, before generating a block, a miner needs to pick a transaction which he/she wants to package from a pool made for all the transactions, then package it into a block according to a consensus algorithm, and finally put them in the chain. Before being packaged into the block and broadcast to the entire network, the transaction remains in an unconfirmed state, which will hinder the

write operation of other transactions and thus make the transaction confirmation time in the traditional blockchain very long.

DAG's characteristic of parallel execution make it possible to bring new technological innovations and upgrades to economic activities and value transactions in various industries to bridge the wide gaps in trust between different trading entities, and at the same time to reduce transactional costs and realize more expressive smart contracts, faster confirmation of transactions, broader application scenarios, stronger security and privacy protection with its characteristics such as time stamping, irreversibility, traceability, parallel processing and other features.

Unit structure

In the "DAG structure in PalletOne" shown in Figure 4.2, the structure of a storage unit of the DAD is called Unit. The units can be divided into the following categories:

(1) Foundation unit: It, also known as G unit, is a unit that was generated as the first transaction in the DAG took place and does not have any parent unit and is the end point that all subsequent units can reach by references.

(2) Parent and child units: Assuming that there are unit A and unit B, and unit A can directly reach unit B, that is, the length of the path from unit A to unit B is 1, then unit B is called the parent unit of unit A and unit A the child unit of unit B.

(3) Top units: They do not have any child units, thus can also be referred to as no-child units or unverified units, as the ones indicated by the yellow color in the above figure.

In the structure of a DAG unit, there are the following constituents:

(1) Unit data: The data is composed in the form of Message;

(2) Address signature: Input the required corresponding address signature;

(3) Parent units: A list of parent units of the current unit.

The apps appearing in the field of message can be divided into the following categories:

(1) A PalletOne token transaction;

(2) Deployment of the template of a smart contract;

(3) Creation of a smart contract (instantiation);

(4) Invocation of a smart contract;

(5) Verification of a transaction/the deployment of the template /creation / invocation of a smart contract

(6) .....

The following will explain respectively the above five categories of messages.

Token transaction

Taking the structure of a token transfer transaction unit in PalletOne as the example, the following will illustrate the JSON format of storage unit:

```
{
    version: '1.0',
    messages: [{
          app: 'payment',
          payload_location: 'inline',
          payload_hash: 'the hash text of payload field',
          payload: {
                inputs: [{
                      unit: 'create token unit hash and this unit must have
 became a stable packet',
                      message_index: 0,
                      output_index: 1
                }],
                outputs: [{
                      address: 'transfer to address',
                      amount: 208,
                      asseId: 'an immutable identifier of an asset that is
 created at issuance',
                      uniqueId: 'every token has its unique id'
                },
                ......]
          }
    }],
    authors: [{
          address: 'create or sign author addresses for this unit',
          pubkey: 'the authors's public key if needed ',
          authentifiers: {
                r: 'the authors 's ECC signature'
          }
    }],
    parent_units: ['referenced unit hash text', 'referenced unit hash
 text', 'referenced unit hash text'],
    last_packet: 'the hash of last packet',
    last_packet_unit: 'the unit hash of last packet',
}
```

(1) The version refers to the number of the protocol version. The unit will be construed in accordance with it;

(2) Messages refer to an array of one or more message(s) containing actual data:

- App represents the type of the messages. For example, “payment” stands for payment; “text” stands for arbitrary text information; and so on.
- Payload_location indicates where to find out the effective payload in the messages. If the effective payload is included in the messages, the “payload_location” will be "inline"; if available at an Internet address, it will be "uri"; if the payload is not published, it will be "none", which means that the effective payload is privately stored and/or shared, and will be proved existing at a specific time by the payload_hash;
- Payload_hash is the hush of the effective payload in base64 encoding.

Payload is the actual load (In the example above is "inline"). The structure of an effective payload is a specific application. The struction on the payment is described as follows:

- Inputs represents a string of input currencies consumed by the payment (command). The owner of the input currencies must be included in the authors of the unit;
- Unit refers to a hash unit of producing currencies. The unit can be consumed only after being counted into the unit of the last packet;
- Message_inde refers to the index of the UTXO message

- Output_index refers to the index of the UTXO output

➢ Outputs represents a set of outputs that indicate who received the money;

- address (address) refers to the address of the recipient, which can be an external account address or a contract address.
- Amount refers to the amount the recipient can received;
- AssetId refers to the immutable identifier generated when the token identifier is issued;
- UniqueId is the unique identifier each type of tokens has.

(2) Authors refers to the array of authors who create and sign this unit. All the input currencies must belong to the authors;

- Address refers to the addresses of the accounts corresponding to the tokens of authors;
- Pubkey refers to the public keys of the authors who sign the unit and is an optional field. When needed, any other units, such as an authentication unit, can be added to it.
- Authentifiers are the author's signatures, The most common format of which is the ECDSA (Elliptic Curve Digital Signature Algorithm) signature;

(3) Parent_units refer to a hash array of parent units arranged in alphabetical order;

(4) .Last_packet and Last_packet_unit refer respectively to the hash values of the last packet and its unit.

All hash values in the structure are encoded in accordance with Base64

contract_template：Deployment of the template of a smart contract

```
unit: {
    …
    messages: [{
        app: "contract_template",
        payload_location: "inline",
        payload_hash: "hash of payload field",
        payload: {
            config: "configure xml file of contract",
            bytecode: "contract        bytecode",
        }
    },
    {
        app: 'payment',
        payload_location: 'inline',
        payload_hash: 'hash of payload field',
        payload: {
            inputs: [{
                unit: 'create token unit hash and this unit must have became a stable packet',
                message_index: 0,
                output_index: 1
            }],
            outputs: [{
                address: 'the address of contract issuer, the difference between inputs and
outputs is the transaction fee',
                amount: 208,
                asseId: 'an immutable identifier of an asset that is created at issuance',
                uniqueId: 'every token has its unique id'
            },
            ]
        }
    },
    ],
    authors: [{
        address: 'create or sign author addresses for this unit',
        pubkey: 'the authors's public key if needed ',
        authentifiers: {
            r: 'the authors 's ECC signature'
        }
    }],
    …
}
```

In the field of messages featuring a message type of "contract_template" of the above unit, subfields below the payload field have the meanings as follows:

(1) Config refers to the serialization value of the xml files that the contract template configured. In it, the language and initialization parameters and a detailed list of functions of the contract shall be indicated.

(2) Bytecode refers to the serialization value of the packaged file of the contract compilation files. Because the contract ends up with more than one file and there may be many reference files or libraries, a zip package is needed to transmit them.

In the above unit, the field of messages featuring a type of "payment" indicates the commission for deployment of the contract; the one of outputs points to the addresses of the authors who deployed the contract; and the value of the difference between inputs and outpus is the transaction fee paid to the jury.

The field of “authors” has the same meaning as the one of "payment".

contract_deploy：Creation of a smart contract (instantiation)

```
unit: {
   …
   messages: [{
         app: "contract_deploy",
         payload_location: "inline",
         payload_hash: "hash of payload field",
         payload: {
               template_id: "contract template id",
               config: "configure xml file of contract instance
parameters",
         }
   },
   {
         app: 'payment',
         payload_location: 'inline',
         payload_hash: 'hash of payload field',
         payload: {
               inputs: [{
                     unit: 'create token unit hash and this unit must have
became a stable packet',
                     message_index: 0,
                     output_index: 1
               }],
               outputs: [{
                     address: 'the address of contract issuer, the
difference between inputs and outputs is the transaction fee',
                     amount: 208,
                     asseId: 'an immutable identifier of an asset that is
created at issuance',
                     uniqueId: 'every token has its unique id'
               },
               ]
         }
   },
   ],
   authors: [{
         address: 'create or sign author addresses for this unit',
         pubkey: 'the authors's public key if needed',
         authentifiers: {
               r: 'the authors 's ECC signature'
         }
   }],
   …
}
```

In the field of messages featuring a message type of ““**contract_deploy**” of the above unit, the subfields under the field of payload have meanings as follows:

(1) Contract template ID.

(2) The serialization value of the xml files that the contract configured. In it, the actual initial values of the initialization parameters of the contract needs to be indicated.

In the above unit, the field of messages featuring a meassage type "payment" indicates the commission for deployment the contract; the one of outputs points to the addresses of the authors who deployed the contract; and and the value of the difference between inputs and outpus is the transaction fee paid to the jury.

The field of “authors” has the same meaning as the one of "payment".

contract_invoke：Invocation of a smart contract

```
unit: {
   …
   messages: [{
         app: "contract_invoke",
         payload_location: "inline",
         payload_hash: "hash of payload",
         payload: {
               contract_id:  “the hash id of contract”
               function: "serialized value of invoked function with call
parameters"
         }
   },

   {
         app: 'payment',
         payload_location: 'inline',
         payload_hash: 'hash of payload field',
         payload: {
               inputs: [{
                     unit: 'create token unit hash and this unit must have
became a stable packet',
                     message_index: 0,
                     output_index: 1
               }],
               outputs: [{
                     address: 'the address of contract issuer, the
difference between inputs and outputs is the transaction fee',
                     amount: 208,
                     asseId: 'an immutable identifier of an asset that is
created at issuance',
                     uniqueId: 'every token has its unique id'
               },
               ]
         }
   },

   ],
   …
}
```

In the field of messages featuring a message type of ““**contract_invoke**” of the above unit, the subfields under the field of payload have meanings as follows:

(1) The ID of the contract to be called;

(2) The serialization value of the functions with the invocation parameters;

In the above unit, the payment type messages have the same meaning as the contract deployment. The field of “messages” featuring a message type of “**Payment**” has the same meaning.

contract_deploy：Verification of a token transaction/the deployment of the template /creation / invocation of a smart contract

```
{
    version: '1.0',
    messages: [{
        app: 'verify',
        payload_location: 'inline',
        payload_hash: 'the hash text of payload field',
        payload: {
            inputs: [{
                unit: 'create token unit hash and this unit must have
became a stable packet',
                message_index: 0,
                output_index: 1
            }],
            outputs: [{
                address: 'transfer to address',
                amount: 208,
                asseId: 'an immutable identifier of an asset that is
created at issuance',
                uniqueId: 'every token has its unique id'
            },
            ......]
        }
    }],
    authors: [{
        address: 'create or sign author addresses for this unit, the
first author could be the unit author',
        pubkey: 'the authors's public key if needed ',
        authentifiers: {
          r: 'the authors 's ECC signature'
        }
       }],
    parent_units: ['referenced unit hash text', 'referenced unit hash
text', 'referenced unit hash text'],
    last_packet: 'the hash of last packet',
    last_packet_unit: 'the unit hash of last packet',
}
```

In the above unit, all the fields have the same meanings as in the unit of “payment”. The field of authors will include signatures of all the verifler in a unit of “verify”.

Status Storage

Account statuses

PalletOne employs a UTXO model at its bottom, which is different from the account model of Ethereum. But there also are external accounts, which can be divided into single-signature and multi-signature accounts, and contract accounts in PalletOne. An account in PalletOne, whether an external or contract account, has the following states:

- codeHash: The hash of the contract code owned by the account;

- storageRoot：The MPT Root of the status data saved in this account.

Smart contract storage and verification

In the DAG, the statues of a contract are written into in real time and in parallel, so the state information of the contract is stored directly in a unit of the structure of the DAG. But in an external and contract account, the two states of codeHash and storageRoot are still reserved for the convenience of inquiry and storage and verification of the account.

Token Defination

In PalletOne, a token is defined at the layer of token abstraction, and is not stored in the database of contract states but in a separate token storage area.

Tokens created at the layer of token abstraction, each of which has only a few fixed external interfaces similar to ones defined by ERC20 and ERC721, are stored directly in the UTXO database, and are distinguished on the basis of asset Id and unique Id.

If a high-level token operation needs to be defined, we need to write a custom contract to complete it. By referencing the custom token, the contract can have control over it.

Database Design

In PalletOne, we will design four types of databases, as shown in Figure 4.3., which will employs the LevelDB in key-value format to store data. The four types are:

- DAG raw database - all data will be stored in the database in the structure of DAG.
- Index Database - Some historical contract states and other historical data that needs to be retrieved in smart contracts will be stored in the database and indexed.
- UTXO database -The UTXO data of all the tokens will be store in it.
- Contract State Database - The latest state data of all the contracts will be stored in it

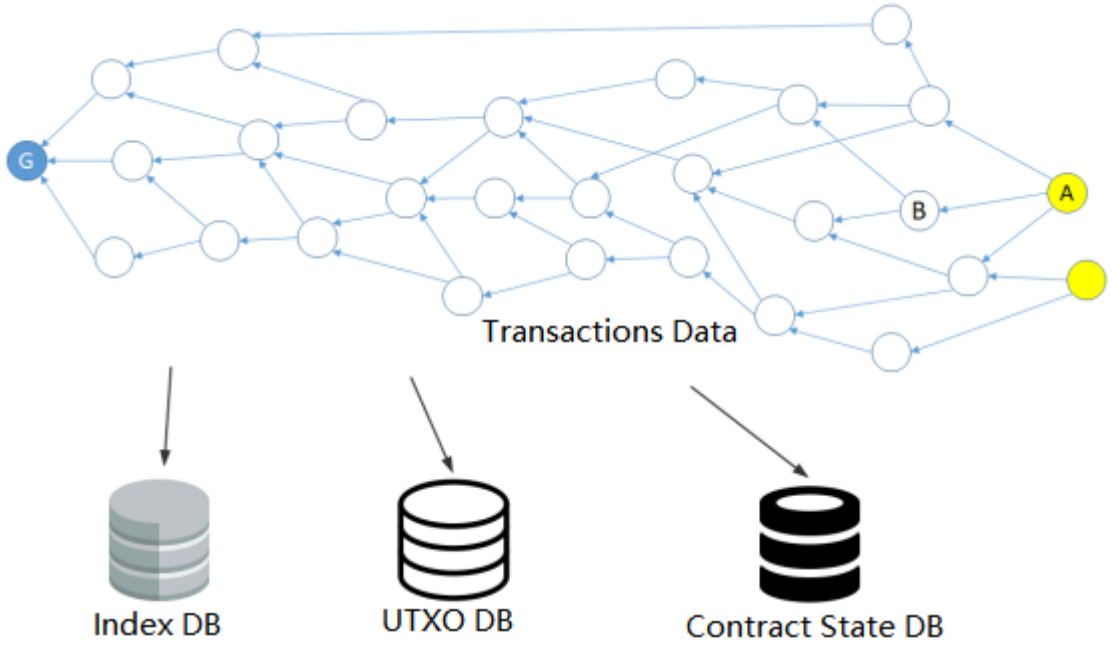

Figure 4.3 Types of databases in PalletOne

The LevelDB will be employed to store data at the underlying layer of the databases in PalletOne. LevelDB is a key-value database, storing data in a way one-to-one correspondence between keys and values. A key is generally related to the storage type and the hash, and a value generally the data structure to be stored.

UTXO (Unspent Transaction Output) Storage

Taking the UTXO storage format in Bitcoin, PalletOne will employ the following UTXO format:

- Key

'c'（或者 0x63）（or 0x63	ballHash

- value

the non-spent CTxOut	nHeight

The non-spent CTxOut has the following structure:

amount	scriptType	address	assetId	uniqueId

（1） The amount of the outputs;

（2） scriptType：00-P2PKH；01-P2SH 3; Script types of 00-P2PKH and 01-P2SH 3;

（3） address：The address of outputs;

（4） assetId： The unique identifier of a token;

（5） uniqueId： The identifier of a type of tokens;

Contract state storage

- Key： 'a' + Addresses of accounts of a contract
- value

codeHash	storageRoot

DAG

DAG raw data storage

1. Unit data storage
   - key：'u' + Unit hash
   - value：rlpEncode(unitdata)
2. Packet data storage
   - key：'b'+Unit hash
   - value: rlpEncode(packet data)
3. Tarnsaction data storage
   - key: 't'+payload hash
   - value: rlpEncode(payload data)

DAG Index data storage

1. Transaction-unit index
   - key: 'tu'+payload hash
   - value: unit hash
2. Contract-transaction index

- key: 'ct'+contract id
- value: payload hash

DAG Network

Recording right selection

In PalletOne, the following categories of units need to be recorded:

1. Ordinary transaction units

The recording right of the ordinary trading units shall belong to the transaction initiator.

