# EQcoin Bible

Xun Wang

https://t.me/EQcoinUinverse

www.eqcoin.org

# Table of contents

## 1. Terms

1. EQcoin is the original commodity of EQcoin ecosystem. EQcoin is a cryptocurrency，hereinafter referred to as "**EQC**".
2. Type represents the type of lock，hereinafter referred to as "**T**".
3. Operation is defined in Section 2.7.1 below，hereinafter referred to as "**OP**".

## 2. About EQcoin

### 2.1 What's EQcoin?

EQcoin is the first passport-oriented decentralized finance ecosystem of the people, by the people, for the people. EQcoin is open source, decentralized, permissionless, distributed and public shared digital ledger. Passport and EQC are the original commodities of EQcoin ecosystem. Passport and EQC are issued and circulated according to the EQcoin consensus mechanism via the EQcoin ecosystem based on the decentralized finance, everyone can participate in the issuance and circulation of Passport and EQC via crowdsourcing(private property is sacrosanct). The evolution of EQcoin is based on crowdsourcing, everyone can improve and perfect EQcoin via

EQcoin Improvement Proposal.

### 2.2 What's Passport?

Passport is the original commodity of EQcoin ecosystem. Passport has a status, an ID, an EQcoin balance, a nonce, a Publickeys and a StateRoot that can send transactions on EQcoin network. Passport ID is a natural number. Passport ID starts from zero and increases one by one according to the order of Passport issuance. Using the status state object, Passport can add or delete Passport relevant state objects created by the EQcoin Improvement Proposal to add or delete its specific functions. Passport is user-controlled and can be used to deploy multiple smart contracts. Passport owners can provide issue/sell Passport services and smart contract deployment services for everyone and decide how much EQC to charge for issuing/selling Passport and deploying smart contract. Just like Bitcoin Address and Ethereum Account, Passport is anonymous and do not contain information about the owner.

The singularity block is the first block of EQcoin, the No.0 to No.1001 Passports will be issued in this block. Due to “without time at this time” so singularity block's timestamp is

0.

### 2.2.1 POS based Passport issuance consensus mechanism

In order to issue a new Passport, a specific amount of EQC must be staked in it.

#### 2.2.1.1 List of the amount of EQC that needs to be staked for different Passports

| Passport ID No. | The amount of EQC that needs to be staked |
|---|---|
| 0~9 | 51 |
| 10~99 | 51 |
| 100~999 | 51 |
| 1000~9999 | 51 |
| 10000~99999 | 51 |
| 100000~999999 | 51 |
| 1000000~9999999 | 51 |
| 10000000~99999999 | 25.5 |
| 100000000~999999999 | 12.75 |

### 2.2.2 What's Lock?

Passport owner uses Lock to control its relevant Passport and assets. Lock has a lock type and the lock relevant state objects. Lock consists of two lock types at the current stage. New locks can be created through EQcoin Improvement Proposal to extend the functionality of the lock.

#### 2.2.2.1 T0 lock

T0 lock's lock type is 0 and use secp256r1(NIST P-256) elliptic curve to control relevant Passport and its assets. T0 lock has a lock type 0, a SHA3-256 public key hash of secp256r1(NIST P-256) elliptic curve and a CRC32C cyclic redundancy checksum.

#### 2.2.2.2 T1 lock

T1 lock's lock type is 1 which is pay to script hash, it can provides multi-party assets control and 0-trust-based assets security control services against loss and damage of private keys. T1 lock has a lock type 1, a status, a SHA3-256 hash of the relevant redundant lock pairs and a CRC32C cyclic redundancy checksum.

The T1 lock supports the following functions:

1. User is allowed to create a redundant lock pair, which contains N (1<=N<=4) T0 locks from different devices provided by the user. User can select a lock to unlock from the redundant lock pair. Thus, if a single device is damaged, the locks in the redundant devices are still available for use.
2. Single user is allowed to create N(1<=N<=8) lock pairs, and select the M(1<=M<=N) locks in the N lock pairs to unlock.
3. N(1<=N<=8) users are allowed to provide one lock pair per user and must use the N locks in the N lock pairs to unlock.

### 2.2.3 What's Publickeys?

Publickeys has a status, one or more publickeys that used to verify Passport relevant digital signatures. Because the corresponding public keys is stored in relevant Passport when the lock is unlocked for the first time, it can resist preimage attacks.

### 2.2.4 What's StateRoot?

The StateRoot used to store the state objects[1]' MerklePatriciaTrie root of the relevant smart contracts deployed in the current Passport.

---

[1] For examples: smart contract ID, the MerklePatriciaTrie root of the state object for a specific smart contract, etc.

### 2.3 What's EQC?

EQC is the original commodity of EQcoin ecosystem. EQC is a cryptocurrency. EQC needs to be paid in order to use EQcoin decentralized financial services.

### 2.4 Passport and EQC total supply

The max total supply of Passport is 4,294,967,296.

The total supply of EQC is a constant 210,000,000,000 and the decimal is 8. The first block, the Singularity block, will issue about 21,000,000 EQCs, and then will issue 21,000,000 EQCs every year.

## 2.5 What's Intelligent?

Intelligent is an object-oriented programming language designed for developing smart contracts that run on EQCVM. It is compatible with Solidity. It is statically typed, supports inheritance, an Intelligent Standard Library, and complex user-defined programming.

## 2.5.1 What's Intelligent Standard Library (ISL)?

The Intelligent Standard Library (ISL) provides a wide range of features that significantly expand the core Intelligent

language, enhancing its versatility. The ISL is a collection of algorithms, data structures, and other components that can be used to simplify the development of Intelligent programs. The ISL is a collection of Intelligent contracts and interfaces designed to provide commonly used programming data structures and functions, such as tokens, utilities, access control, upgrades, etc. One of the primary advantages of ISL is that it provides the ability to import specific versions of contracts and interfaces included in ISL through an import statement, so there is no need to include ISL-related source code when deploying smart contracts. This can significantly reduce the cost of deploying smart contracts.

## 2.6 What's smart contracts?

Smart contracts are similar to the contracts and agreements in the real world. The only distinction is that they are digital. In fact, a smart contract is a specialized type of computer program designed to execute, control, or document events and actions in accordance with the terms of a contract or an agreement. It runs on EQCVM, itself and its related state object data are stored on the EQcoin blockchain. Once this data is recorded, it becomes traceable and irreversible.

智能合约被部署在 Passport 中，每个 Passport 都可以部署多个绑定的智能合约。Passport 在部署关联的智能合约时为其分配一个唯一的附属的智能合约 ID。

Smart contracts are written in Intelligent. Smart contracts enable trusted transactions, contracts, and agreements to be conducted among diverse, anonymous parties without requiring a central authority, legal system, or external enforcement mechanism.

