EQcoin Bible

Xun Wang

https://t.me/EQcoinUinverse

www.eqcoin.org

Table of contents

1. Terms 6

2. EQcoin Overview 7

2.1 About EQcoin 7

2.1.1 Features of EQcoin 8

2.1.1.1 Passport-based blockchain module 8

2.1.1.1.1 Blockchain Model Comparison of Bitcoin, Ethereum, and EQcoin 9

2.1.1.2 Supports performing multiple different operations in the same transaction 9

2.1.1.3 User-based issuance and sale of Passports and deployment of smart contract services. 10

2.1.1.4 A currency supply model with a constant total supply and is more scarce than Ethereum 10

2.1.1.4.1 EQcoin and Ethereum currency supply comparison 10

2.1.1.5 Support the deployment and operation of more cost-effective smart contracts compatible with Ethereum . 11

2.1.1.6 High TPS and low transaction fees 11

2.1.1.7 Flexibility to combine and extend protocols and state objects 12

2.1.1.8 Minimization of state data 12

2.1.1.9 Consistency in transaction sorting and execution order 12

2.2 About Passport 13
2.2.1 PoS based Passport issuance consensus mechanism 13
2.2.1.1 List of the amount of EQC that needs to be staked for different Passports 14
2.2.2 About Status 14
2.2.3 About Identity 15
2.2.4 About Balance 15
2.2.5 About Nonce 15
2.2.5 About LockNonce 16
2.2.6 About Lock 16
2.2.6.1 About Type 18
2.2.6.2 About Body 18
2.2.6.3 Elliptical curve cryptography (ECC) based Lock 18
2.2.6.3.1 T0 lock 19
2.2.6.3.2 T1 lock 20
2.2.6 About Key 21
2.2.7 SmartContract 21
2.2.7.1 About smart contracts 21
2.2.7.2 About SmartContract 22
2.2.7.2.1 About Status 23
2.2.7.2.2 About Balance 23
2.2.7.2.3 About Nonce 24

2.2.7.2.4 About CodeHash 24
2.2.7.2.5 About StateRoot 24
2.3 About EQC 24
2.4 Passport and EQC total supply 25
2.5 About Intelligent 25
2.5.1 About Intelligent Standard Library (ISL) 25
2.6 About EQcoin virtual machine(EQCVM) 26
2.7 About EQswap 27
2.8 About Transaction 27
2.8.1 About Transaction Nonce 28
2.8.2 About Operation 28
2.8.3 Transaction storage structure 31
2.9 Transaction use case 31
2.9.1 Issue Passport 31
2.9.2 Transfer 36
2.9.3 Change lock 41
2.9.4 Execute smart contract 43
2.9.5 Complex transaction 51
2.10 About MerklePatriciaTrie 54
2.10.1 About ZeroOneMerklePatriciaTrie and ZeroOneDynamicMerklePatriciaTrie 54
2.10.1.1 About BinaryNode 56

2.10.1.2 About BinaryNode Status 56
2.10.1.3 About ZeroNode 59
2.10.1.4 About OneNode 59
2.10.1.5 About RootNode 60
2.10.2 About HexMerklePatriciaTrie 60
2.10.2.1 About BranchNode 60
2.10.1.2 About BranchNode Status 61
2.10.2.3 About LeafNode 63
2.10.2.3.1 About StateObjectMate 64
2.10.2.3.2 About StateObjectMateArray 64
2.10.2.3.3 The HashKey collisionless design of HexMerklePatriciaTrie 65
2.10.3 Passport/Transaction Global State chart 68
2.10.3 Smart Contract state object Global State chart 70
2.11 Trusted State Object Verification Protocol 70
2.12 About EQCBlock 70
2.12.1 About SingularityBlock 70
2.12.2 About EQCBlockHeader 71
2.12.3 EQCBlock included transactions sorting priority design 71
2.12.4 EQCBlock included transactions packaging design 71
2.12.5 EQCBlock's block time interval and maximum TPS 72
2.12.6 EQCBlock included transactions concurrent execution

design 72
2.13 About EQcoinFBI 72
2.14 EQcoin roadmap 72
2.15 EQcoin milestones 73
2.16 EQcoin GitHub 74
2.17 EQcoin developer community 74
2.18 EQcoin Thanksgiving Day 74
2.19 Copyright 75

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**".