2. Contract template deployment units

The recording right of the contract template units shall belong to the members of Mediator who work at a certain time slice.

3. Contract creation (instantiation) units

The recording right of the contract creation (instantiation) units shall belong to the members of Mediator who work at a certain time slice.

4. Contract invocation units

The recording right of the contract calling units shall belong to the Leader of the Jury. But it should be noted that although the Leader possesses the recording right, it needs to obtain the signatures of a majority (>1/2 or all) of the jury members to preform such right, when a contract is created and invocated.

5. Transaction confirmation units

Transaction confirmation units shall be recorded by the currently active members of Mediator, but the recording can be completed only when all the members of Mediator signed the units.

6. Contract template deployment authentication unit

Contract template authentication units shall be recorded also by the currently active members of Mediator, whose tasks in authentication of a contract template deployment mainly include checking contract parameters, contract codes, transaction fees, and so on. Similar to the above, the recording of a contract template authentication unit can be completed only when all the members of Mediator signed the unit.

7. Contract creation authentication units

Contract authentication units shall be recorded also by the currently active members of Mediator, whose tasks in authentication of a contract creation mainly include checking contract parameters, contract codes, Initialization Parameters, transaction fees, and so on. Similar to the above, the recording of a contract creation authentication unit can be completed only when all the members of Mediator signed the unit.

8. Contract invocation authentication units

Contract invocation authentication units shall be recorded also by the currently active members of Mediator, whose tasks in authentication of a contract invocation mainly include checking invocation compliance, method parameters, and so on. Similar to the above, the recording of a contract invocation authentication unit can be completed only when all the members of Mediator signed the unit.

Parent unit selection

The selection of parent units is divided into two cases: that one is the selection of parent units by the ordinary transaction, contract template deployment, contract creation and contract invocation units; and the other is the one by the authentication units.

In the first case, the selection shall abide by the principle of proximity, that is, all the child units of them are sorted according to their hashes, and then the smaller units as many as possible shall selected as the parent units. The purpose of employing the principle is to converge the DAG and finally make it a

chain.

In the second case, the principle for the selection is to directly reference the authenticated transaction units.

Packet Generation

After a unit is stable, based on the structure of which, we can generate a new structure as follows:

```
packet: {
        unit: "hash of unit",
        parent_packets: [array of hashes of packets based on parent units],
        is_nonserial: true, // this field included only if the unit is
nonserial
        skiplist_packets: [array of earlier packets used to build
skiplist]
}
```

The purpose of introducing the concept of packet is to help the authentication of a light nodes. In it, the filed of skiplist_packets functions to accelerate the authentication. Skiplists exist only in the packets which are connected directly with the main chain (MC) in the way that each such pocket has one skiplist. It has MC indexes, which can be divided by 10, and lists the packets closest to the previous MCs. The index at the end of it has the same or a smaller number of zeros. For example, the packet at MCI (MC Index) 190 has the shiplist referring to MCI 180; The packet at MCI 3000 has the skiplist referring to MCI2990, 2900 and 2000.

In the generation of it:

- Whether a unit is stable or not is determined during the process of determining a main chain, where the foundation unit is a default stable point.
- MCI represents a main chain index value. The MCI of the foundation unit is 0, and the child units of it is 1. A unit that is not on the main chain wall have a MCI which shall firstly include (directly or indirectly) its own MC index.

The last packet

In order to protect the packet (most importantly, the non-sequence identifier) from being not modified, we require that each new unit have to contain the hash of the last packet known to the authors (it is the packet created on the last stable unit and is located on the MC)). In this way, the last packet will be protected by the authors' signature. After that, the authentication unit will (directly or indirectly) be counted into the new unit itself.

Main chain

A main chain shall be determined through the following three steps:

1. Selecting optimal parent unit

a) The parent unit with the highest witness level shall be the optimal parent unit;

b) In the case of a same witness level, the unit with the lowest unit level shall be the optimal parent unit;

c) In the cases of same witness and unit levels, the unit with the smaller hash value (encoded based on Base 64) shall be the one

2. Generation of candidate main chains

Optimal paths from any top unit to the foundation unit are called the candidate main chains, which shall be determined during the selection of the optimal parent unit. Different candidate main chains may intersect at a certain unit location (the worst is at the foundation unit). The intersection location shall be

called Stable Point.

3. Generation of the stable main chain

For all the candidate mainchains, the path from the stable point to the foundation unit is exactly the same. This path is called Stable Mainchain, as shown in Figure 4.4., which is a certain path. Changing from candidate maichains to a stable mainchain is a process from uncertainty to certainty.

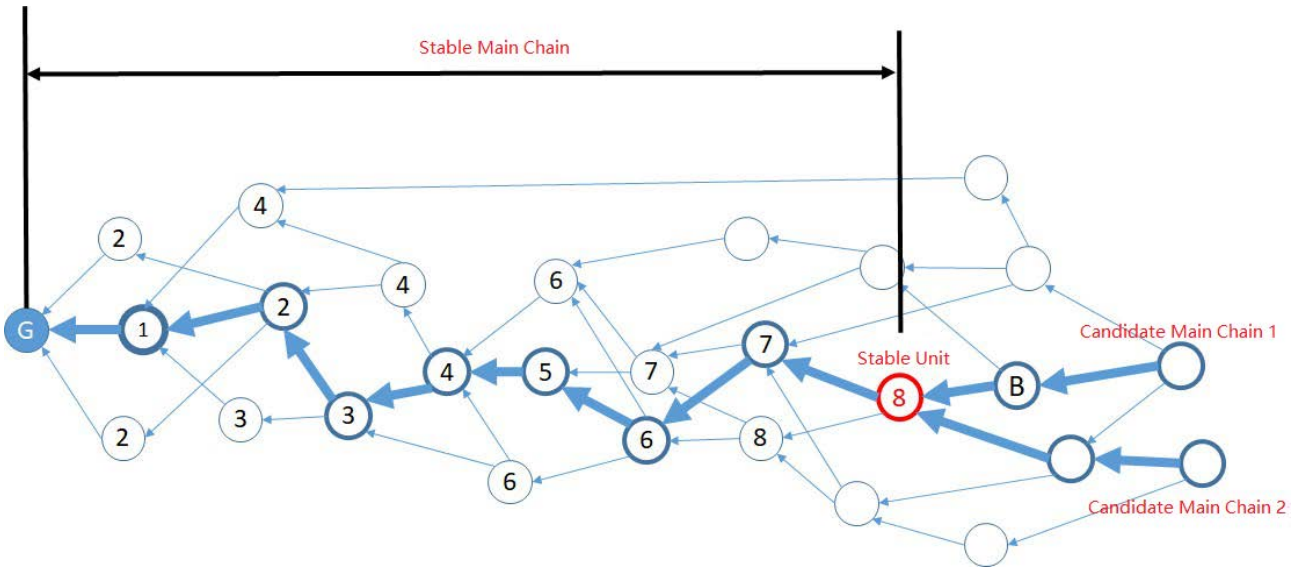

Figure 4.4 Stable Mainchain in PalletOne

Among them:

- Unit level: The lenghth of the longest path from a unit to the foundation unit.
- Witnessed level: A backtracking is begun from a unit along the mainchain and at the same time begin to count the number of different mediators encountered in the process(the repeated ones can only be counts as one) and stopped when the number is sufficient (here the sufficient number can be set to 11, because the mediator is constantly on rotation and the members of it are number at 21). Then calculate the unit level of the stop position. This unit level shall be deemed the witnessed level of the unit.
- Current Mainchain: Since different candidate main chains can be formed, starting from different top units at a certain moment, we will refer to the candidate main chain starting from the top node with the highest witness level as the Current Mainchain.

Double Spending

When an address issues a new unit, the newly-issued unit is required to directly or indirectly include all units issued by the address before, that is, all units of the same address are connected (to guarantee order or continunity). Therefore, the judgment of a double spending should be discussed under the following two circumstances:

1. When units have a continunity

In this circumstance, the earlier transactions in the DAG shall be valid transactions.

2. When units have not a continunity

In this circumstance, similar to the case of a shadow chain, the valid transactions can be determined according to the serial number of the main chain, and the transactions with a smaller serial number shall be valid transactions. Because the units on a shadow chain have a relatively low mediation level, the shadow chain cannot be on the stable mainchain by the confirmation of the optimal parent unit.

Tamper resistence

Due to the particularity of the DAG storage structure, if the data of any unit is modified after it became to

a parent unit, the hash of it need to be changed accordingly, and all the child units that reference it will also need to be modified. Besides the selection of parent units in the DAG is random, so a perpetrator shall need more partners to work together when he/she wants to modify the DAG structure. At the same time, even if the perpetrators can immediately issue multiple units to reference the modified units, their goal will also not be achieved, because different units are authenticated by different members of Mediator, and the Meditor members will have to pay a higher price when they commit a perpetration, that their deposits shall be confiscated; and they will lose the qualification to be a member of Mediator to earn transaction fees.

UTXO (Unspent Transaction Output) and account synchronization

Comparison with Qtum

Qtum employs the bitcoin architecture as its underlying architecture, and added the following three operations into the Bitcoin scripting language to support EVM:

- OP_EXEC: Usually used to deploy new contracts;
- OP_EXEC_ASSIGN: Usually used to combine the token transfers.
- OP_TXHASH: Used by a contract ot transfers funds to other addresses

Of them, OP_EXEC_ASSIGN and OP_TXHASH are used to indicate the way vint is spent. OP_EXEC is used to transmit the data in vout to the EVM and help the EVM to execute the relevant codes, and update the outputs of a contract outputs at the ame time. Therefore, Qtum equipped itself with an Account Abstract Layer (AAL) as the communication layer between Bitcoin core and EVM.

PalletOne differs from Qtum chains in the following ways:

1. We only support the two Bitcoin standard scripts of P2PKH and P2SH, and features separation between execution of a contract execution of the script.

2. Since we use the "Unit" as data's storage structure, in the two unit structures of contract template deployment and invocation, token transfers, transaction fee payment and other operations are indicated by a single payment field.

3. In **PalletOne, there** is no separate list of contract outputs, and the contract-related tokens are also stored in UTXO.

4. PalletOne adds assetId and uniqueId into its UTXO and thus expands the Token representation range.

getBalance(address)

getBalance(address): The invocation interface of PalletOne Core, used to query the balance of an account information. The underlying operation of it is to query and sum the information about an address in the UTXO database.

Inputs token picking algorithm

By analyzing the tnputs token picking interface bool in the source code of Bitsincoin (bitcoin-master/wallet/wallet.cpp), CWallet::SelectCoins(...) and void CWallet::AvailableCoins(...) , it can be realized that Bitcoin's iinputs token picking strategy is the rule of First-In-First-Out (also used by **Qtum and** Byteball).

But the best inputs token picking algorithm is to find the UTXO that meets the following two standards in the fastest time:

1. Change is required as less as possible;

2. Tokens with a small nominal amount shall be preferred to be spent.

The purpose of these two standards is to maintain the stability of the UTXO database size, thereby reducing the time consumption of UTXO database operations. PalletOne basically follows the rule in, so its inputs token selection process is as follows:

1. Sort in ascending order all UTXOs of the user.

2. Find the UTXO in the sequence equal to the target inputs (have been set as the Target); if found, return; if not, go to step 3.

3. Sum n smallest UTXOs and stop when the sum (n of smallest UTXOs)>=Target. If the sum(n of smallest UTXOs) == Target, return; If sum(n of smallest UTXOs)>Target, proceed to step 4.

4. This step can be divided into the following small steps:

a) Remove the smallest UTXO from the sum combination and recalculate the sum(n of smallest UTXOs).

b) Return to step a when the sum<=Target; repeat the step if the sum>Target.

c) Perform step 5 when the number of loops exceeds the maxRound.

5. Use the combination cooresponding to the maxRound in step 4 as the inputs

Transaction process

Ordinary transaction process

The process of an ordinary transaction is shown in Figure 4.5.

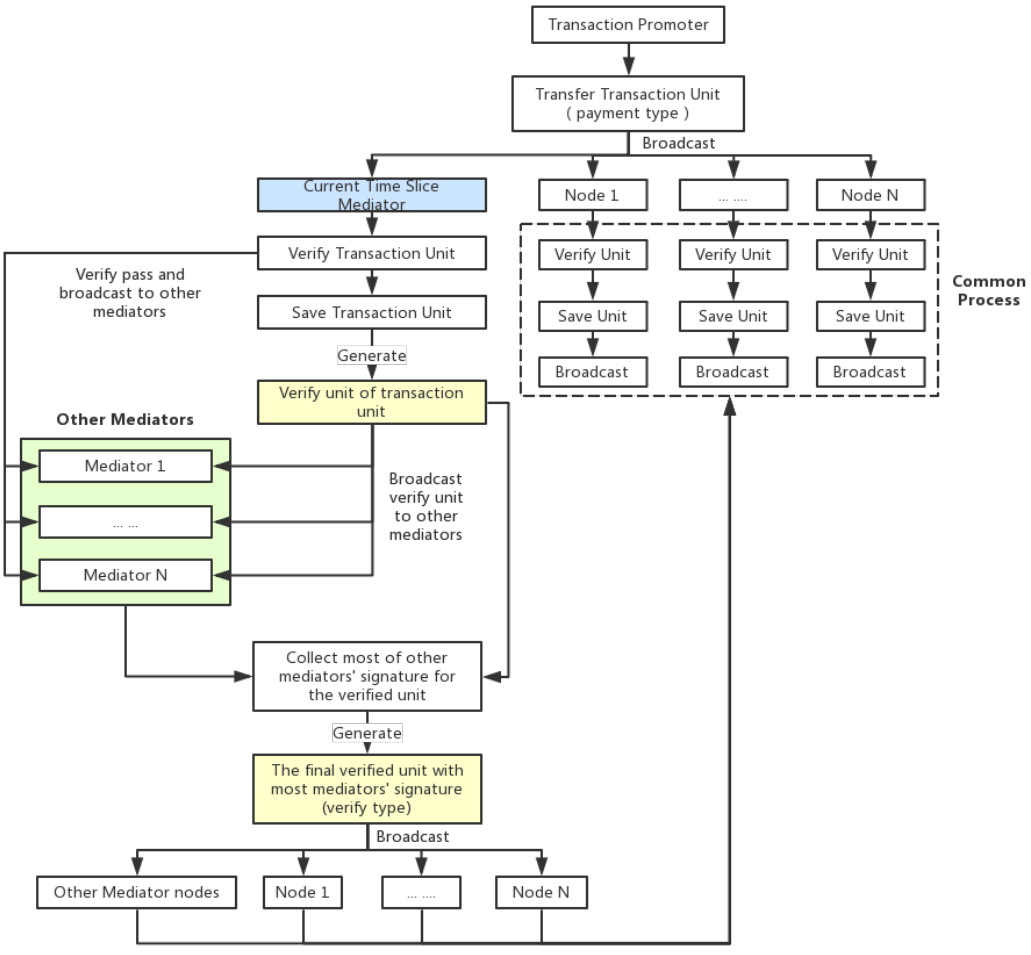

Figure 4.5 The process of an ordinary transaction

Contract creation process

The process of a contract creation is shown in Figure 4.5.