### 2.6.1 What's Passport dependent smart contract's ID?

Each smart contract is assigned a unique identifier, which is its ID. The associated smart contract can be referenced using its ID. It consists of two adjacent state objects. The first state object is PassportID, which is the ID of the Passport where the smart contract is deployed, and the second state object is SubSmartContractID, which is the sub smart contract ID of the Passport where the smart contract is deployed. The type of the PassportID and SubSmartContractID is EQCBits. The smart contract ID is presented as a string in the form of "PassportID.SubSmartContractID". For example, 1001.1 represents the first smart contract deployed at the 1001st

Passport, while 1001.2 represents the second smart contract deployed at the same Passport, and so on.

SubSmartContractID, which is the sub smart contract ID of the Passport where the smart contract is deployed.

SubSmartContractID 是当前智能合约部署的子智能合约的唯一标识符。The associated sub smart contract can be referenced using its ID.

**2.8 What's EQcoin virtual machine(EQCVM)?**

EQcoin virtual machine(EQCVM) is a crucial component of the EQcoin blockchain. It serves as the runtime environment for managing the state of state variables , enabling smart contract functionality, executing smart contracts and decentralized applications (DApps). It operates as a decentralized computer that runs on the global network of EQcoin nodes. It is responsible for processing and executing code written in EQcoin's native programming language, Intelligent. It is a Turing-complete, sandboxed execution environment. It is compatible with the Ethereum Virtual Machine.

### 2.11 What's Transaction?

Transaction is essentially a signed set of instructions from one Passport. Transaction is used to affect a state change on the EQcoin blockchain, such as transfer of digital assets, change the lock of Passport or execute a function within a smart contract.

Transaction has a status, a Passport ID, a nonce，one or more TxOut arrays and a signature. Using the status state object, Transaction can add or delete Transaction TxOuts created by the EQcoin Improvement Proposal to add or delete its specific functions.

#### 2.11.1 What's Free Trading Power Credit?

During each EQcoin Week (with a different number of blocks set according to the block interval, the total interval is about 7 days), different Passports have specific free trading power credits(the smaller the Passport ID, the more free trading power credit it has) used to execute free transactions. Each Passport will include 2 free trading power credit related status objects, InitialHeight (records the block height of the first free transaction execution of the current Passport of the current EQcoin week, its type is EQCBits and it must be divisible by 2

so that its lowest bit can be reserved as the credit identifier bit) and TotalFreePowerUsed (records the total amount of free power used by the current EQcoin week).

#### 2.11.1.1 List of the Free Trading Power Credit for different Passports

| Passport ID No. | The amount of the Free Trading Power Credit |
|---|---|
| 0~9 | 21 |
| 10~99 | 10.5 |
| 100~999 | 5.25 |
| 1000~9999 | 2.625 |
| 10000~99999 | 1.3125 |
| 100000~999999 | 0.65625 |
| 1000000~9999999 | 0.328125 |
| 10000000~99999999 | 0.1640625 |
| 100000000~999999999 | 0.08203125 |

#### 2.11.1.2 Free Trading Power Credit Algorithm

When each EQcoin week executes a specific Passport transaction, its InitialHeight will be checked, and if it is not in

the current EQcoin week, it will be set to the height of the current block (if it is not divisible by 2, it will be +1) And reset TotalFreePowerUsed to 0. Then execute the current transaction and update TotalFreePowerUsed. If its value is less than half of its free trading power credit, set its credit identifier bit to zero and store it directly. Otherwise, set its credit identifier bit to 1, and if its value is less than its free trading power credit stores the difference between its free trading power credit and TotalFreePowerUsed, set its value to 1.

### 2.11.2 What's Operation?

Operation is essentially a set of instructions from one Passport. Operation is used to affect a state change on the EQcoin blockchain, such as change Passport's lock or deploy a smart contract. Each OperationTxOut contains one or more operation.

Operation has an OP ID and one or more OP state objects. New operations can be created through EQcoin Improvement Proposal to extend the functionality of the operation.

Operation consists of one operation type at the current stage:

1. ChangeLockOP

ChangeLockOP is used to change relevant Passport's lock.

ChangeLockOP has an OP ID of 0 and a lock which is the new lock for the current Passport.

2. ReservedNonceOP

ReservedNonceOP is used to reserve some nonces for future transactions, so that these reserved nonces can be used to execute some offchain transactions (such as EQC lightning network transactions), and these reserved nonces can be used to execute transactions update the relevant global state on the EQC network when needed.

ReservedNonceOP has an OP ID of 1 and a reserved nonce quantity(Values range from 1 to 256), which is the number of nonces reserved.

When the ReservedNonceOP is executed, the following operations will be performed:

1. The nonce of the current Passport will increase the number of reserved nonces.
2. The reserved nonce flag will be marked in the corresponding Passport's status state object (if necessary), and the value of the total number of reserved nonce state object will be increased according to the total number of reserved nonces, and the corresponding reserved nonces' value will be added to the reserved nonce

ZeroOneDynamicMerklePatriciaTrie.

Transaction consists of four TxOut types at the current stage:

1. IssuePassportTxOut

IssuePassportTxOut is used to issue passports. A maximum of 129 IssuePassportTxOuts can be included in the IssuePassportTxOut array.

2. OperationTxOut

OperationTxOut is used to execute operation, for example ChangeLockOP. A maximum of 65 OperationTxOuts can be included in the OperationTxOut array.

3. TransferTxOut

TransferTxOut is used to transfer EQC. A maximum of 257 TransferTxOuts can be included in the TransferTxOut array.

4. SmartContractTxOut

SmartContractTxOut is used to execute smart contract function. A maximum of 5 SmartContractTxOuts can be included in the SmartContractTxOut array.

### 2.11.3 Transaction storage structure

The transaction storage structure consists of two state objects: L(length) state object(the transaction length, its data

type is EQCBits，its count index does not start from 0 but from 71 (because the size of the smallest transaction is 71 bytes)) and V(value) state object(the transaction body).

## 2.12 Transaction use case

### 2.12.1 Issue Passport

1. Adam issues passport and transfers 101 EQCs for Eve (Lock: 0bb...bb).

Before Adam sends the transaction：

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 0<br>Balance:<br>2100000000000000 | |
| LockMate | Lock: 0aa...aa<br>Publickey: null |

Transaction sent by Adam to issue Passport：

| Transaction |
|---|
| Status[2]: 00000001[3] |

[2] The type of the Status state object is EQCBits.

[3] Indicates whether transaction includes IssuePassportTxOut, 0: excludes, 1: includes.

| Passport ID: 0 | |
|---|---|
| Nonce: 0 | |
| IssuePassportTxOut | Status[4] :<br>0000000[5]0[6] |
| | Lock: bb...bb[7]<br>Value:<br>10100000000 |
| Signature: xx...xx | |

**The size of the transaction is 105 bytes.**

After Adam sends the transaction:

| Adam's Passport |
|---|
| Status: 0<br>ID: 0<br>Nonce: 1<br>Balance: |

---

[4] The type of the Status state object is EQCBits.