4. Elliptical curve cryptography, hereinafter referred to as

"**ECC**".

5. "nonce" is an abbreviation for "number used only once".
6. Transactions per second, hereinafter referred to as "**TPS**".
7. Proof of Stake, hereinafter referred to as "**PoS**".
8. Proof of Work, hereinafter referred to as "**PoW**".

2. EQcoin Overview

2.1 About EQcoin

EQcoin is the first passport-based decentralized finance ecosystem of the people, by the people, for the people. EQcoin is an open-source, decentralized, permissionless, distributed, and publicly shared digital ledger. Passport and

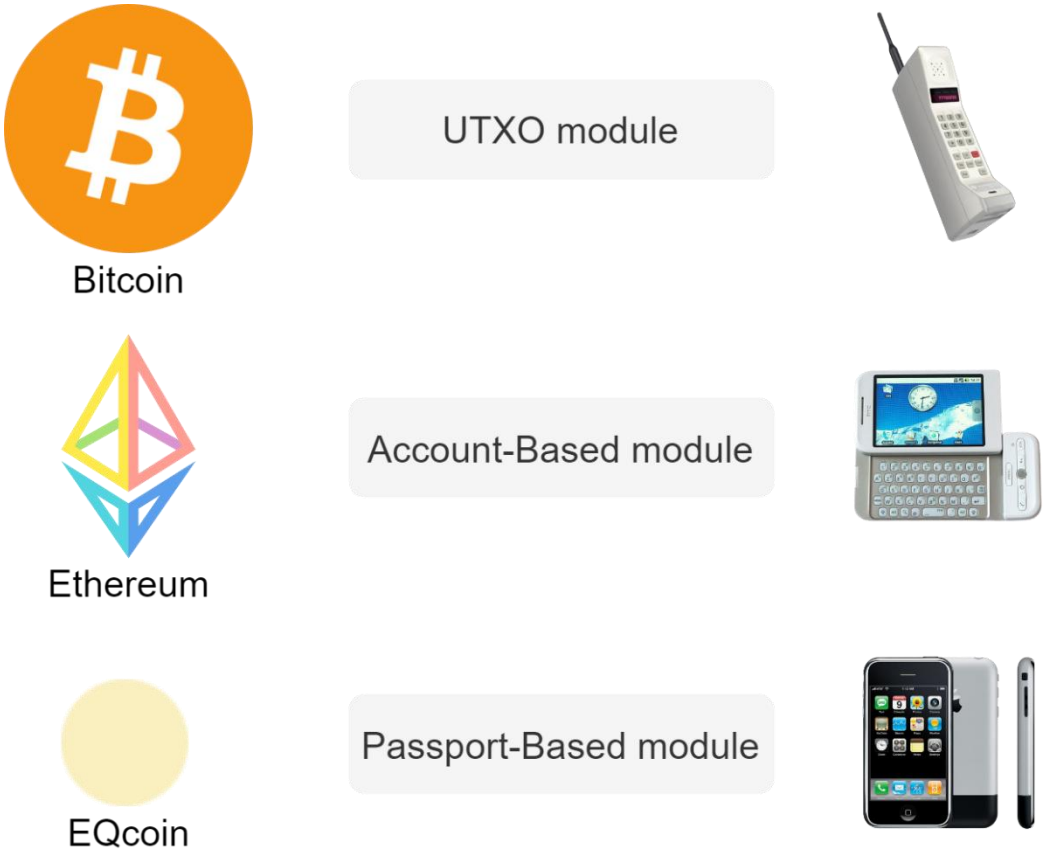
EQC (an abbreviation for "EQcoin") are the original commodities of the EQcoin ecosystem. Passport and EQC are issued and circulated in accordance with the EQcoin consensus mechanism, which operates on decentralized finance principles. This enables everyone to participate in the issuance and circulation of Passport and EQC through crowdsourcing. The development of EQcoin is based on crowdsourcing. Everyone can contribute to the enhancement and refinement of EQcoin through the EQcoin Improvement Proposal (EIP).

2.1.1 Features of EQcoin

2.1.1.1 Passport-based blockchain module

EQcoin invented the first passport-based blockchain model. It enables users to easily manage and trade digital assets, improving their efficiency, trust, comfort, satisfaction, and loyalty by reducing complexity and providing an intuitive, user-friendly experience similar to iPhone.