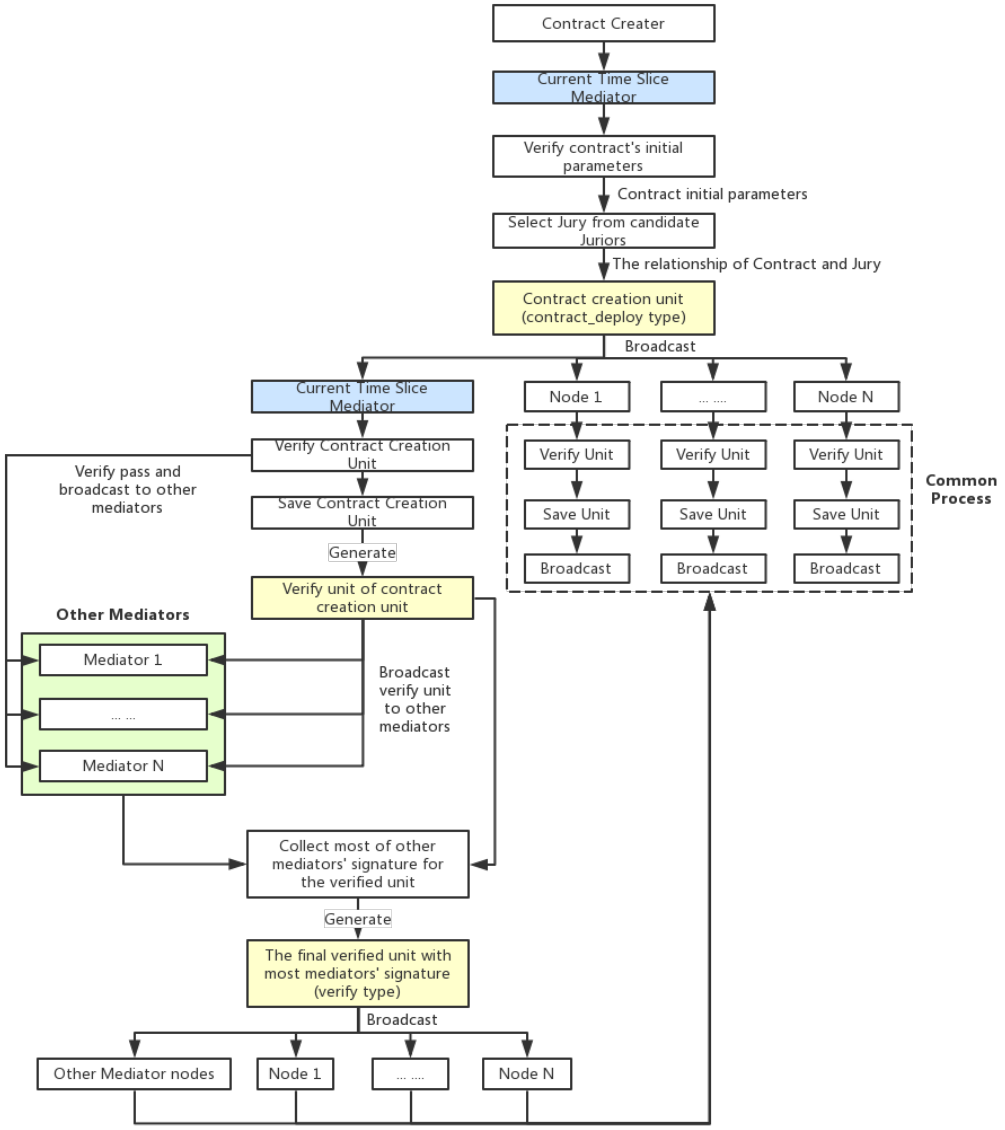

Figure 4.6 The process of a contract creation

Contract invocation process

The process of a contract invocation is shown in Figure 4.7.

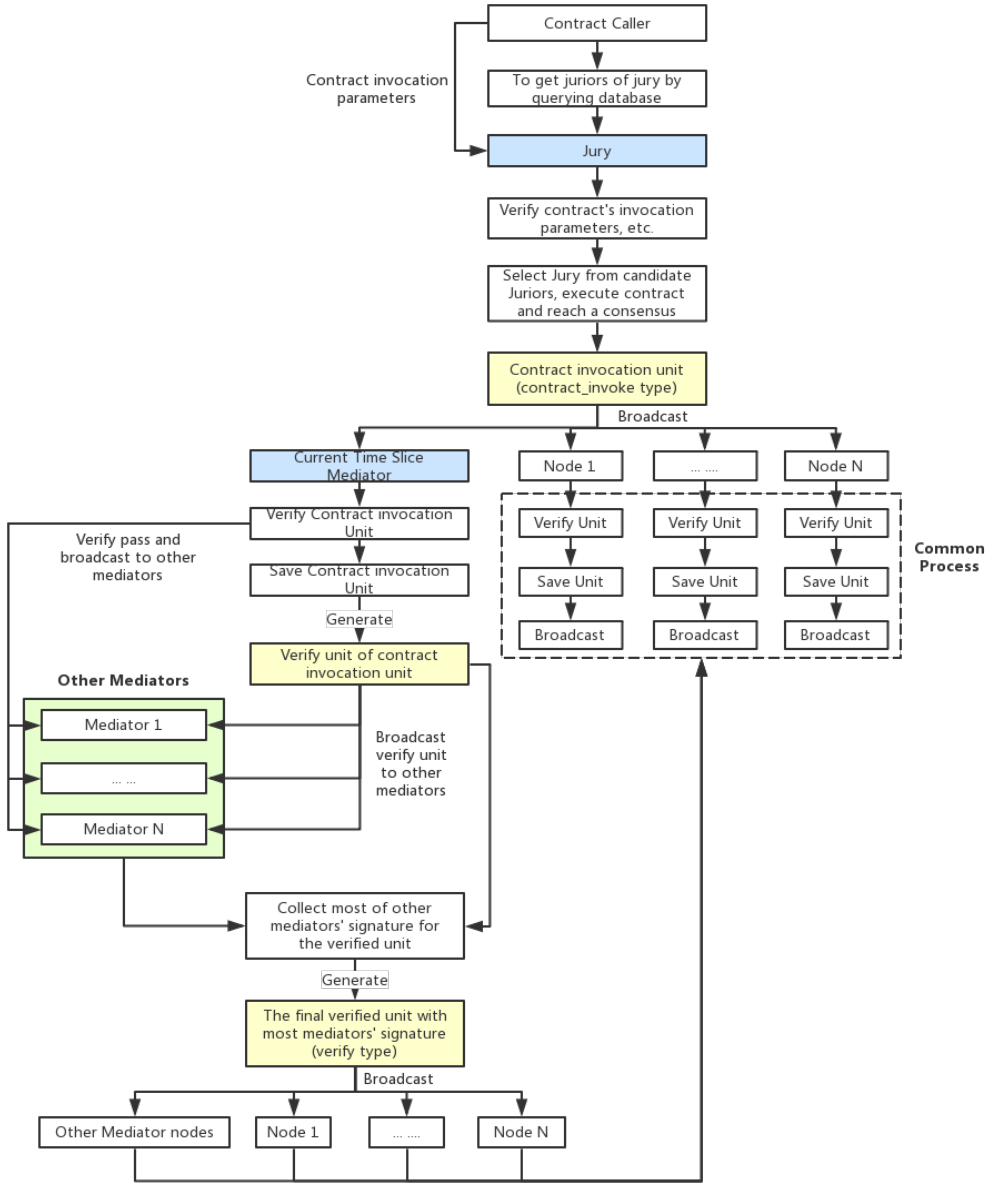

Figure 4.7 The process of a contract invocation

Contract termination process

The process of a contract termination process is shown in Figure 4.8.

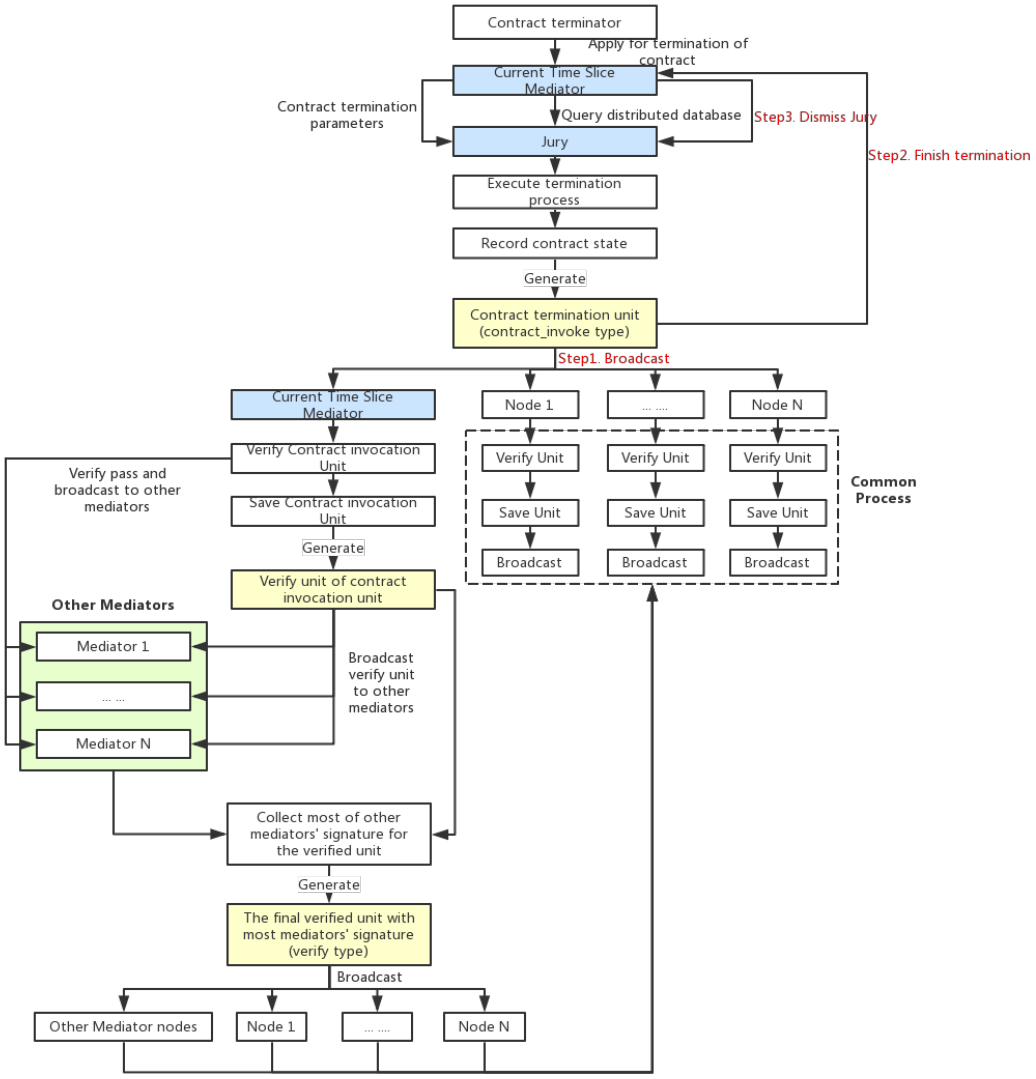

Figure 4.8 The process of a contract termination

Inter-chain transaction process

For examples of an inter-chain transaction, see the section “1. BTC and ETH Interchange (Jury Endorsement)”. The process of an inter-chain transaction is divided into the following steps:

1. Both parties of the transaction deploy a transaction contract template for the inter-chain contract, which is the contract deployment procedure of the transaction.

During it, after the jury is generated, it will generate a corresponding multi-signature account and return the account’s information to both parties.

2. The two parties of the transaction invoke the transfer interface of the contract to transfer t the corresponding amount of currency to the multi-signed account, which is a contract invocation procedure of the transaction.

3. The two parties of the transaction invoke the currency receipt request interface of the contract to file a currency receipt request, which is a contract invocation procedure of the transaction.

In this step, the jury needs to verify whether the second step has been completed. If completed, the jury will perform the fourth step.

4. The jury invokes the transfer interface to transfer the currency in the multi-signature account to the

corresponding accounts of counterparties, which is a contract invocation procedure of the transaction.

5. The two parties of the transaction initiate a request to terminate the transaction, which is the contract termination procedure of the transaction.

If the parties do not initiate the transaction termination request, the jury will do it after the timeout period has elapsed. The above process is shown in Figure 4.9. The procedures of contract deployment, contract invocation, and contract termination involved in the process are same with those shown in the “Contract creation process” and “Contract call process”.

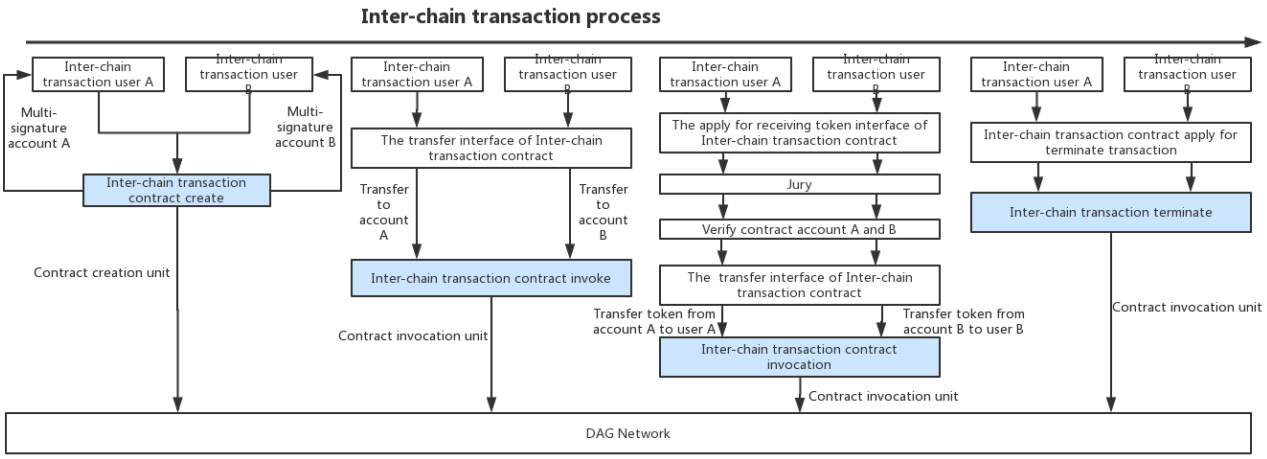

Figure 4.9 The process of an inter-chan transaction

Smart contract

Overview of PalletOne Smart Contract

The concept of Smart Contract, firstly put forward by Nick Szabo in 1996, is widely applied to blockchains, which lead to generation of many blockchains supporting a Turing-complete smart contract, such as Ethereum, Qtum and EOS. Generally speaking, a smart contract has the following characteristics:

（1） A smart contract must be a contract, an agreement between equal parties on performance of contents agreed upon by them. But it has a difference from a traditional contract, which is that it is in a digital form, and can only be read and executed by computers.

（2） A smart contract are partially or fully self-executed or self-enforced.

（3） A smart contract requires a safe operating environment, which must be safe and reliable and where multiple parties can reach consensuses on the results of it execution.

（4） Only with a smart contract, the ledger data can be altered.

In consideration of the characteristics, security and compatability with multiple languages and platforms of a smart contract, PalletOne shall employ the Docker containerization technology as the default means to realize a smart contract. In the industry, Docker, as a virtual contract virtual machine, has been applied into HyperLedger Fabric and has been demonstrated by a large number of enterprises and projects.

Docker containerization – Supporting multi-language smart contracts

The system provides different runtime Docker mirrors for different programming languages, as well as SDKs in the corresponding languages to help contract developers quickly and easily develop smart contracts. A contract developer can fulfill the development of a smart contract by tow steps, firstly compiling locally a contract program based on the SDK and secondly releasing the compiled file in PalletOne. The released contract program is a contract template. When a user creates a contract based on the template, the contract execution nodes (jurors) will select the corresponding runtime mirror based on the language of the contract, then build a new contract template mirror and finally instantiate the contract container, as shown in Figure 5.1.