[5] This state object includes a series of consecutive bits. When IssuePassportTxOut includes only one sub-IssuePassportTxOut uses it to record the lock type part of the current sub-IssuePassportTxOut's lock, and when IssuePassportTxOut includes multiple sub-IssuePassportTxOuts uses it to record the number of sub-IssuePassportTxOuts. The current record value is the lock type part 0000000(0) of the sub-IssuePassportTxOut's lock.

[6] Indicates whether IssuePassportTxOut includes multiple sub-IssuePassportTxOuts, 0: one, 1: multiple.

[7] When IssuePassportTxOut includes only one sub-IssuePassportTxOut uses it to record the hash part of the lock, and when IssuePassportTxOut includes multiple sub-IssuePassportTxOuts uses it to record the full lock. The current record value is the public key hash part bb...bb of sub-IssuePassportTxOut's T0 lock.

| 2099989800090000 | |
|---|---|
| LockMate | Lock: 0aa...aa<br>Publickey:<br>xx...xx |

| Eve's Passport | |
|---|---|
| Status: 0<br>ID: 1<br>Nonce: 0<br>Balance: 10100000000 | |
| LockMate | Lock:<br>0bb...bb<br>Publickey:<br>null |

3. Adam issues passports and transfers 101 EQCs for Moses (Lock: 0cc...cc) and Noah (Lock: 0dd...dd) and charge the service fee of 1 EQC per person. Therefore, after deducting the service fee, the transfer amount is 100 EQC.

Before Adam sends the transaction:

| Adam's Passport |
|---|
| Status: 0 |

| | |
|---|---|
| ID: 0<br>Nonce: 1<br>Balance:<br>2099989800090000 | |
| LockMate | Lock: 0aa...aa<br>Publickey:<br>xx...xx |

Transaction sent by Adam to issue Passports:

| Transaction | |
|---|---|
| Status: 00000001 | |
| Passport ID: 0 | |
| Nonce: 1 | |
| IssuePassportTxOut | Status :<br>0000000[8]1[9] |
| | Lock: 0cc...cc<br>Value:<br>10000000000 |
| | Lock: 0dd...dd<br>Value:<br>10000000000 |

[8] Indicates IssuePassportTxOut includes multiple sub-IssuePassportTxOuts.

[9] Record the current number of sub-IssuePassportTxOuts is 0000000(2).

| Signature: xx...xx |
| --- |

**The size of the transaction is 144 bytes.**

After Adam sends the transaction:

| Adam's Passport | |
| --- | --- |
| Status: 0<br>ID: 0<br>Nonce: 2<br>Balance:<br>2099969800080000 | |
| LockMate | Lock: 0aa...aa<br>Publickey:<br>xx...xx |

| Moses' Passport | |
| --- | --- |
| Status: 0<br>ID: 2<br>Nonce: 0<br>Balance: 10000000000 | |
| LockMate | Lock:<br>0cc...cc<br>Publickey:<br>null |

| Noah's Passport | |
|---|---|
| Status: 0<br>ID: 3<br>Nonce: 0<br>Balance: 10000000000 | |
| LockMate | Lock:<br>0dd...dd<br>Publickey:<br>null |

### 2.12.2 Transfer

1. Adam transfers 101 EQCs to Moses.

Before Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 2<br>Balance:<br>2099969800080000 | |
| LockMate | Lock: 0aa...aa<br>Publickey: |

| | |
|---|---|
| | xx...xx |

Transaction sent by Adam to transfer:

| Transaction | |
|---|---|
| Status: 0000[10]1[11]000 | |
| Passport ID: 0 | |
| Nonce: 2 | |
| TransferTxOut [12] | Passport ID: 2<br>Value[13]:<br>10100000000 |
| Signature: xx...xx | |

**The size of the transaction is 72 bytes.**

After Adam sends the transaction:

| Adam's Passport |
|---|

[10] When transaction includes TransferTxOut indicates whether TransferTxOut includes multiple sub-TransferTxOuts, 0: one, 1: multiple, otherwise indicates whether transaction includes SmartContractTxOut, 0: excludes, 1: includes.

[11] Indicates whether transaction includes TransferTxOut, 0: excludes, 1: includes.

[12] EQcoin uses EQCHelix to store the transfer value and relevant Passport ID's bytes' length in TransferTxOut. On the underlying storage, Value is stored first, followed by Passport ID.

[13] The lowest 5 bits of the binary digits of Value are reserved as identifier bits to store the number of bytes occupied by Value and Passport ID respectively, among which the upper 3 bits are used to store the number of bytes occupied by Value and the lower 2 bits are used to store the number of bytes occupied by Passport ID. In the TransferTxOut, Value is stored first and then Passport ID is stored. Therefore, if the value of the transfer amount entered in TransferTxOut is not divisible by 32, some adjustments need to be made to make it divisible by 32 so that the lowest 5 bits of its binary digits are reserved as identifier bits. It is recommended to adjust it in the following ways: 1. Use the result of value-(value%32) as the transfer amount. 2. Use the result of value+(32-(value%32)) as the transfer amount.

| Status: 0<br>ID: 0<br>Nonce: 3<br>Balance: 2099959700070000 | |
|---|---|
| LockMate | Lock: 0aa...aa<br>Publickey: xx...xx |

| Moses' Passport | |
|---|---|
| Status: 0<br>ID: 2<br>Nonce: 0<br>Balance: 20100000000 | |
| LockMate | Lock: 0cc...cc<br>Publickey: null |

2. Adam transfers 101 EQCs to Moses and Noah.

Before Adam sends the transaction:

| Adam's Passport |
|---|

| | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 3<br>Balance:<br>2099959700070000 | |
| LockMate | Lock: 0aa...aa<br>Publickey:<br>xx...xx |

Transaction sent by Adam to transfer:

| Transaction | |
|---|---|
| Status: 0001[14]1000 | |
| Passport ID: 0 | |
| Nonce: 3 | |
| TransferTxOut | Array length : 00000000[15] |
| | Passport ID: 2<br>Value: 10100000000 |
| | Passport ID: 3<br>Value: 10100000000 |
| Signature: xx...xx | |

[14] Indicates TransferTxOut includes multiple sub-TransferTxOuts.

[15] Record the current number of sub-TransferTxOuts is 00000000(2).