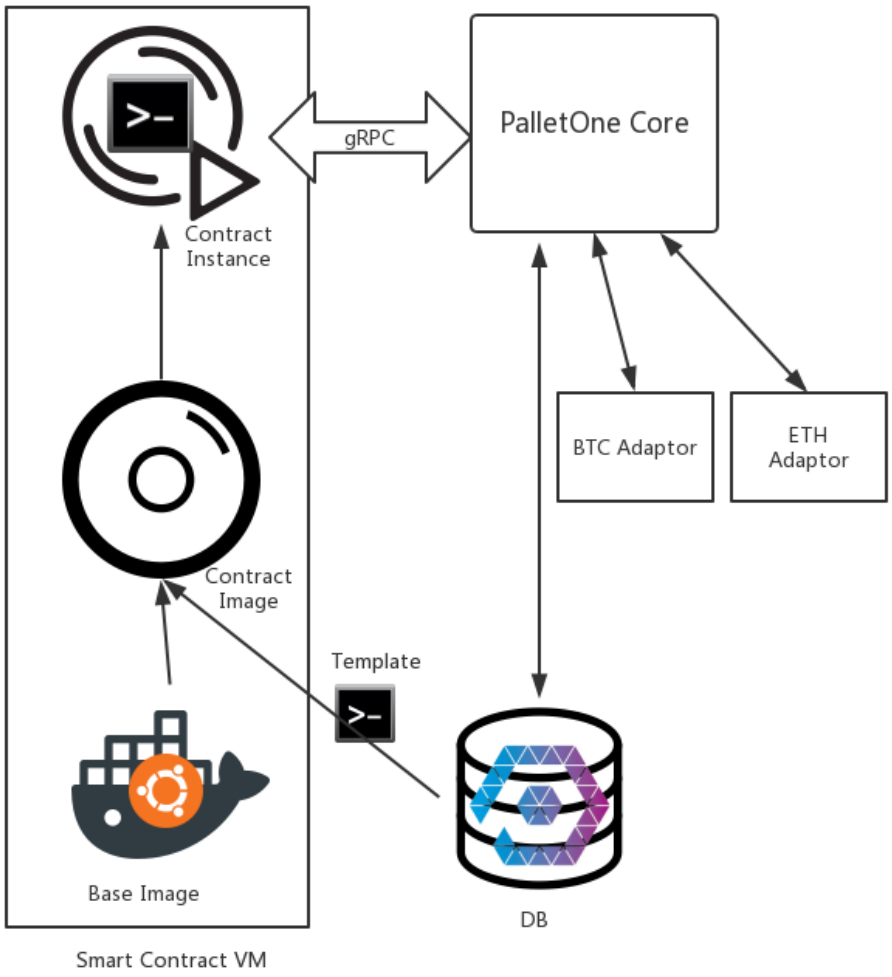

Figure 5.1 The process of a Docker-based contract creation

Contract security

The contract program shall run in a sandbox environment provided by Docker, which isolates the host's environment and network accesses and provides only limited CPU, memory and hard disk resources, avoiding the possibility of malicious contracts attacking the host.

In order to prevent the occurance in a contract of a large number of operations or infinite loops attacking the host, Ethereum employs the Gas mechanism. PalletOne shall also employ a similar penalty mechanism. Between them have a difference that PalletOne won't calculate in detail the Gas consumed by each command, because doing so will seriously affect the performance of the contract, and the pricing of each command is uncertain. In PalletOne, a user is required, when running a contract, to specify the amount of memory consumption, timeout of the contract execution and commission (PalletOne Token) he/she is willing to pay for the entire contract lifecycle. If the performance of the contract is not completed within the stipulated time, the jurors will terminate the execution and collect the token paid by the user.

The specific structure of the contract system is shown in Figure 5.2. In it, the contract manger is responsible for the unified management of contracts.

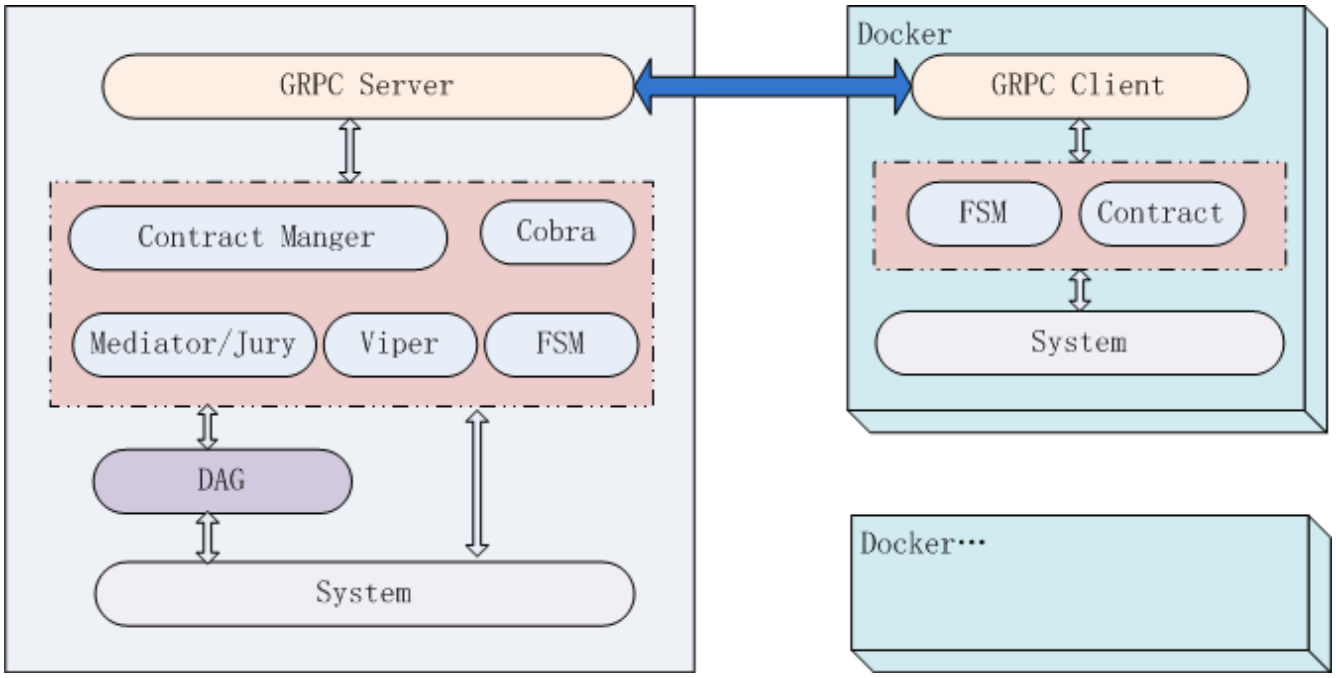

Figure 5.2 The structure of the contract system

Smart contract lifecycle

Template development

The template of a smart contract is an executable program that is based on a corresponding PalletOne contract SDK and conforms to the PalletOne contract specification. PalletOne supports at present smart contract developers to use their own familiar development language (PalletOne will provide support for C++/Go and Java in the early stage, and later for more languages such as NodeJS and C#) to locally write and compile contract template programs, and later will provide a simulated operation of the standalone PalletOne network for the sake of facilitating the development and debugging.

Template deployment

When deploying the template of a smart contract, the developer of it do not need to publicize the source code of it to the PalletOne network, but only need to deploy the compiled program to the PalletOne network. The steps for such deployment are as follows:

1. The contract developer signs the template program with his private key.

2. The contract developer combines the template program, template configuration files (including interface descriptions, usage fees, signatures, etc.) and the PalletOne Token required to be paid for the deployment into a Template Deployment Request (TDR: Template Deploy Request) and sends it to the PalletOne network.

3. The mediator in PalletOne will verify, after receiving the TDR, whether the template program matches the template interface specification and conforms to the policy of a PalletOne contract template.

4. After fulfilling the verification and thinking that the TDR is qualified, the Mediator will generate an ID for the template, sign the template and record the template program and its configurations into the distributed storage.

The sequence of a contract deployment is shown in Figure 5.3.

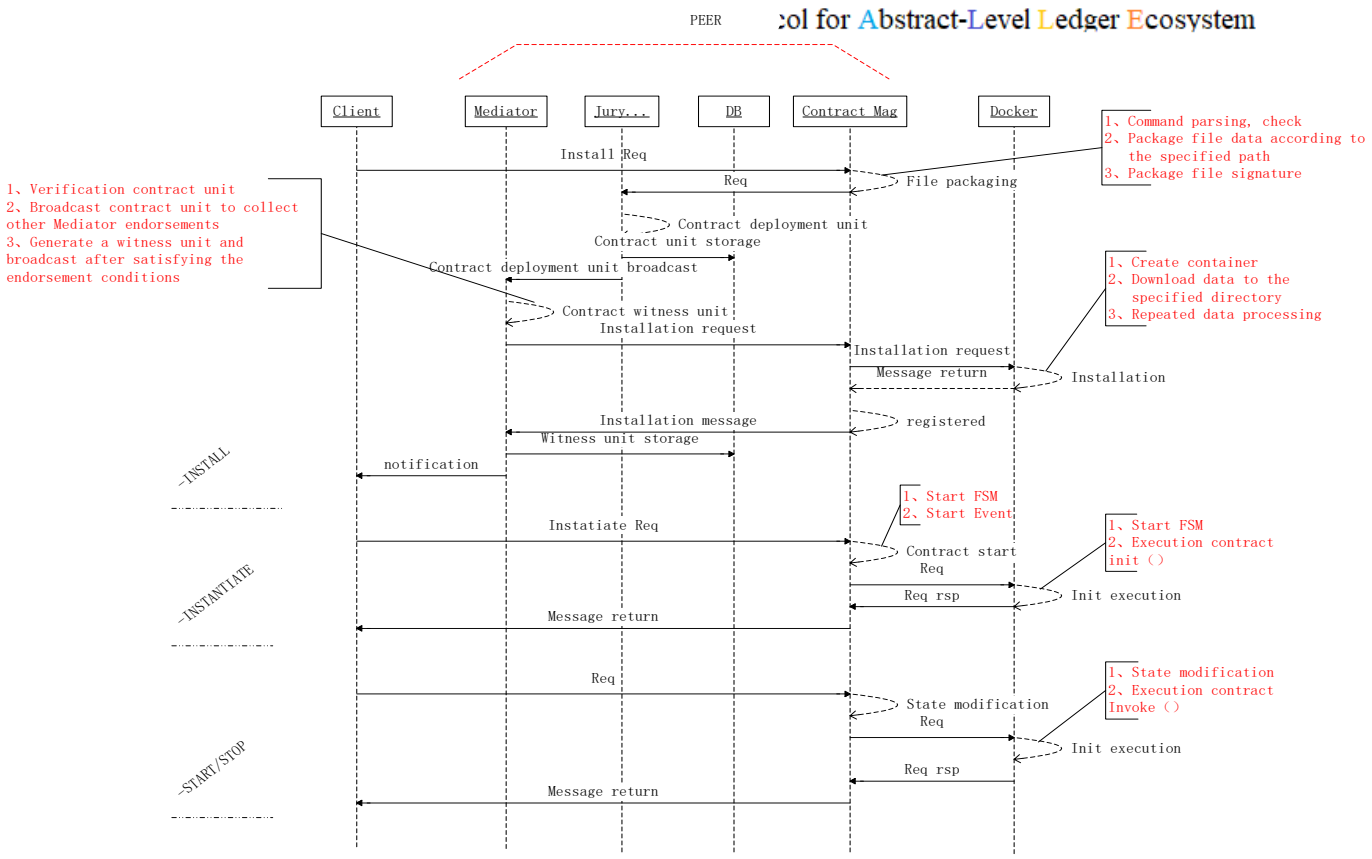

Figure 5.3 The sequence of a contract deployment

Contract creation/initialization

After being deployed to the PalletOne network, the template of a smart contract can be used by all users. In PalletOne, developers can publish fee-charge contract templates and free-charged ones, like publishing apps in an App Store. If a user wants to use a fee-charged contract template, he/she needs to pay a certain amount of PalletOne Token to its developer when initializing (instantiating) the contract. The amount is set by the developer.

A user can create a contract by the following steps:

1. The user files, to the PalletOne network at a client, a contract creation request (CCR: Contract Creation Request), which will include the contract template ID, initialization parameters (specific parameters vary according to different contract templates), transaction fee in PalletOne Token, template usage fee in PaleOne Token, signatures and others, and the hash value of it, namely the contract ID.

2. After receiving the CCR, the Mediator will inquire in the distributed storage the configuration parameters related to the contract template according to its ID, and determine, according to the configuration parameters, the mode of generation of the jury.

a) In the case of a locked mode of jury, the mediator will select the specified number of juror nodes will be selected to form the jury, associate them with the contract ID and recorded them into the distributed storage.

b) In the case of an unlocked mode of jury, just need to determine a list of jurors outnumbering the specified ones and record it into distributed storage.

3. The Mediator forwards the CCR to the jury.

4. Every member of the jury inquires, based on the received CCR, the corresponding contract program and its configurations in the distributed storage and finds out, based on the contract template ID, the corresponding mirror and container instance, and performs Steps 5 and 6 if they are not found.

5. Every member of the jury finds out, based on the contract template program language, the corresponding Docker base mirror and creates a new contract mirror by combining it with the contract program obtained from the distributed storage.

6. Every member of the jury creates an instance of Docker container based on the newly-created contract mirror.

7. Every member of the juty performs the initialization function in the contract and writes the result returned by the initialization function to the distributed storage.

Contract invocation

Through inquiring the distributed storage according to the ID of a smart contract, a user can see whether the contract is initialized or not, and which jurors are assigned to the contract; and can fine all the interfaces for the invocation of the contract according to the configuration files of the template of the contract. If the user wants to invoke the contract, he/she needs to create a contract call request (CIR: Contract Invoke Request), which will contain the following information: the contract ID, invoked function names, list of parameters, transaction fee, signature, etc.

If the contract featuring a locked mode of jury, the user can obtain the list of jurors corresponding to the contract from the distributed storage and then send the CIR to all of them. After obtaining the CIR, the jurors will communicate with each other and randomly elect the Leader. All the jurors will check the preconditions of the contract invocation (the preconditions for every operation of the contract have been defined in its template) and feed them back to the Leader. It is the Leader who can finally decide whether to actually execute the contract invocation, based on the feedbacks of the jurors and the policy of the contract execution. The sequence of the foregoing contract invocation is diagramed in Figure 5.4.

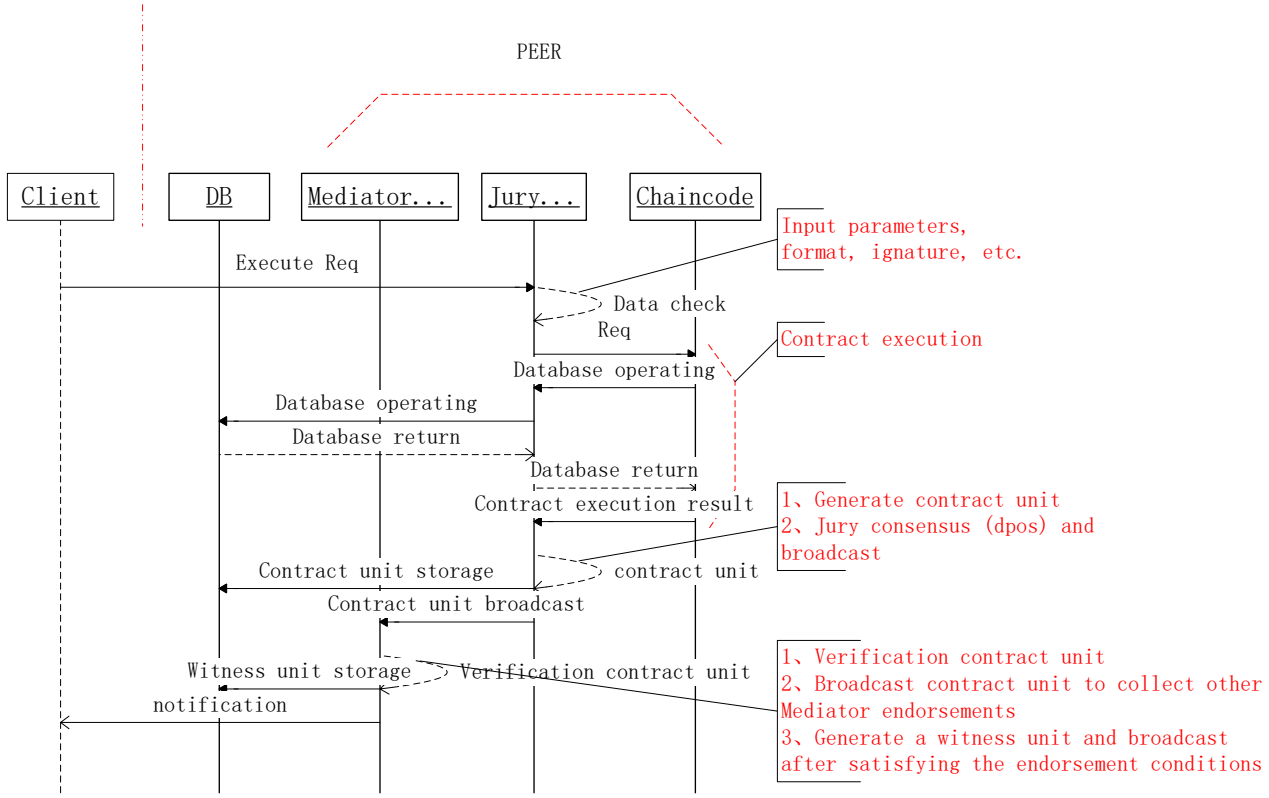

Figure 5.4 The sequence of a contract invocation

Contract upgrade

Unlike Ethereum where a smart contract is no longer allowed to be upgraded after being published, which makes DApps impossible to be fixed when they have a bug, PalletOne will, by introducing a mechanism of contract upgrade into itself, allow an request of upgrading a contract to be filed from the template of it when it is needed, and then to be made after the users of the contract voted to undertake it.

The process f a contract upgrade is as follows:

1. The developer of the template modifies and compiles the contract code, and then file to the PalletOne network a contract template upgrade request (TUR: Template Upgrade Request), which will include the original contract template ID, new contract program, description of new contract interfaces, upgrade instructions, contract instance upgrade policy (forced upgrade, optional upgrade, no upgrade), upgrade fee in PalletOne Token and signature.

2. After receiving the TUR, the Mediator will verify whether the template program matches the description of template interfaces and conforms to the policy of a PalletOne policy template.

3. After fulfilling the verification and thinking that the TUR is qualified, the Mediator will generate a template ID for and sign the template program and record it and its configurations into the distributed storage.

4. Users who are using the old version of the template will receive a prompt about the new version during the subsequent use of the contract. If it is a forced upgrade, the users shall not be allowed to continue using the old version. If an optional upgrade, the user can vote to decide whether the upgrade shall be accepted.

5. After receiving the users' confirmation or voting of the upgrade, the jury will retrieve the TUR from the distributed storage and upgrade the local contract template mirror and the contract container.

Contract termination

The request of terminating a contract can be filed after the contract is executed, or when both parties of the contract came to a consensus to to terminate it due to certain reasons. they may initiate a termination application. The process of a contract termination is as follows:

1. File to PalletOne a Contract Termination Request (CTR), which will include the following information: the contract ID, contract termination parameters and signature.
2. After receiving the CTR, the jury will perform a check for the contract termination. If the conditions for terminating a contract is met, the leader will change the contract state to terminated and record it into distributed storage.

The process of an interaction between a smart contract and PalletOne Core

After a smart contract is created, the jurors will create for the contract on their computers a Docker mirror and container where the contract program will be executed. When the contract is executed, it will communicate with the PalletOne Core of the Leader' s computer by using a gRPC, and then the PalletOne Core will communicate with other modules and nodes. Therefore, every contract instance will have a corresponding gRPC connection. The PalletOne Core will construct a context for the contract before invoking its container and then initiates the gRPC connection, and when the contract was executed and the execution result is returned to it, will be responsible to sign the result and then forward the signed result to the other jurors.

The SDKs that PalletOne provides for the developers of smart contracts are equipped with a number of interfaces that need to interoperate with PalletOne Core, including the following functions:

Functions to get invoked parameters

```
GetArgs() [][]byte
```

Get the input parameter lists as an array of byte arrays. GetStringArgs() []string: Get the input parameter lists as an array of strings.

```
GetFunctionAndParameters() (string, []string)
```

Divide the parameters of the string array into two parts in a way that the first word of the array is Function, and the rest are Parameter.

```
GetArgsSlice() ([]byte, error)
```

Get the argument lists as a byte slice.

Functions of operations of contract state data

```
GetState()
```

```
PutState()
```

```
DelState()
```

Functions of operations of PalletOne token

```
CreateAddress()
```

```
GetBalance()
```

```
GetTxHistory()
```

Functions of operations of external chians

```
GetOutChainBalance(string chainId, string address)
```

```
CreateOutChainTx(string chainId, Transaction tx)
```

```
InvokeOutChainFunc(string chainId, string function, string[] args)
```

All the above-imentioned functions are invocations by a contract of the PalletOne Core of the Leader's computer through the gRPC, where the PalletOne Core will judge, based on a specific function, whether it is a read or write operation of the distributed storage, or an invocation of a blockchain adapters.

Smart contract states

A smart contract itself does not store any data, because its logical processing is an interaction with the PalletOne Core of the Leader's computer through a good gRPC connection. The interaction is realized through a finite state machine (FSM). At both the PalletOne Core and the side of a smart contract defines all the states in their respective life cycles through the FSM, and how to respond to various events and transit to other states in various states. The state transition of a smart contract is shown in Figure 5.5.

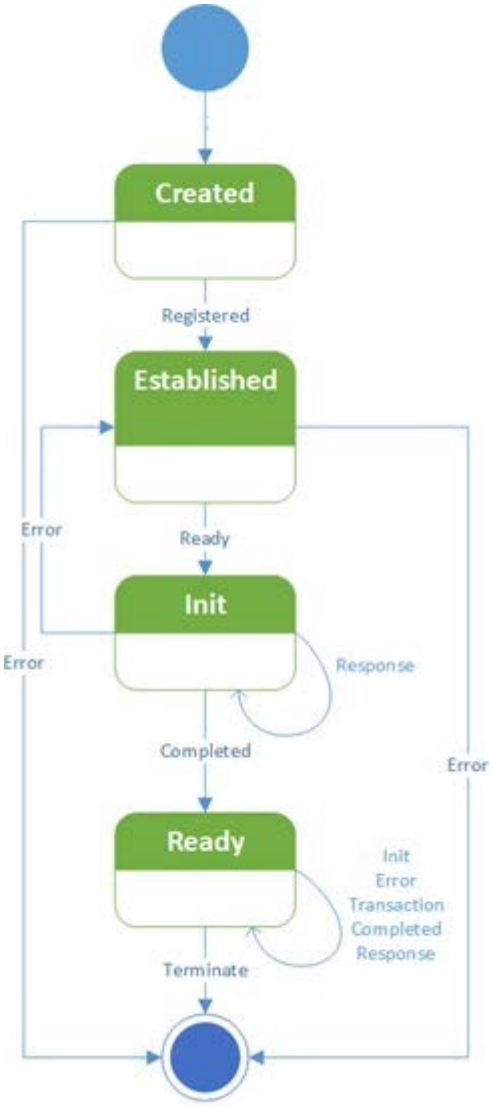

Figure 5.5 The state transition of a smart contract

The state transition of the PalletOne Core is shown in Figure 5.6.

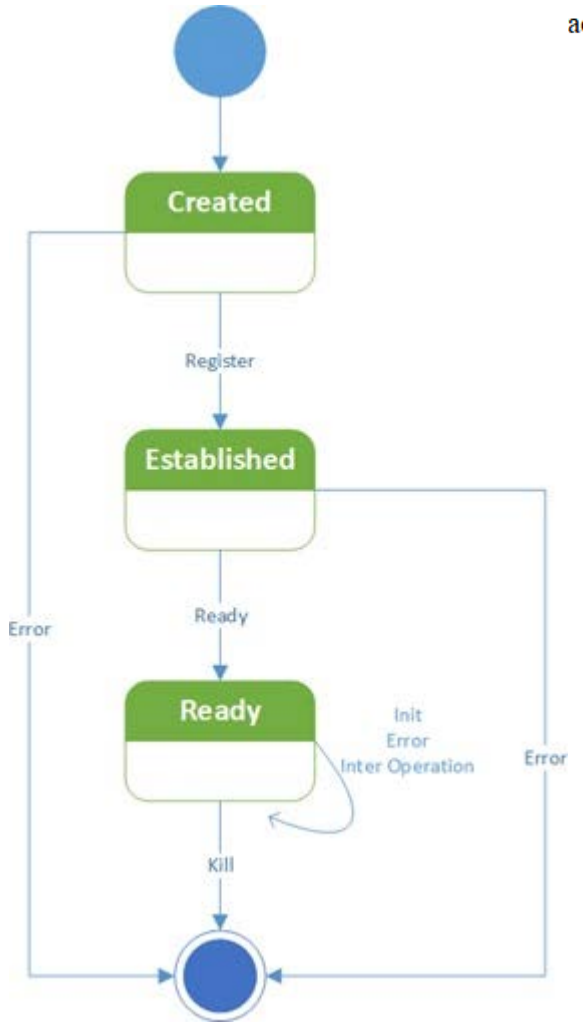

Figure 5.6 The state transition of the PalletOne Core

The adaption layer

Adapter is the communication medium for PalletOne to interact with different chains, and needs different operational logic to realize the adaption to different chains. Therefore, it is necessary to equip the adapter with interfaces corresponding to different chains to realize PalletOne's invocations of them.

After a comparative analysis of several mainstream chains, PalletOne defines these operations as the following four stages, as shown in Figure 6.1.

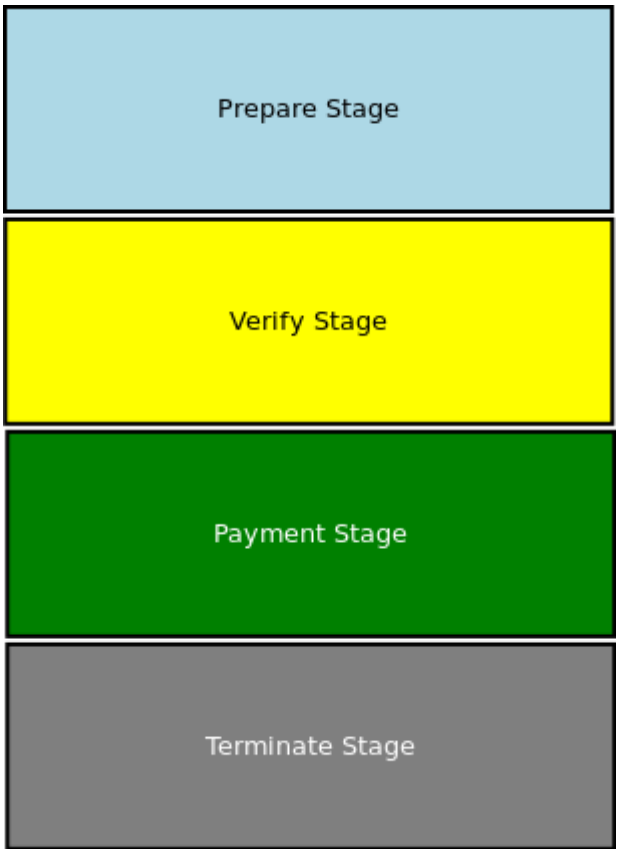

Figure 6.1 The stages of a PalletOne adaptor

1. Prepare Stage: This stage is used to create the corresponding preparation work. In the case of Bitcoin, this stage is to create a multi-signature address; in the case of Ethereum, it is a deployment contract.

2. Verify Stage: This stage is a hybrid stage that requires a verification mechanism for both a PalletOne contract and the adapter to interoperate. In the case of Bitcoin, this stage is to do multiple signatures; in the case of Ethereum, it is to verify the invocation of functions for a contract and for the notaries to verify/vote.

3. Payment Stage: This stage is the output stage of the corresponding chains of both parties, where, after the completion of the verify, the payer is requested to check the contract logic when he/she is invoking the contract and output the payment completion request according to the contract result.

4. Terminate Stage: This stage indicates that the contract has been completed or is invoked when the contract is voided due to a failure that both parties cannot reach a consensus. In the case of Ethereum, this is the action to invoke a contract termination.

Bitcoin adapter

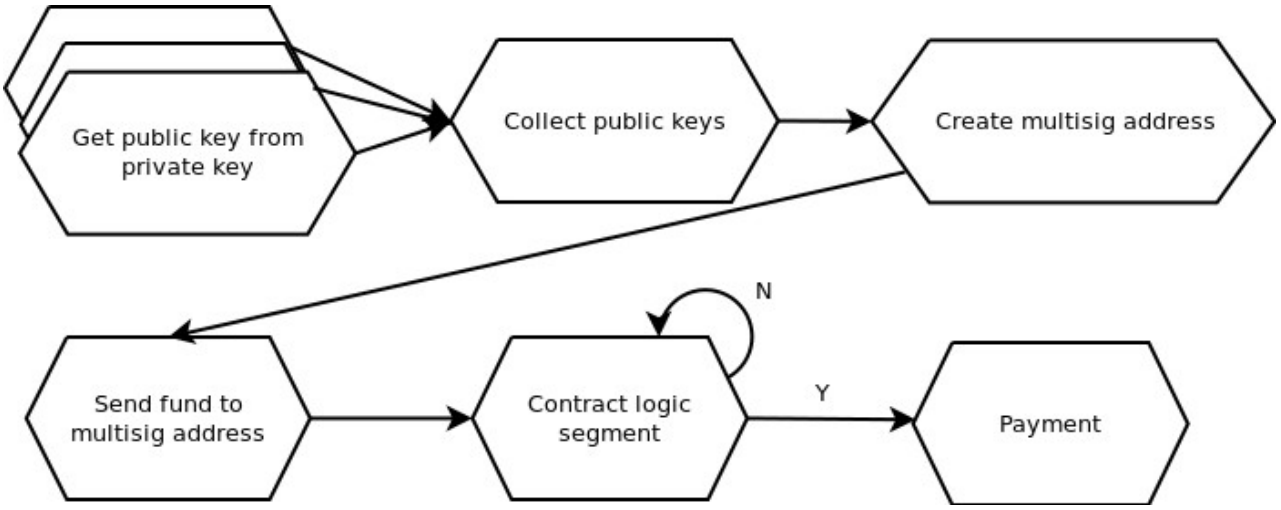

Figure 6.2 Bitcoin adapter

In PalletOne, the workflow of the Bitcoin Adapter is shown in Figure 6.2, which generally includes the following three steps:

1) A creator who initiates a bitcoin multi-signature address generates a multi-signature address after collecting the public keys of multiple parties, and then putting the fund into the multi-signature address;

2) The PalletOne contract logic determines, based on the result of running of the PalletOne contract, whether each of the jurors, who holds a private key, needs to perform a signature action, where the results of the contract logic can be a one-time payment or a multi-time payment;

3) If the PalletOne contract logic determines that a payment is required, it will return the sign transaction of jurors signatures and broadcast the transaction. After this, the payment will be completed..