**The size of the transaction is 78 bytes.**

After Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 4<br>Balance:<br>2099939500060000 | |
| LockMate | Lock: 0aa...aa<br>Publickey:<br>xx...xx |

| Moses' Passport | |
|---|---|
| Status: 0<br>ID: 2<br>Nonce: 0<br>Balance: 30200000000 | |
| LockMate | Lock:<br>0cc...cc<br>Publickey:<br>null |

| Noah's Passport | |
|---|---|
| Status: 0<br>ID: 3<br>Nonce: 0<br>Balance: 20100000000 | |
| LockMate | Lock:<br>0dd...dd<br>Publickey:<br>null |

### 2.12.3 Change lock

1. Adam changes his Passport's lock to 0bb...bb.

Before Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 4<br>Balance:<br>2099939500060000 | |
| LockMate | Lock: 0aa...aa |

| | |
|---|---|
| | Publickey:<br><br>xx...xx |

Transaction sent by Adam to change lock:

| Transaction | |
|---|---|
| Status: 000000[16]1[16]0 | |
| Passport ID: 0 | |
| Nonce: 4 | |
| OPTxOut | Status[17]:<br><br>0000000[18]0[19]<br><br>Lock: 0bb...bb[20] |
| Signature: xx...xx | |

**The size of the transaction is 101 bytes.**

After Adam sends the transaction:

| Adam's Passport |
|---|
| Status: 0<br><br>ID: 0 |

---

[16] Indicates whether transaction includes OPTxOut, 0: excludes, 1: includes.

[17] The type of the Status state object is EQCBits.

[18] This state object includes a series of consecutive bits. When OPTxOut includes only one sub-OPTxOut uses it to record the OP ID part of the sub-OPTxOut, and when OPTxOut includes multiple sub-OPTxOuts uses it to record the number of OPTxOuts. The current record value is the OP ID part 0000000(0) of the ChangeLockOP.

[19] Indicates whether OPTxOut includes multiple sub-OPTxOuts, 0: one, 1: multiple.

[20] When OPTxOut includes only one sub-OPTxOut uses it to record the OP body part of the OP, and when OPTxOut includes multiple sub-OPTxOuts uses it to record the full OP. The current record value is the lock part 0bb...bb of ChangeLockOP.

| | |
|---|---|
| Nonce: 5<br>Balance:<br>2099939500050000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: null |

### 2.12.4 Execute smart contract

1. Adam executes the Buy function (ID: 4) of the EQswap smart contract (ID: 1002.2) and uses 0.00000051 EQC to buy 201 Bethard tokens from Bethard.

Before Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 5<br>Balance:<br>2099939500050000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: null |

Transaction sent by Adam to execute smart contract:

| Transaction |
|---|

| | |
|---|---|
| Status: 000[21]1[22]0000 | |
| Passport ID: 0 | |
| Nonce: 5 | |
| SmartContractTxOut | Status: 00100[23]01[24]0[25]<br>Smart contract ID: 1002.2<br>Function ID: 4[26]<br>Value[27]: 51 |
| Signature: xx...xx | |

**The size of the transaction is 72 bytes.**

After Adam sends the transaction:

| Adam's Passport |
|---|

---

[21] When transaction includes SmartContractTxOut indicates whether SmartContractTxOut includes multiple sub-SmartContractTxOuts, 0: one, 1: multiple, otherwise reserved for indicates other states of the transaction.

[22] Indicates whether transaction includes SmartContractTxOut, 0: excludes, 1: includes.

[23] When the current smart contract function is called once uses it to record the function ID, and when the current smart contract function is called multiple times uses it to record the number of calls. The current record value is the current smart contract function ID 00100(4) which is Buy function.

[24] Record the bytes of the Passport ID in the smart contract ID. The current record value is the size of the byte stream corresponding to Passport ID 1002, which is 01(2) bytes.

[25] Indicates whether current smart contract function includes multiple calls, 0: one, 1: multiple.

[26] The value is saved in the Status state object identified by note 21.

[27] The lowest 3 bits of the binary digits of Value are reserved as identifier bits to store the number of bytes occupied by Value. Therefore, if the value of the transfer amount entered in SmartContractTxOut is not divisible by 8, some adjustments need to be made to make it divisible by 8 so that the lowest 3 bits of its binary digits are reserved as identifier bits. It is recommended to adjust it in the following ways: 1. Use the result of value-(value%8) as the transfer amount. 2. Use the result of value+(8-(value%8)) as the transfer amount.

| | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 6<br>Balance:<br>2099939500039949 | |
| Bethard token :<br>20100000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey:<br>xx...xx |

2. Adam executes the Buy function (ID: 4) of the EQswap smart contract (ID: 1002.2) and uses 0.00000051 EQC to buy 201 Bethard tokens from Bethard, then executes the betting function (ID: 3) of the Bethard horse racing smart contract (ID: 1002.3) and uses 201 EQCs to bet that No. 9 of the Royal Ascot's Her Majesty's Plate will win.

Before Adam sends the transaction:

| Adam's Passport |
|---|
| Status: 0<br>ID: 0<br>Nonce: 6<br>Balance: |

| | |
|---|---|
| 2099939500039949 | |
| Bethard token : 20100000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: xx...xx |

Transaction sent by Adam to execute smart contract:

| Transaction | |
|---|---|
| Status: 001[28]10000 | |
| Passport ID: 0 | |
| Nonce: 6 | |
| | Status: 000000[29]00[30] |
| SmartContractTxOut | Status: 00100010<br>Smart contract ID: 1002.2<br>Function ID: 4<br>Value: 51 |
| | Status: 00011010<br>Smart contract ID: 1002.3<br>Function ID: 3 |

[28] Indicates SmartContractTxOut includes multiple sub-SmartContractTxOuts.

[29] Reserved status flag bits.

[30] Record the number of sub-SmartContractTxOuts. The current record value is 00(2).

| | |
|---|---|
| | Value: 20100000000<br>Winner: 9 |
| Signature: xx...xx | |

**The size of the transaction is 83 bytes.**

After Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 7<br>Balance:<br>2099919400029898 | |
| Bethard token : 40200000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: xx...xx |

3. Adam executes the Transfer function (ID: 1) of the Bethard token contract (ID: 1002.1 and transfer 100 Bethard tokens to Eve, Moses and Noah.

Before Adam sends the transaction:

| Adam's Passport |
|---|

| | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 7<br>Balance:<br>2099919400029898 | |
| Bethard token :<br>40200000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey:<br>xx...xx |

Transaction sent by Adam to execute smart contract:

| Transaction | |
|---|---|
| Status: 00100000 | |
| Passport ID: 0 | |
| Nonce: 5 | |
| SmartContractTxOut | Status: 00001[31]011[32]<br>Smart contract ID: 1002.1<br>Function ID: 1<br>Passport ID: 1<br>Value: 100 |

[31] The current record value is the number of current smart contract function calls: 00001(3).