Interfaces of the Bitcoin Adapter

1. GetPublicKey [get public key from private key]：

- To Use the private key operation to get a public key

```
input:
{
  "params":
    { "private_key":
"cUKo4q8MSsDGcTgvmrb8T3D2WDVYQ1DqA3DicnNQc5u9EyDVJ4rA",
    "address": "mrMUE2a5oFNzontQcSDBzGL4ZXdrTg1Sxh"
  }
}
output:
{
  {
    "data":
      { "public_key"
      :
"0351b9d44456a74c8912d3324e75e0600417a519db73112e6c52d21e542e624267",
      "address": "mrMUE2a5oFNzontQcSDBzGL4ZXdrTg1Sxh"
    }
  }
}
```

2. CreateMultiSigAddress [create mutiple signature address]

- To use jurors' public keys to generate a one-time multi-signature address

- Generate a n-m multi-signature addresses

```
input:
{
  "params": {
    "public_keys":
["04a882d414e478039cd5b52a92ffb13dd5e6bd4515497439dffd691a0f12af9575fa349b5
694ed3155b136f09e63975a1700c9f4d4df849323dac06cf3bd6458cd","046ce31db9bdd54
3e72fe3039a1f1c047dab87037c36a669ff90e28da1848f640de68c2fe913d363a51154a0c6
2d7adea1b822d05035077418267b1a1379790187","0411ffd36c70776538d079fbae117dc3
8effafb33304af83ce4894589747aee1ef992f63280567f52f5ba870678b4ab4ff6c8ea600b
d217870a8b4f1f09f3a8e83"],
    "n": 3,
    "m": 2
  }
}
output:
{
  "data": {
    "multisig_addr": {
      "p2sh_address": "347N1Thc213QqfYCz3PZkjoJpNv5b14kBd",
      "redeem_script":
"524104a882d414e478039cd5b52a92ffb13dd5e6bd4515497439dffd691a0f12af9575fa34
9b5694ed3155b136f09e63975a1700c9f4d4df849323dac06cf3bd6458cd41046ce31db9bdd
543e72fe3039a1f1c047dab87037c36a669ff90e28da1848f640de68c2fe913d363a51154a0
c62d7adea1b822d05035077418267b1a1379790187410411ffd36c70776538d079fbae117dc
38effafb33304af83ce4894589747aee1ef992f63280567f52f5ba870678b4ab4ff6c8ea600
bd217870a8b4f1f09f3a8e8353ae"
    }
  }
}
```

3. GetUnspendUTXO [To get the list of Utxos of a multi-signature addresses]

- minconf (numeric, optional, default=1), The minimum confirmations to filter
- maxconf (numeric, optional, default=9999999), The maximum confirmations to filter

```
input:
{
  "params": {
    "addresses": ["mkMQUVCtDmc8WVFURQ5ANuy9cRg7vJRQ4d",
"msJ2ktSgDaqs2jnytW1XUcR9WXiNK2pH1M"],
    "minconf": 0,
    "maxconf": 99999,
    "maximumCount": 2
  }
}
output:
{
  "data": {
    "utxo": [
      {
          "txid":
"129a5e3a8f746d332177e6559958251c304832b749df147117f232c1828a11f2",
          "vout": 1,
          "address": "mkMQUVCtDmc8WVFURQ5ANuy9cRg7vJRQ4d",
          "scriptPubKey":
"76a914350a52baeffa008dcd910af6e56e7b36e2d7d86f88ac",
          "amount": 49.89996200,
          "confirmations": 17,
          "spendable": true,
          "solvable": true,
          "safe": true
      },
      {
          "txid":
"6bf1b5672b633123ebbcd562f2e80ae90a8e3c71d3b3e35cdd84d5c4059101fb",
          "vout": 0,
          "address": "msJ2ktSgDaqs2jnytW1XUcR9WXiNK2pH1M",
          "scriptPubKey":
"21026be0e41fa7b42fa3b41c8f15e2f8fa2e2aedde35a6abf4f01d47f30be0b703ceac",
          "amount": 50.00000000,
          "confirmations": 106,
          "spendable": true,
          "solvable": true,
          "safe": true
      }
    ]
  }
}
```

4. RawTransactionGen [Raw transaction that generates transactions]

```
input
{
  "params": {
    "inputs": [
      {
        "txid":"id",
        "vout":n,
        "sequence":n,
      }
    ],
    "outputs": [
      {
        "address": 0.001,
        "data": "hex"
      }
    ],
    "locktime": 0,
    "replaceable": false
  }
}
output
{
  "data": {
    "rawtx":
"0200000001fb019105c4d584dd5ce3b3d3713c8e0ae90ae8f262d5bceb2331632b67b5f
16b00000000000ffffffff01000e27070000000017a914800bf99cf4d78e58e86513baa0c
4c79f7a4d70208700000000"
  }
}
```

5. DecodeRawTransaction [To show the detailed information of a raw transaction]

```
input
{
  "params": {
    "rawtx":
"0200000001fb019105c4d584dd5ce3b3d3713c8e0ae90ae8f262d5bceb2331632b67b5f16b
0000000000ffffffff01000e27070000000017a914800bf99cf4d78e58e86513baa0c4c79f7
a4d70208700000000"
  }
}
output
{
  "data": {
    "txid":
"44a081ba701f17eee823f1d1a5a27312dd692e347ad21a56ed80e45434ed9154",
    "hash":
"44a081ba701f17eee823f1d1a5a27312dd692e347ad21a56ed80e45434ed9154",
    "version": 2,
    "size": 83,
    "vsize": 83,
    "locktime": 0,
    "vin": [
      {
        "txid":
"6bf1b5672b633123ebbcd562f2e80ae90a8e3c71d3b3e35cdd84d5c4059101fb",
        "vout": 0,
        "scriptSig": {
          "asm": "",
          "hex": ""
        },
        "sequence": 4294967295
      }
    ],
    "vout": [
      {
        "value": 1.20000000,
        "n": 0,
        "scriptPubKey": {
          "asm": "OP_HASH160 800bf99cf4d78e58e86513baa0c4c79f7a4d7020
OP_EQUAL",
          "hex": "a914800bf99cf4d78e58e86513baa0c4c79f7a4d702087",
          "reqSigs": 1,
          "type": "scripthash",
          "addresses": [
            "2N4vGpsxXQvuKoExz8XVxYwvu5kX5CPk6zP"
          ]
        }
      }
    ]
  }
}
```

6. SignTransaction [Sign transaction]

```
input
{
  "params":
    { "hexstring
    ":
"0200000001fb019105c4d584dd5ce3b3d3713c8e0ae90ae8f262d5bceb2331632b67b5f
16b0000000000ffffffff01000e27070000000017a914800bf99cf4d78e58e86513baa0c
4c79f7a4d70208700000000",
    "privkeys": ["cT82eQM6ZLuRY8CS2rKZ9XhjvQ8fhCQbV6oTcai7E7WFYbx98amX"]
  }
}
output
{
  "data": {
    "hex":
"0200000001fb019105c4d584dd5ce3b3d3713c8e0ae90ae8f262d5bceb2331632b67b5f
16b00000000474630430220347 11d03a650dc1639736a28475799273039038a24aa9e16c
1bb060f555e2562021f31685371525d155e17a9fe9a008f59c377839d6a100bb876e2ada
d49df347201ffffffff01000e27070000000017a914800bf99cf4d78e58e86513baa0c4c
79f7a4d70208700000000",
    "complete": true
  }
}
```

7. GetBalance [To get the available balance of a multi-signature address]

- minconf (numeric, optional, default=1) The minimum confirmations to filter

```
input
{
  "params": {
    "address": "2N4vGpsxXQvuKoExz8XVxYwvu5kX5CPk6zP",
    "minconf": 3
  }
}
output
{
  "data": {
    "value": 1.2
  }
}
```

8. GetTransactions [To get the transactions related to a multi-signature address]

- skipping the first [from] transactions for account

```
input
{
  "params": {
    "account": "2N4vGpsxXQvuKoExz8XVxYwvu5kX5CPk6zP",
    "count": 10,
    "skip": 0
  }
}
```

9. ImportMultisig [To import a multi-signature address to a wallet observation address

- To scan a block and establish an index of information related to the address of the block.

```
input
{
  "params": {
    "account": "2N89aUC4JUjXyapvPvc1ivJVcfXpqBR4P2g",
    "rescan": true
  }
}
output
{
  "data":
    { "success": rue
  }
}
```

Ethereum adapter

Ethereum itself supports complex types of smart contracts. But here is just a sample to demonstrate how to realize the interfaces to a simplest Ethereum contract. For different types of Ethereum contracts, different types of external interfces are needed to be equiped to realize CalletOne contracts' invocation of them.

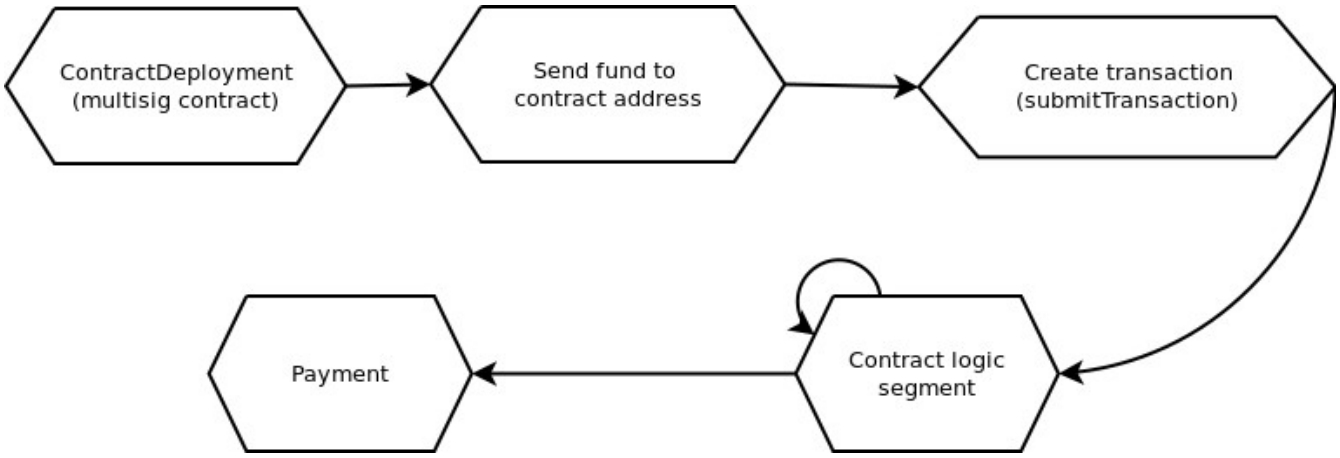

Figure 6.3 Ethereum adapter

PalletOne's Ethereum adapter works in the way shown in Figure 6.3, which can be roughly divided into the following three steps:

1. PalletOne invokes the Ethereum interactive contract to deploy a multi-signature transaction smart contract in solidity, and specify list of Ethereum jurors and the signature policy (the number of valid signatures) when deploying it.
2. After placing the fund in the address of the deployed Ethereum smart contract, the adapter can create a payment transaction contract within the contract, whose reasonableness will be verified by the Jury through the PalletOne contract logic. If the PalletOne contract logic determines the transaction valid, it will invoke the function of confirm Transaction to execute the Ethereum contract;
3. To execute the signature instruction on the transaction. When a transaction signature satisfies the signature policy, the function of execute Transaction can be invoked to complete the payment request on Ethereum.

Interfaces of the Ethereum adapter

1. GetBalance [To get the balance of an account]

```
input
{
  "params": {
    "account": "0xf947D41eA54953c212ecCaa4dF5c015821Ba8731"
  }
}
output
{
  "data":
    { "balance":
    1.0042
  }
}
```

2. GetEthereumTransactionByHash [To get the information of a transaction by hash]

```
input
{
  "params": {
    "hash":
"0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"
  }
}
output
{
  "data": {
    "id":1,
    "jsonrpc":"2.0",
    "result": {

"hash":"0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d105
5b",
      "nonce":"0x",
      "blockHash":
"0xbeab0aa2411b7ab17f30a99d3cb9c6ef2fc5426d6ad6fd9e2a26a6aed1d1055b",
      "blockNumber": "0x15df",
      "transactionIndex":  "0x1",
      "from":"0x407d73d8a49eeb85d32cf465507dd71d507100c1",
      "to":"0x85h43d8a49eeb85d32cf465507dd71d507100c1",
      "value":"0x7f110",
      "gas": "0x7f110",
      "gasPrice":"0x09184e72a000",

"input":"0x603880600c6000396000f300603880600c6000396000f3603880600c60003
96000f360",
    }
  }
}
```

3. ContractDeployment [To deploy the template of a contract]

- address：The address of the reviewer required by a Multi-signature transaction
- num_of_confirm: A setup that a transaction can be executed with signatures of several persons;

```
input
{
    "params": {
        "address": ["0x627306090abaB3A6e1400e9345bC60c78a8BEf57",
"0xf17f52151EbEF6C7334FAD080c5704D77216b732"],
        "num_of_confirm": 2
    }
}
output
{
    "data": {
        "contract_address": "0x8f0483125fcb9aaaefa9209d8e9d7b9c8b9fb90f"
    }
}
```

4. submitTransaction

```
input
{
  "params": {
    "address": "0xf17f52151EbEF6C7334FAD080c5704D77216b732",
    "value": 1000,
    "bytes": "data"
  }
}
output
{
  "data": {
    "status": "ok"
  }
}
```

5. getTransactionIds [To get a list of the IDs of transactions waiting for confirmation signatures]

```
input
{
  "params": {
    "from": 0,
    "to": 10,
    "pending": true,
    "executed": false
  }
}
output
{
  "data": {
    "ids": [1,2,3]
  }
}
```

6. getTransactionById [To get the contents of a transaction by its ID]

```
input
{
  "params": {
    "id": 1
  }
}
output
{
  "data": {
    "destination": "0xf17f52151EbEF6C7334FAD080c5704D77216b732",
    "value": 1000,
    "bytes": "data",
    "bool": false
  }
}
```

7. getConfirmations [A transacton to get the confirmation signature of a transaction]

```
input
{
  "params": {
    "id": 1
  }
}
output
{
  "data": {
    "address": ["0xf17f52151EbEF6C7334FAD080c5704D77216b732"]
  }
}
```

8. confirmTransaction [To sign a transaction]

```
input
{
  "params": {
    "id": 1
  }
}
output
{
  "data": {
    "status": "ok"
  }
}
```

9. executeTransaction [To execute a transaction]

● (Anyone can trigger the execution of a transaction), the transaction will be executed when the confirmation signature is sufficient

```
input
{
  "params": {
    "id": 1
  }
}
output
{
  "data": {
    "status": "ok"
  }
}
```

Multi-signature contract template

Ethereum does not support the Bitcoint-lke native multi-signature. Its muti-signature is realized through a smart contract. The following is an open source contract source code for making a multi-signature transaction in Ethereum. Based on the template, PalletOne will make further optimizations to meet its needs for multi-signature transactions in Ethereum.

https://github.com/gnosis/MultiSigWallet

Examples

Exchange between BTC and ETH (with Jury endorsement)

With the locked mode of Jury

When Alice wants to change 1BTC for 16 ETHs, Bob also wants to do the reverse. But they don't trust each other on network. So they decide to make the exchange through a BTC-ETH exchange contract on PalletOne.

The specific operation process of the exchange is as follows:

1. Alice finds, in the contract market of PalletOne, the “BTC-ETH exchange contract”, fills in the number of BTCs she wants to exchange, namely 1, the number of ETHs she wants to exchange for, namely16, the number of default deposit, namely10, and the number of contract timeout, namely 24, and signs to create the contract, and then sends the contract to Bob.
2. After receiving the contract and thinking that it is right, Bob will sign the contract. At this time, the contract is formally activated, and both parties were requested to provide respectively 4 public keys for each of the coins and a contract deposit.
3. Alice generates 4 pairs of BTC public and private keys and 4 pairs of ETH public and private keys in her wallet, and sends the public keys and 10 PalletOne tokens to the contract.
4. Bob does the same as in Step3.
5. After receiving the public keys sent by Alice and Bob, the contract will generate a BTC 7/12 multi-signature address and an ETH7/12 multi-signature address based on the public keys from Alice and Bob and the public keys of the 4 jurors, and send them to Alice and Bob.

The above is the preparation and signing and receiving stage of the contract. The following is the execution process of it, which needs to be discussed in the following two circumstances:

- That the contract is completed normally
- That one party performs the contract while the other defaults on it

The following describes the logic of executing the contract in these two circumstances:

The circumstance that the exchange contract is completed normally

After generating the public keys and paying the contract deposit, Alice and Bob will receive 1 BTC multi-signature address and 1 ETH multi-signature address. Then, both parties and the system will perform operations as follows:

1. Alice transfers 1BTC to the BTC multi-signature address generated by the contract.
2. Bob transfers 16 ETHs to the ETH multi-signature address generated by the contract.
3. Bob checks the BTC address and finds that Alice has transferred 1BTC into it, so he files a BTC receipt request and invokes a function of transaction to tansfer the 1 BTC to his wallet, and signs it with his own 4 BTC private keys.

After receiving the BTC receipt request from Bob, the jury will check the ETH multi-signature address and finds that Bob has already transferred 16 ETHs to it, and does the same to the BTC address and finds that Alice has also transferred 1BTC to it. At this time, the conditions of the exchange have been satisfied.

4. After verifying the satisfaction, each juror will sign, with his or her private key, the Transaction invoked by Bob for the receipt of the 1 BTC.
5. When the three jurors of the jury complete the signing, the 7/12 multi-signature condition is met, and the 1 BTC will be transferred to Bob's wallet.
6. After learning that Bob has transferred 16 ETHs to the ETH multi-signature address, Alice will do the same as Bob does in Step 3, to transfer the 16 ETHs to her wallet. And the jury will do the same as above.

7. After receiving the 16 ETHs and also seeing that Bob has received the 1 BTC, she can file a contract termination request to the jury.

8. After checking the two multi-signature addresses and ensuring that the coins had been exchanged, the members of the jury can transfer the deposits of the two parties, 10 PalletOne tokens by each, to their respective wallets, and change the state of the contract to being terminated. At this time, the contract was executed.

The above steps are diagramed as in Figure 7.1.

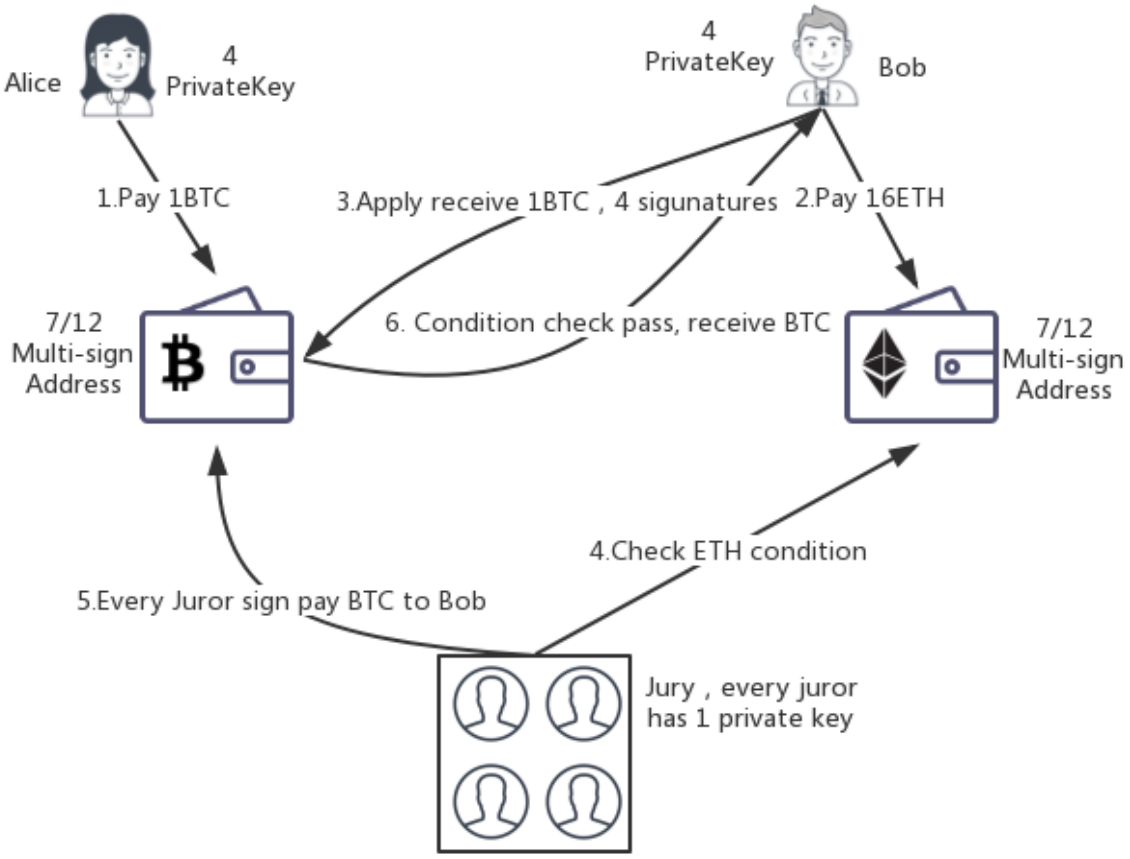

Figure 7.1 The normal process of executing the exchange contract

The circumstance that one party performs the exchange contract while the other defaults on it

The real world is always full of uncertainties, such as changes in the prices of currencies, which will lead one of the parties to feel that the previously signed contract will incur a loss on him/her. At this time, the party will be subjectively unwilling to continue to execute the exchange contract, while the other party has already transferred the agreed coin amount to one of the multi-signature address. Under such circumstance, the party performing the contract may suffer a loss. Therefore, there must be a deposit in the exchange contract to ensure that the loss suffered by the party performing the contract is compensated. The specific operation of it shall be made as follows:

1. Alice transfers 1BTC to the BTC multi-signare address generated by the contract.

2. Bob repents and is reluctant to transfer 16 ETHs to the ETH multi-signature address.

3. After 24 hours, Bob still does not transfer the ETH. Therefore, Alice will file a request to the jury to demand it to adjudicate Bob default on the contract and to return the 1 BTC, and sign it with her

own 4 private keys.

4. After checking the BTC and ETH multi-signature addresses, verifying that Alice has transferred the coin, but Bob has not, and finding at the same time that the time have exceeded the timeout specified in the contract, the jury will adjudicate Bob default on the contract.
5. The 1 BTC is returned to Alice's wallet when 3 or more jurors signed the function of Tansaction invoked by Alice, which means that the 7/12 multi-signature condition is met.
6. The total of 20 PalletOne Tokens which were paid by Alice and Bob as the deposits to ensure a fulfillmen of the contract, to Alice's wallet as a compensation for the loss caused by Bob's default to her.
7. The jury changes the contract state to being terminated. The contract was executed

Exchange between BTC and ETH (with Mediator endorsement)

In the locked mode of jury mentioned above, the number of jurors can only be up to five due to the multi-signature restriction of BTC, and it is very troublesome for the users of Alice and Bob to hold multiple private keys for signing, so we introduce another exchange mode: Mediator endorsement mode. The advantage of the Mediator endorsement mode is that it has more online nodes than the Jury's, which can guarantee the persistence of a multi-signature wallet. This mode is more suitable for long-duration inter-chain contracts that require a multi-signature.

Here we still take the above-mentioned same example as the background: when Alice wants to change 1BTC for 16 ETHs, Bob also wants to do the reverse. but they don't trust each other on network; so they decide to make the exchange through a BTC-ETH exchange contract on PalletOne.

The specific operation process is as follows:

1. Alice finds, in the contract market of PalletOne, the "BTC-ETH exchange contract", fills in the number of BTCs she wants to exchange, namely 1, the number of ETHs she wants to exchange for, namely16, the number of default deposit, namely10, and the number of contract timeout, namely 24, and signs to create the contract, and then sends the contract to Bob.
2. After receiving the contract and thinking that it is right, Bob will sign the contract. At this time, the contract is formally activated, and both parties were requested to provide respectively 1 public key for each of the coins and a contract deposit.
3. Alice generates a pair of BTC public and private keys and a pair of ETH public and private keys in her wallet and sends the public keys and 10 PalletOne Tokens to the contract.
4. Bob will do the same as in Step3.
5. After receiving the public keys sent by Alice and Bob, the contract will inquire the the current votes of the Mediator, select the first 10 nodes and find out their public keys to generate a BTC 7/12 multi-signature address and an ETH7/12 multi-signature address, and sends them to Alice and Bob.

The above is the preparation and signing and receiving stage of the contract. The following is the execution process of it, which needs to be discussed in the following two circumstances:

- That the contract is completed normally
- That one party performs the contract while the other defaults on it

The following describes respectively the logic of executing the contract in these two circumstances:

The circumstance that the exchange contract is completed normally

After generating the public keys and paying the contract deposit, Alice and Bob will receive 1 BTC multi-signature address and 1 ETH multi-signature address. Then, both parties and the system will perform operations as follows:

1. Alice transfers 1BTC to the BTC multi-signature address generated by the contract.

2. Bob transfers 16 ETHs to the ETH multi-signature address generated by the contract

3. Bob checks the BTC address and finds that Alice has transferred 1BTC into it, so he files a BTC receipt request and invokes a function of Transaction to tansfer the 1 BTC to his wallet, and signs it with his own 1 BTC private keys.

4. After receiving the BTC receipt request from Bob, the jury will check the ETH multi-signature address and finds that Bob has already transferred 16 ETHs to it, and does the same to the BTC multi-signature address and finds that Alice has also transferred 1BTC to the. At this time, the conditions of the exchange have been satisfied; and the jury will perform an operation to reach a consensus on the result of the contract execution;

5. When more than 2/3 of the jurors in the jury signed the BTC Transaction, the threshold of signature for the result of a contract execution is satisfied. Under such situation, the jury leader will send the execution result and the function of BTC Transaction to the corresponding nodes of Mediator.

6. After receiving the request from the jury leader, the Mediator will verify the signatures of the request and the Redeem Script of the BTC multi-signature. If the verification is successful, the BTC Transaction will be signed and returned to the jury's Leader.

7. When the jury leader receives 6 or more Mediator signatures, he/she will broadcast, to to the BTC network, that the 1 BTC in the multi-signature address will be transferred to Bob.

8. After receiving the function of Transaction that meets the requirement of multi-signature, it will package the transaction, and Bob's wallet will receive the 1 BTC.

9. After learning that Bob has transferred 16 ETHs to the ETH multi-signature address, Alice will do the same as Bob does in Step 3, to transfer the 16 ETHs to her wallet. And the other steps are made as above.

10. After receiving the 16 ETHs and also seeing that Bob has received the 1 BTC, Alice can file a contract termination request to the jury.

11. After checking the two multi-signature addresses and ensuring that the coins had been exchanged, the members of the jury can transfer the deposits of the two parties, 10 PalletOne tokens by each, to their respective wallets, and change the state of the contract to being terminated. At this time, the contract was executed.

The above steps are diagramed as in Figure 7.2.

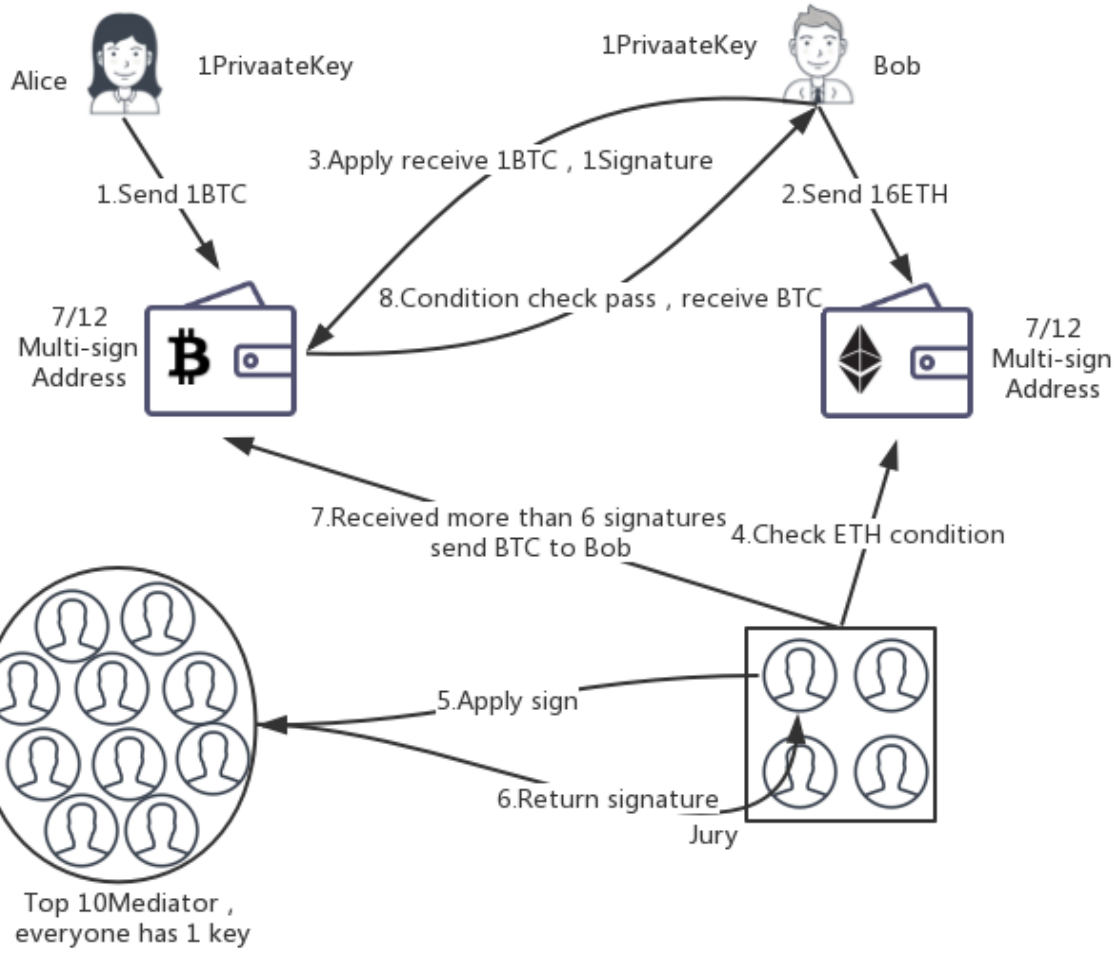

Figure 7.2 The normal process of executing the exchange contract

The circumstance that one party performs the exchange contract while the other defaults on it

The real world is always full of uncertainties, such as changes in the prices of currencies, which will lead one of the parties to feel that the previously signed contract will incur a loss on him/her. At this time, the party will be subjectively unwilling to continue to execute the exchange contract, while the other party has already transferred the agreed coin amount to one of the multi-signature address. Under such circumstance, the party performing the contract may suffer a loss. Therefore, there must be a deposit in the exchange contract to ensure that the loss suffered by the party performing the contract is compensated. The specific operation of it shall be made as follows:

1. Alice transfers 1BTC to the BTC multi-signare address generated by the contract.
2. Bob repents and is reluctant to transfer 16 ETHs to the ETH multi-signature address.
3. After 24 hours, Bob still does not transfer the ETH. Therefore, Alice will file a request to the jury to demand it to adjudicate Bob default on the contract and to return the 1 BTC, and sign it with her own 1 private key.
4. After checking the BTC and ETH multi-signature addresses and verifying that Alice has transferred the coin, but Bob has not, and finding at the same time that the time have exceeded the timeout specified in the contract, the jury will adjudicate Bob default on the contract.
5. When more than 2/3 of the jurors signed the Alice's request, the Leader will send the Transaction of the request to the corresponding nodes of Mediator.

6. After verifying the Transaction from the Leader, the Mediator will sign the tansaction and return it to the Leader.

7. When 6 or more jurors signed the Tansaction, which means that the condition of muti-signature is met, the 1 BTC will be transferred to Alice's wallet.

8. The total of 20 PalletOne tokens which were paid by Alice and Bob as the deposits to ensure a fulfillmen of the contract, to Alice's wallet as a compensation for the loss caused by Bob's default to her.

9. The jury changes the contract state to being terminated. The contract was executed.

Purchase a game currency with BTC and ETH

A game studio released a game currency on PalletOne, with which users can purchase value-added services such as special equipment and accelerators. The game currency is priced in the ways of 1 BTC=1600 GTokens (Game Token) and 1ETH=100 GTokens. Therefore, users can choose BTC or ETH to purchase it.

1. The Game Studio releases, based on the homogenization token transaction template in PalletOne, 1 billion GTokens at a time to represent the game currency in the game.

2. The game studio creates a BTC and ETH based token purchase contract on PalletOne, specifies an address and an exchange rate for the BTC purchase, which is 1:1600, and the same for the ETH purchase, which is 1:100. The circulated token shall be the Gtoken created in Step 1.

3. The game studio transfers the 1 billion GTokens to the address of the contract. From now on, the contract will be responsible for allocating the tokens.

The above is the preparation stage of the studio. The following describes the process that a user purchases the game currency, which is shown in Figure 7.3:

1. Alice owns a BTC and wants to buy 1600 GTokens with it, so she files a request for purchasing the tokens in PalletOne and fills in the number of tokens she wants to purchase, namely 1600, the payment means, namely by BTC, and her own BTC wallet address.