[32] Indicates current smart contract function includes multiple calls.

| | |
|---|---|
| | Passport ID: 2<br>Value: 100<br>Passport ID: 3<br>Value: 100 |
| Signature: xx...xx | |

**The size of the transaction is 77 bytes.**

After Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 8<br>Balance: 2099919400019898 | |
| Bethard token : 10200000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: xx...xx |

| Eve's Passport |
|---|
| Status: 0<br>ID: 1 |

| | |
|---|---|
| Nonce: 0<br>Balance: 10100000000 | |
| Bethard token : 10000000000 | |
| LockMate | Lock: 0bb...bb<br>Publickey: null |

| Moses' Passport | |
|---|---|
| Status: 0<br>ID: 2<br>Nonce: 0<br>Balance: 30200000000 | |
| Bethard token : 10000000000 | |
| LockMate | Lock: 0cc...cc<br>Publickey: null |

| Noah's Passport |
|---|
| Status: 0<br>ID: 3<br>Nonce: 0<br>Balance: 20100000000 |

| Bethard token : 10000000000 | |
|---|---|
| LockMate | Lock: 0dd...dd<br>Publickey: null |

### 2.12.5 Complex transaction

Adam issues passport and transfers 51 EQC for Amon（Lock: 0ee...ee, transfers 101 EQCs to Moses, executes the Buy function (ID: 4) of the EQswap smart contract (ID: 1002.2) and uses 0.00000051 EQC to buy 201 Bethard tokens from Bethard and changes his Passport's lock to 0ff...ff and set the power price to 11 to execute transactions at a faster accounting rate. Before Adam sends the transaction：

| Adam's Passport |
|---|
| Status: 0<br>ID: 0<br>Nonce: 8<br>Balance: 2099919400019898 |
| Bethard token : 10200000000 |

| | |
|---|---|
| LockMate | Lock: 0bb...bb<br>Publickey:<br>xx...xx |

Complex Transaction sent by Adam:

| Transaction | |
|---|---|
| Status: 001011[33]11 | |
| Passport ID: 0 | |
| Nonce: 8 | |
| IssuePassportTxOut | Status: 00000000<br>Lock: ee...ee<br>Value: 5100000000 |
| OPTxOut | Status: 00000000<br>Lock: 0ff...ff |
| PowerPrice | Value: 11 |
| TransferTxOut | Passport ID: 2<br>Value: 10100000000 |
| SmartContractTxOut | Status: 00100010 |
| | Smart contract ID: 1002.2<br>Function ID: 4<br>Value: 51 |

[33] Indicates whether transaction specifies a custom PowerPrice, 0: default, 1: custom.

| Signature: xx...xx |
|---|

**The size of the transaction is 151 bytes.**

After Adam sends the transaction:

| Adam's Passport | |
|---|---|
| Status: 0<br>ID: 0<br>Nonce: 9<br>Balance: 2099904200008847 | |
| Bethard token : 30300000000 | |
| LockMate | Lock: 0ff...ff<br>Publickey: null |

| Moses' Passport | |
|---|---|
| Status: 0<br>ID: 2<br>Nonce: 0<br>Balance: 40300000000 | |
| Bethard token: 10000000000 | |
| LockMate | Lock: 0cc...cc |

| | |
|---|---|
| | Publickey: null |

| Amon's Passport | |
|---|---|
| Status: 0<br>ID: 4<br>Nonce: 0<br>Balance: 5100000000 | |
| LockMate | Lock:<br>0ee...ee<br>Publickey:<br>null |

### 2.13 About MerklePatriciaTrie

#### 2.13.1 What's ZeroOneMerklePatriciaTrie and ZeroOneDynamicMerklePatriciaTrie?

ZeroOneMerklePatriciaTrie is used to store the Global State of the IDKey storage object(for example Passport 、Transaction 、active Smart Contract relevant state objects) in each block of EQcoin. The IDKey state object has a unique ID, which is a natural number encoded consecutively from zero. In the ZeroOneMerklePatriciaTrie, it is bit by bit addressed from

high bit to low bit according to the binary value of the ID of the relevant IDKey state object. ZeroOneMerklePatriciaTrie includes two types of keys, ZeroKey(0) and OneKey(1), they are consist of only one bit(0 or 1).

Note： Due to the ZeroKey and OneKey contain only one character, there is no need to store it, and it can be obtained directly according to the corresponding status bit of its parent node.

ZeroOneDynamicMerklePatriciaTrie is used to store the Global State of the IDKey state objects(for example active and inactive Smart Contract relevant state objects) in each block of EQcoin. In the ZeroOneDynamicMerklePatriciaTrie, it is bit by bit addressed from high bit to low bit according to the binary value of the Hash of the relevant state object. ZeroOneDynamicMerklePatriciaTrie includes two types of keys, ZeroDynamicKey(0xxx) and OneDynamicKey(1xxx)， they are consecutive binary sequences consisting of one or more bits starting with 0 and 1.

Note： When the ZeroDynamicKey and OneDynamicKey contain only one character, there is no need to store it, and it can be obtained directly according to the corresponding status bit of its parent node. When the ZeroDynamicKey and

OneDynamicKey contain multiple characters, there is no need to store its first character because it can be obtained directly according to the corresponding status bit of its parent node.

#### 2.13.1.1 What's BinaryNode?

BinaryNode is the key node that constitutes the ZeroOneMerklePatriciaTrie or ZeroOneDynamicMerklePatriciaTrie dictionary tree. BinaryNode has a status, a key, a ZeroNode, a OneNode and a value. ZeroOneMerklePatriciaTrie includes only two BinaryNodes, ZeroNode and OneNode. BinaryNode's underlying storage implementation includes a HashKey(Hash of BinaryNode's binary raw data, used to support state object verification based on light client protocol) and a StorageKey(UpdateNonce(A natural number starting from 0 and increments by 1 with each modification of the BinaryNode) of BinaryNode, used to access state objects from StateDB).

#### 2.13.1.2 What's BinaryNode Status?

BinaryNode Status is used to identify the composition of the state objects included in the BinaryNode. The type of the BinaryNode Status state object is EQCBits.
The default order of state objects that BinaryNode includes is:

ZeroNode, OneNode, and Value.

BinaryNode Status consists of two status types at the current stage(in the underlying storage, they are compositely stored together):

1. HashStatus, the default universal identifier bit of the BinaryNode which participate in the calculation of the BinaryNode Hash.

00000[0][34][0][35][0][36]

2. StorageStatus, includes HashStatus identifier bit and storage relvant identifier bit of the BinaryNode which does not participate in the calculation of the BinaryNode Hash but it is used to get BinaryNode from StateDB.

2.1 When BinaryNode includes two state objects(ZeroNode&Value or OneNode&Value).

000[0][37][0][38]000[39]

Note: The state object has the smallest StorageKey hereinafter referred to as “A” (when there are multiple equal value minimums, the one with the lowest default order is taken), another state object hereinafter referred to as “B”. If A's StorageKey equals B's StorageKey, the underlying data is A's

---

[34] Indicates whether node includes OneNode, 0: excludes, 1: includes.

[35] Indicates whether node includes ZeroNode, 0: excludes, 1: includes.

[36] Indicates whether node includes Value, 0: excludes, 1: includes.

[37] Identifies whether the two state objects'storageKey are equal, 0: no, 1: yes.

[38] Identifies which node state object'storageKey is the smallest in the default order, 0: left, 1: right.

[39] These 3 identifier bits are the same as in HashStatus.