2. The contract will, based on the price of the token, generate an order and inform Alice of the ID of the order and that she needs to transfer 1 BTC, and where to pay it.

3. Alice uses her own BTC wallet to transfer 1 BTC to the BTC payment address. If it is a wallet that supports leaving a remark for a transfer, she can fill in the order number.

4. After transferring the 1 BTC, Alice can file, to the contract, an request for receipt of 1600 GTokens, and fill in the IDs of her order and transfer of 1BTC.

5. After receiving the request from Alice, the jury of the contract will check the BTC transaction record and GToken payment record to ensure that the BTC is received and the token has not yet been paid.

6. The contract checks whether Alice's BTC wallet address matches the one she input during the transaction, to ensure that Alice is not taking someone else's payment record.

7. The contract transfers 1,600 GTokens to Alice's wallet.

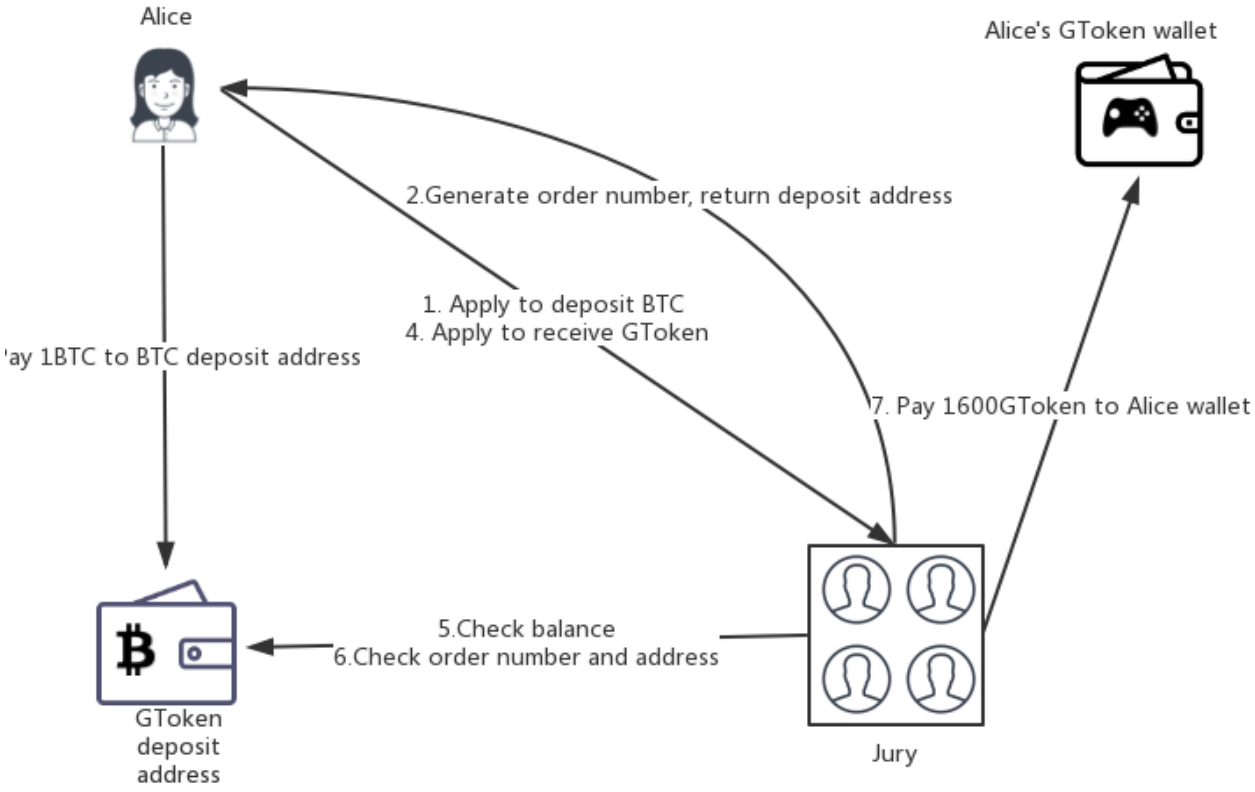

Figure 7.3 The normal process of user purchasing game coin

When Bob owns a ETH and wants to use it to purchase corresponding GTokens, he should do the process same to the above, except that the address, whose balance and transaction record shall be checked by the contract in the process, is the one for the ETH purchase.

Since the above contract does not involve a multi-signature addresses, there is no need to lock a jury, that is, the execution of each transaction is not fulfilled by fixed jury members.

PalletOne Token Exchange

This is a PalletOne- based token exchange contract. Unlike the BTC-ETH exchange contract, the contract hereof refers to the exchange between PalletOne's Token (simply written as PToken) and a Token issued by a user on PalletOne (assume its name is UToken).

Alice owns a certain amount of PTokens and wants to exchange 10 PTokens for 100 UTokens in Bob's hands. They do the following steps:

1. Alice finds a “PalletOne Token Echange” contract in the contract market of PalletOne the fills the information that one party of the exchange is Pokeken, whose quatity is 10 and the other is UToken, whose quantity is 100; the default deposit is 1 PToken; and the contract timeout is 24 hours, and sign to create it. Then send it to Bob.
2. After receiving the contract and verifying that it is right, Bob will sign the contract. At this time, the contract is formally activated, and both parties were requested to provide respectively the agreed amount of default deposit.
3. Alice sends 1 PToken of her wallet to the contract.
4. Bod does the same as above.

The above is the preparation and signing and receiving stage of the contract. The following is the

execution process of it, which needs to be discussed in the following two circumstances:

- That the contract is completed normally
- That one party performs the contract while the other defaults on it

The following describes respectively the logic of executing the contract in these two circumstances:

The circumstance that the exchange contract is completed normally

Alice and Bob can make the exchange transactions after both of them have paid the default deposit. Alice transfers 10 PTokens to the address of the contract, and Bob will, after seeing that the address has received 10 PTokens, directly transfer 100 UTokens to Alice's wallet.

After confirming that the 100 UTokens is transferred successfully, Bob will file to the contract a request of token receipt, attach the TxId of the transfer of 100 UToken to it, and sign it with his private key.

Then the jurors will check the PalletOne's network based on the TxId, and if verify that Bob did transfer 100 UTokens to Alice, which means the condition of transfer is satisfied, will sign the Transaction of token receipt invoked by Bob. The above steps are diagramed as in Figure 7.4.

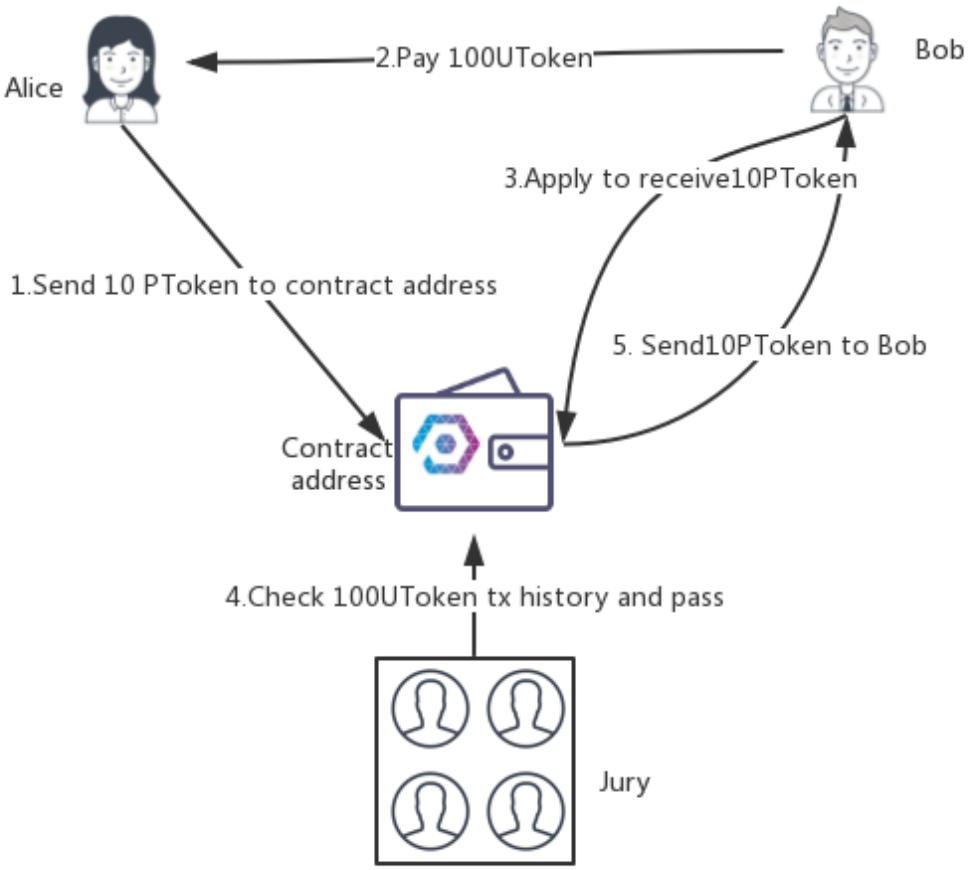

Figure 7.4 The normal process of executing the exchange contract

The circumstance that one party performs the exchange contract while the other defaults on it

The real world is always full of uncertainties, such as changes in the prices of currencies, which will lead one of the parties to feel that the previously signed contract will incur a loss on him/her. At this time, the party will be subjectively unwilling to continue to execute the exchange contract, while the other party has already transferred the agreed coin amount to one of the multi-signature address. Under such circumstance, the party performing the contract may suffer a loss. Therefore, there must be a deposit in

the exchange contract to ensure that the loss suffered by the party performing the contract is compensated. The specific operation of it shall be made as follows:

1. Alice transfers 10 PTokens to the multi-signare address generated by the contract.
2. Bob repents and is reluctant to transfer 100 UTokens to Alice.
3. After 24 hours, Bob still does not transfer the UTokens. Therefore, Alice will file a request to the jury to demand it to adjudicate Bob default on the contract and to return the 10 PTokens, and sign it with her own private key.
4. After checking the contract address and verifying taht Alice has transferred the tokens, but Bob has not, and finfing at the same time that the time have exceeded the timeout specified in the contract, the jury will adjudicate Bob default on the contract.
5. The jury signs the Transaction of return of 10 PTokens invoked by Alice. When 3 or more jurors signed, the 10 PTokens will be transferred to Alice's wallet.
6. The total of 2 PTokens which were paid by Alice and Bob as the deposits to ensure a fulfillmen of the contract, to Alice's wallet as a compensation for the loss caused by Bob's default to her.
7. The jury changes the contract state to being terminated. The contract was executed.

Decentralized exchange

In the example of a decentralized transaction contact, there are four roles: PalletOne jury, an exchange server (responsible for the operation of a trading website and the verification of KYC, AML, currency, coin payment, coin withdrawal, etc.), and Alice and Bob. Although the exchange server is centralized, the finalcompletion and data recoding of a transaction are decentralized, which can prevent the occurrence of such cases as the opaque operation and the theft of a exchange account in a traditional centralized exchange.

Take a transaction between XToken (a kind of token issued by PalletOne) and BTC transaction as an example.

The jury holds 6 pairs of public and private keys, the exchange server holds the same number, and each user holds one, forming a 10/13 multi-signature address, which can form a 10/13 multi-signature address. Since the public keys provided by users are different from each other, each user's coin payment address is also unique and featuring a one-to-one correspondence. A user can do the following in the decentralized exchange:

1. Coin payment

1) Alice wants to exchange a certain number of Bitcoins in her hands for that of XTokens in the decentralized exchange, so she sends the public key of her bitcoin wallet to the contract, which then will generate a multi-signature address.

2) Alice transfer the 1 Bitcoin of her wallet to the multi-signature.

3) After detecting that the wallet is successfully be recharged, the exchange server will file a BTC-Token (this is a Token issued on PalletOne) invocation to ask the contract to send a BTC-Token to Alice's PalletOne address. The Token is in one-to-one correspondence with the currency used to recharge the wallet in kind and face value.

2. Announcement of a purchase order

1) Alice wants to exchange 1 BTC for 1000 XTokens, so she issues an order to purchase XTokens. And then 1 BTC-Token will enter the contract account.

2) The exchange server will also record the purchase order and display it on the website.

3. Announcement of a sale order

1) Bob also wants to exchange the 1000 XTokens in his hands for a certain amount of BTCs, so he

2) announces an oreder to sell 1000 XTokens and transfers the 1000XToken to the contract account.

3) The exchange server provides a matching service. Therefore, when it finds that Alice’s and Bob's

orders match, it will file an invocation to the contract.

4) The contract checks the conditions of both parties and finds that the condition is met, so the 1000 XTokens are transferred to Alice's address, and the 1 BTC-Token to Bob's address.

4. Cancellation of the purchase and sale orders

1) If Alice’s order has not fulfilled after she announced it, she can file to the contract a request to cancel the order.

2) After checking the state of the order, the contract will modify the state to being cancelled if the withdrawal condition is met. The order cannot be matched after being cancelled

5. Coin withdrawal

1) Bob obtained 1 BTC-Token after selling the 1000 XTokens. He files a request to withdraw the token to the contract and provides his own Bitcoin address and the 1 BTC-Token.

2) The contract destroys the 1 BTC-Token and transfers the 1 BTC in the multi-signature BTC address to Bob's bitcoin address.

Conclusion

PalletOne is both an inter-chain protocol and a high-performance "super public chain". PalletOne encapsulates the underlying layers of all blockchains into its adapter through interfaces such as digital currency abstraction, contract abstraction and UTXO abstraction and provide them with a uniform interface to the top layers. The PalletOne virtual machine provides a secure and stable smart contract running environment for common programming languages such as Java and C++, which allows developers to write inter-chain blockchain applications by their common development language without paying attention to the details of a blockchain and enables an application to run on multiple chains simultaneously. At the same time, with the original jury mechanism, DAG data storage and DPOS's Mediator Verification mechanism, it enables both contract execution and data storage to be made in parallel, thus making itself a high-performance "super public chain".

Compared to other inter-chain projects, PalletOne has higher performance and universality. Through PalletOne, multi-stakeholders can benefit each other mutual and build a stable ecosystem.

As more and more public chains go online, different applications can only run on different chains. The separation of information and values brings serious inconvenience to the popularity of blockchain applications and makes the value exchange of inter-chain become the premise of the large-scale application of blockchain. However, the current inter-chain technology is not fully mature. Although many products such as Cosmos, Polkadot, ArcBlock and Wanwei Chain are inter-chain born products but they have their own characteristics. PalletOne is different from them. The difference lies in that PalletOne can not only cross all the chains, but also do more optimization designs in performance consideration, token design, developer friendliness, which make it easier to build a good blockchain application ecosystem on it.

PalletOne itself is just a cross-chain blockchain platform, based on which it is very easy to set up a decentralized exchange application. Compared with the current decentralized exchanges such as 0x, EtherDelta, Kyber.Network, etc., The exchange on PalletOne not only have the characteristics of transaction transparency and asset security, but also can realize more currency supports and faster processing speed, with which it set up a benchmark for the next generation of decentralized exchanges.

The above advantages of PalletOne will become more prominent as more and more participants join, and the ecosystem will become more robust.

Glossary

Vocabulary	Definition
BLS	A threshold signature Algorithm
DAG	Directed acyclic graph, a type of distributed ledger technology that enables parallel writing of data. This technology is applied to blockchain projects such as IOTA, ByteBall, and TrustNote.
DKG	DKG distributed key generation
Jury	Jury, a local consensus group assigned to a specific contract instance, consists of multiple jurors. There are many juries in the entire PalletOne.
Juror	Juror, an actual executor of a contract; a juror can participate in multiple juries at the same time.
Mediator	Mediator, generated through DPOS, consists of 21 super nodes.
Packet	Packet, a stable data structure in DAG, where the units are stable and will not be modified by the insertion of subsequent units.
Unit	The basic data unit in DAG; A unit contains a transaction, can reference multiple parent units, or be referenced by multiple child units.
UTXO	Unspent transaction output, a recording mode, proposed in Bitcoin
VRF	Verifiable Random Function