StorageKey . If A's StorageKey is not equal to B's StorageKey, the underlying data is A's StorageKey and (B's StorageKey - A's StorageKey)(this saves more storage space than directly stores the two state objects'storageKey). For example, if B's StorageKey is 100,001 and the A's StorageKey is 100000, the underlying stored data is 100,000 and 1(this saves a lot of storage space than direct storage 100,000 and 100,001). When need to restore the B's StorageKey, can obtain its value through 100000+1.

2.2 When BinaryNode includes three state objects(ZeroNode, OneNode and Value).

0[40]0[41]0[42]00[43]000[44]

Note: The state object has the smallest StorageKey hereinafter referred to as "A" (when there are multiple equal minimum values, the one with the lowest default order is taken), the state object after A in the default order hereinafter referred to as "B" (If A is Value, the order is calculated from the beginning, so B is ZeroNode) , the state object after B in the default order hereinafter referred to as "C" (If B is Value, the order is calculated from the beginning, so C is ZeroNode) . The one with the smaller StorageKey in B and C hereinafter referred to

---

[40] Identifies whether E's StorageKey is equal to D's StorageKey, 0: no, 1: yes.

[41] Identifies whether D's StorageKey is equal to A's StorageKey, 0: no, 1: yes.

[42] Indicates B's and C's StorageKey who is bigger, 0: B<=C, 1: B>C.

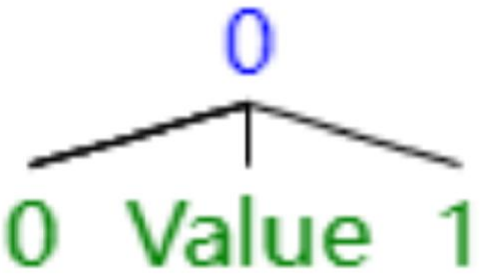
[43] Indicates which state object has the smallest UpdateNonce, 0: ZeroNode, 1: OneNode, 2: Value.

[44] These 3 identifier bits are the same as in HashStatus.

as “D” (when the StorageKey of B and C are equal, the one with the lowest default order is taken)and the The one with the larger StorageKey in B and C hereinafter referred to as “E” (when the StorageKey of B and C are equal, the one with the highest default order is taken).

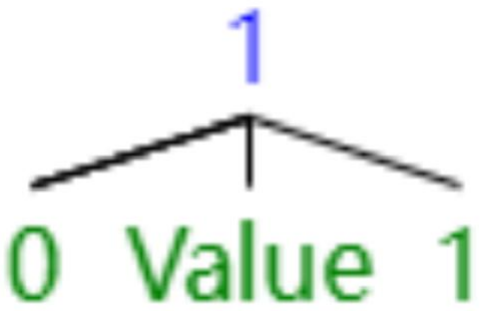
#### 2.13.1.3 What's ZeroNode?

As shown in the figure below, the key of ZeroNode is 0.




#### 2.13.1.4 What's OneNode?

As shown in the figure below, the key of OneNode is 1.



#### 2.13.1.5 What's RootNode?

RootNode is the root of ZeroOneMerklePatriciaTrie which has not Key and Value but has ZeroNode and OneNode. RootNode's StorageKey is equal to its HashKey.

### 2.13.2 What's HexMerklePatriciaTrie?

HexMerkleDynamicPatriciaTrie is used to store the Global State of the HashKey state object(for example Smart Contract relevant state objects) in each block of EQcoin. ~~In the HexMerklePatriciaTrie, it is bit by bit addressed from high bit to low bit according to the binary value of the Hash of the relevant state object~~. HexMerklePatriciaTrie includes 16 keys, DynamicKey0(0xxx), DynamicKey1(1xxx), DynamicKey2(2xxx), DynamicKey3(3xxx), DynamicKey4(4xxx), DynamicKey5(5xxx), DynamicKey6(6xxx), DynamicKey7(7xxx), DynamicKey8(8xxx), DynamicKey9(9xxx), DynamicKeyA(Axxx), DynamicKeyB(Bxxx), DynamicKeyC(Cxxx), DynamicKeyD(Dxxx), DynamicKeyE(Exxx), DynamicKeyF(Fxxx), they are continuous hexadecimal string keywords starting from 0 to F respectively.

#### 2.13.2.1 What's BranchNode?

BranchNode is the key node that constitutes the

HexMerkleDynamicPatriciaTrie dictionary tree. BranchNode has a status, a key, a BranchNode0, a BranchNode1, a BranchNode2, a BranchNode3, a BranchNode4, a BranchNode5, a BranchNode6, a BranchNode7, a BranchNode8, a BranchNode9, a BranchNodeA, a BranchNodeB, a BranchNodeC, a BranchNodeD, a BranchNodeE, a BranchNodeF, and a Leaf. HexMerkleDynamicPatriciaTrie includes 16 BranchNodes, BranchNode0，BranchNode1，BranchNode2，BranchNode3，BranchNode4，BranchNode5，BranchNode6，BranchNode7，BranchNode8，BranchNode9，BranchNodeA，BranchNodeB，BranchNodeC，BranchNodeD，BranchNodeE and BranchNodeF. BranchNode's underlying storage implementation includes a HashKey(Hash of BranchNode's binary raw data, used to support state object verification based on light client protocol) and a StorageKey(UpdateNonce(A natural number starting from 0 and increments by 1 with each modification of the BranchNode) of BranchNode, used to access state objects from StateDB).

#### 2.13.1.2 What's BranchNode Status?

BranchNode Status is used to identify the composition of

the state objects included in the BranchNode. The type of the BranchNode Status state object is EQCBits.

The default order of state objects that BranchNode includes is: BranchNode0，BranchNode1，BranchNode2，BranchNode3，BranchNode4，BranchNode5，BranchNode6，BranchNode7，BranchNode8，BranchNode9，BranchNodeA，BranchNodeB，BranchNodeC，BranchNodeD，BranchNodeE, BranchNodeF, and Value.

BranchNode Status consists of two status types at the current stage(in the underlying storage, they are compositely stored together):

1. HashStatus, the default universal identifier bit of the BranchNode which participate in the calculation of the BranchNode Hash.

0[45]0[46]0[47]0[48]0[49]0[50]0[51]0[52]0[53]0[54]0[55]0[56]0[57]0[58]0[59]0[60]0[61]0[62]

---

[45] Indicates whether BranchNode includes BranchNodeF, 0: excludes, 1: includes.

[46] Indicates whether BranchNode includes BranchNodeE, 0: excludes, 1: includes.

[47] Indicates whether BranchNode includes BranchNodeD, 0: excludes, 1: includes.

[48] Indicates whether BranchNode includes BranchNodeC, 0: excludes, 1: includes.

[49] Indicates whether BranchNode includes BranchNodeB, 0: excludes, 1: includes.

[50] Indicates whether BranchNode includes BranchNodeA, 0: excludes, 1: includes.

[51] Indicates whether BranchNode includes BranchNode9, 0: excludes, 1: includes.

[52] Indicates whether BranchNode includes BranchNode8, 0: excludes, 1: includes.

[53] Indicates whether BranchNode includes BranchNode7, 0: excludes, 1: includes.

[54] Indicates whether BranchNode includes BranchNode6, 0: excludes, 1: includes.

[55] Indicates whether BranchNode includes BranchNode5, 0: excludes, 1: includes.

[56] Indicates whether BranchNode includes BranchNode4, 0: excludes, 1: includes.

[57] Indicates whether BranchNode includes BranchNode3, 0: excludes, 1: includes.

[58] Indicates whether BranchNode includes BranchNode2, 0: excludes, 1: includes.

[59] Indicates whether BranchNode includes BranchNode1, 0: excludes, 1: includes.

[60] Indicates whether BranchNode includes BranchNode0, 0: excludes, 1: includes.

[61] Indicates whether BranchNode's key is one character or multiple characters, 0: one, 1: multiple.

2. StorageStatus, includes HashStatus identifier bit and storage relvant identifier bit of the BranchNode which does not participate in the calculation of the BranchNode Hash but it is used to get BranchNode from StateDB.

**0**[63]**0**[64]**0xxx0**[65]**0000xxx0000**[66]0000000000000000

Note： If Value exists, the smaller absolute value of the difference between its StorageKey and the StorageKey of the largest or smallest BranchNode will be stored.

### 2.13.2.3 What's LeafNode?

LeafNode is the leaf node that constitutes the HexMerkleDynamicPatriciaTrie dictionary tree. LeafNode is used to store state objects. LeafNode has a status, a **StateObjectMate or StateObjectMateArray**[67] and its

---

When the key contains only one character, there is no need to store it, and it can be obtained directly according to the corresponding status bit of its parent HashNode. When the key contains multiple characters, there is no need to store its first character because it can be obtained directly according to the corresponding status bit of its parent HashNode.

[62] Indicates whether BranchNode is a branch node or a leaf node, 0: branch, 1: leaf.

[63] If Value exists indicates the absolute value of the difference between Value's StorageKey and the StorageKey of the largest or smallest BranchNode which is smaller, 0: the smallest, 1: the largest.

[64] Identifies whether the value's StorageKey is larger or smaller, 0: smaller, 1: larger.

[65] The 4-bit identifier identifies the sequence number of each BranchNode included in the current node sorted from small to large(when the StorageKeys of two adjacent BranchNodes are equal, the one with the lowest default order is taken).

[66] The one-bit flag identifies whether each BranchNode( from the second BranchNode sorted in ascending order) is equal to the adjacent BranchNode that is smaller than it.

[67] When the current LeafNode contains only one state object, only one StateObjectMate object is included. When the current LeafNode includes more than two state objects, the current LeafNode

relevant governance state objects. LeafNode related governance state objects can be extended as needed through its status state object. A **HashKey collisionless identifier**[68] bit is included in the current status to identify whether the current LeafNode includes multiple state objects with the same HashKey.

#### 2.13.2.3.1 What's StateObjectMate?

StateObjectMate is used to store state object and its relevant state objects. StateObjectMate has a status, a specific state object and its relevant state objects. StateObjectMate related governance state objects can be extended as needed through its status state object.

#### 2.13.2.3.2 What's StateObjectMateArray?

StateObjectMateArray is used to simultaneously store multiple StateObjectMates, which contain state objects with the same HashKey. These state objects can be identified and distinguished by their UUID(Universally Unique Identifier)s, which are generated by specific algorithms based on the type,

---

includes a StateObjectMateArray (its array subscript 0 represents 2 array elements, array subscript 1 represents 3 array elements, and so on).

[68] HashKey collisionless identifier identifies whether the current LeafNode includes state objects that have collisions, 0: collisionless, 1: collision.

raw data, and associated unique identifiers of specific state objects.

#### 2.13.2.3.3 The HashKey collisionless design of HexMerklePatriciaTrie

Before accessing a specific state object in the HexMerklePatriciaTrie, the global unique access lock bound to its HashKey must be obtained first, and the related state object can be read, written and deleted only after the access right of the access lock is obtained. After the access operation of the relevant state object is completed, its access lock needs to be released.

Read operation:

When performing a read operation, if the current HashKey does not exist, null is returned directly.

When performing a read operation, if the current LeafNode does not have a HashKey collision, it will directly return it including StateObjectMate, and then obtain the corresponding state object from the StateObjectMate according to the UUID provided by the current read operation.

When performing a read operation, if the current LeafNode has a HashKey collision, the StateObjectMateArray included in

it will be returned, and then obtains the corresponding state object from the StateObjectMateArray according to the UUID provided by the current read operation.

Write operation:

When performing a write operation, if the current HashKey does not exist, the current state object is directly stored in the corresponding LeafNode.

When performing a write operation, if the current HashKey exists, and if the current LeafNode does not have a HashKey collision, then compare whether the UUID of the state object contained in the StateObjectMate currently contained in it is consistent with the UUID of the state object to be stored? If the UUIDs are consistent, the corresponding storage is directly overwritten. Otherwise, if the UUIDs are inconsistent, this is a HashKey collision. In this case, the HashKey collisionless identifier of the current LeafNode will be marked as 1 and the StateObjectMateArray will be used to simultaneously store the two state objects.

When performing a write operation, if the current HashKey exists, and if the current LeafNode has a HashKey collision, then compare the UUID of the state object stored in the

current StateObjectMateArray one by one with the UUID of the state object to be stored. If the UUID of a state object stored in the current StateObjectMateArray is consistent with the UUID of the state object to be stored, the corresponding state object is directly overwritten and stored. If the UUIDs of all the state objects stored in the current StateObjectMateArray are inconsistent with the UUID of the state object to be stored, add a new StateObjectMate array element containing the state object currently to be stored in the StateObjectMateArray to store it.

Delete operation：

When performing a delete operation, if the current HashKey does not exist, do nothing.

When performing a delete operation, if the current HashKey exists, and if the current LeafNode does not have a HashKey collision, then compare whether the UUID of the state object contained in the StateObjectMate contained in it is consistent with the UUID provided by the current delete operation. If they are consistent, delete the current LeafNode, otherwise do nothing.

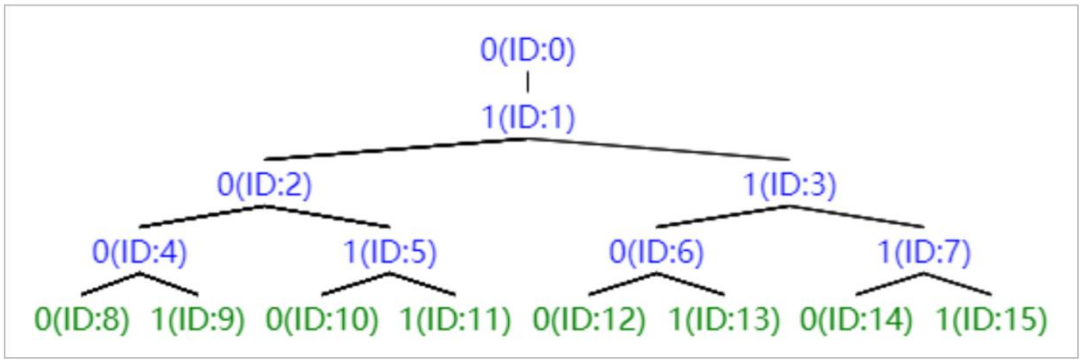
When performing a delete operation, if the current HashKey

exists, and if the current LeafNode has a HashKey collision, then compare the UUID of the state object stored in the current StateObjectMateArray one by one with the UUID of the state object to be deleted. If the UUID of a state object stored in the current StateObjectMateArray is consistent with the UUID of the state object to be deleted, the corresponding StateObjectMate is directly deleted(If the current StateObjectMateArray contains only one StateObjectMate after the delete operation, then mark the HashKey collisionless identifier of the current LeafNode as 0, and delete the current StateObjectMateArray then store the StateObjectMate it contains directly in the LeafNode), otherwise do nothing.

注：这里操作的对象应该统一是 StateObjectMate 而不是 StateObject，从而支持对 StateObject 的数据治理操作。

### 2.13.3 Passport/Transaction Global State chart

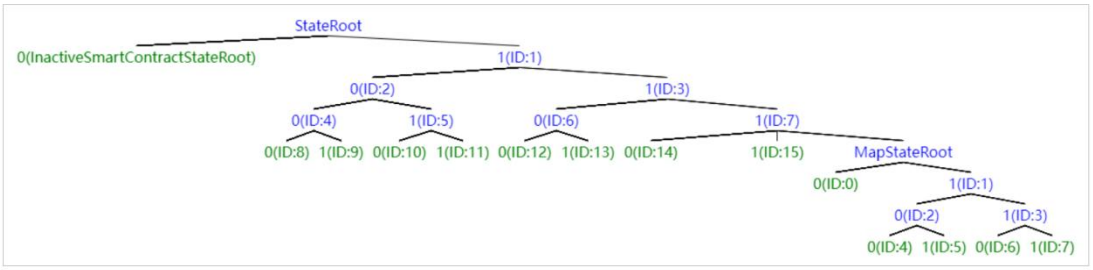
The location of the Passport/Transaction ID from 0 to 15 is depicted in the following figure.




Note: The value of the node in the above figure is the value of the relevant Passport/Transaction that has been omitted in this figure.

### 2.13.3 Smart Contract state object Global State chart

The location of the Smart Contract state object ID from 0 to 15 is depicted in the following figure.




## 2.14 Trusted State Object Verification Protocol

Trusted State Object Verification Protocol is used to prove that a specific state object has been verified by the full node based on the merkle patricia trie proof associated with it.

### 2.15 What's EQCBlock?

EQCBlock is a collection of transactions and other relevant data that are added to the EQcoin blockchain.

#### 2.15.1 What's EQCBlockHeader?

#### 2.15.2 EQCBlock included transactions sorting priority design

#### 2.15.3 EQCBlock included transactions packaging design

#### 2.15.4 EQCBlock's block time interval and maximum transactions per second (TPS)

EQcoin's block time interval is 1 second, and the maximum transactions per second (TPS) is approximately 10,000.

#### 2.15.5 EQCBlock included transactions concurrent execution design

### 2.16 EQcoin's POS&POW full-duplex mode consensus mechanism

EQcoin's POS&POW full-duplex consensus mechanism comprises a POS transaction processing consensus

mechanism and a POW block processing consensus mechanism. These mechanisms work together synchronously, concurrently, and independently.

## 2.16.1 EQcoin's POS transaction processing consensus mechanism

1. 交易收集。

2. 交易验证&执行。

2.1 根据 PowerLimit 确定构成新区块的交易集，并且对其进行排序。

2.2 各个原子并发执行单元按照交易优先级从高到低依次执行各自的交易集队列，并且更新对应的状态对象。

2.3 构造新区块。

2.3.1 根据当前区块包括的交易的最终执行结果更新相关 MPT 树并且生成对应的 EQCHeader 中的 root。

2.3.2 构造新区块。

3.

收到 POW 的获胜者发来的新的 EQCHeader 之后立即广播新的 EQCHeader 和新的区块 body。

## 2.16.2 EQcoin's POW block processing consensus mechanism

竞争新区块添加权，验证新区块，全节点。

## 2.16 EQcoin roadmap

Stage 1 - Inception

EQcoin mainnet online.

Stage 2 - Era

EQcoin supports Intelligent and EQCVM, decentralized exchange based on Matchmaking Transaction Protocol and Lightning Network (LN).

Stage 3 - New dawn

EQcoin supports cross chain through the Interchain Communication Protocol.

Stage 4 - Nirvana

EQcoin moves from POW to POS.

## 2.17 EQcoin milestones

2018-01-01 EQcoin officially launched.

2018-04-10 GitHub Initial commit.

2019 Establish an EQcoin test network to achieve multiple miner nodes based on POW consensus mechanisms to mine, send transactions(issue Passport, transfer EQC and change lock), verify blocks, and compete the longest blockchain.

2020-02-10 Register the domain name of www.eqcoin.org.

2021-02-12 Create EQcoin organization in GitHub.

2021-04 Create EQcoin twitter.

At present, the overall design of the Inception phase of EQcoin has been completed. We have written thousands of pages of research and development technology documents and the code is about 80% complete and including a total of 33000+ lines.

### 2.18 EQcoin GitHub

https://github.com/EQcoin

### 2.19 Our developer community

Currently we have 11 members and 8 outside collaborators. You can visit https://github.com/orgs/EQcoin/people to learn more about our developer community.

### 2.20 Copyright

The copyright of all works released by Wandering Earth 0 Corporation or jointly released by Wandering Earth 0 Corporation with cooperative partners are owned by Wandering Earth 0 Corporation and entitled to protection available from copyright law by country as well as

international conventions.

Attribution — You must give appropriate credit, provide a link to the license.

Non Commercial — You may not use the material for commercial purposes.

No Derivatives — If you remix, transform, or build upon the material, you may not distribute the modified material.

Wandering Earth 0 Corporation reserves any and all current and future rights, titles and interests in any and all intellectual property rights of Wandering Earth 0 Corporation including but not limited to discoveries, ideas, marks, concepts, methods, formulas, processes, codes, software, inventions, compositions, techniques, information and data, whether or not protectable in trademark, copyrightable or patentable, and any trademarks, copyrights or patents based thereon.

For the use of any and all intellectual property rights of Wandering Earth 0 Corporation without prior written permission, Wandering Earth 0 Corporation reserves all rights to take any legal action and pursue any rights or remedies under applicable law.