2.1.1.1.1 Blockchain Model Comparison of Bitcoin, Ethereum, and EQcoin



2.1.1.2 Supports performing multiple different operations in the same transaction

Users can perform multiple operations such as Transfer, SmartContract, IssuePassport, ChangeLock, and multiple OPs at the same time in the same transaction, thus saving transaction fees and providing a better user experience.

2.1.1.3 User-based issuance and sale of Passports and deployment of smart contract services.

Passport owners can provide services for issuing and selling passports, as well as deploying smart contracts for all users.

2.1.1.4 A currency supply model with a constant total supply and is more scarce than Ethereum

EQcoin effectively combats inflation by implementing a currency supply model with a constant total supply, similar to Bitcoin. The decimal of EQcoin is 8, and its circulation and annual issuance are lower than those of Ethereum, which makes it more scarce and more resistant to inflation than Ethereum.

2.1.1.4.1 EQcoin and Ethereum currency supply comparison

Compare items	EQcoin	Ethereum
Genesis block supply	21,000,000~70,000,000 ETH	72,000,000 EQC
Annual supply	About 6,400,000 ETH	2,102,400 or 4,204,800 EQC
Decimal	8	18

Constant total supply	Yes	No

Note1: Ethereum currency supply data comes from Conducting the ETH Census - by Kyle Waters (substack.com).

Note2: Based on the comparison of the minimum unit of Ethereum (wei, 1 ETH = 10*18 wei) and the minimum unit of EQcoin (singular, 1 EQC = 10*8 singularity), the amount of Ethereum issued is 5,714,286 times the maximum supply of EQcoin based on the issuance of Ethereum as shown by the above data sources.

2.1.1.5 Support the deployment and operation of more cost-effective smart contracts compatible with Ethereum

EQcoin supports the deployment and execution of more cost-effective smart contracts that are compatible with Ethereum, establishing a decentralized finance ecosystem that is more cost-effective than Ethereum.

2.1.1.6 High TPS and low transaction fees

EQcoin achieves a high Transactions Per Second (TPS) comparable to EOS and low transaction fees comparable to

Polygon. This provides users with a faster and more cost-effective transaction experience compared to Ethereum.

2.1.1.7 Flexibility to combine and extend protocols and state objects

EQcoin enables the flexibility to combine and extend protocols and state objects, facilitating adaptation to evolving requirements by modifying protocol and state object statuses. Additionally, it can meet new demands by extending protocols and state objects.

2.1.1.8 Minimization of state data

EQcoin significantly reduces the cost of operating a full node by implementing state data minimization. This enables more users to afford the operational costs and ensures the decentralization of EQcoin.

2.1.1.9 Consistency in transaction sorting and execution order

EQcoin ensures consistency in transaction sorting and execution order, effectively preventing fraud and meeting the relevant requirements of decentralized applications (DApps),

such as decentralized exchanges (DEX) or auctions.

2.2 About Passport

Passport is the cornerstone of the EQcoin ecosystem. Passport is user-controlled and can be used for depositing digital assets, sending transactions, and deploying an independent smart contract. A Passport has a Status, an Identity, an Balance, a Nonce, a Lock, a Key and a SmartContract. Passport owners can provide services for issuing and selling passports, as well as deploying smart contracts for all users. They have the authority to determine the amount of EQC to be charged for passport issuance, sales, and deployment of smart contracts. Just like a Bitcoin address and an Ethereum account, a Passport is anonymous and does not contain personal information about the owner.

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 About Status

Status state object 使用 bit 标志位记录它的 Passport 相关的状态，比如有没有部署智能合约，质押模式，是否活跃等. The type of Status state object is EQCBits. 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.

2.2.3 About Identity

Each passport is assigned a unique identifier when it is issued, which is its Identity. The associated Passport can be referenced using its Identity. It is a natural number. The numbering starts from zero and increases sequentially based on the order of Passport issuance.

2.2.4 About Balance

2.2.5 About Nonce

Nonce of a Passport records the total number of transactions that have been sent and confirmed by this Passport using a specific Lock. Nonce ensures that the transaction signed by the Lock bound to the current Passport is unique to the current Lock, thereby avoiding double-spending attacks.

The value starts at 0 and increases by 1 with each sent and confirmed transaction, and is reset to 0 after each Passport Lock change.

2.2.5 About LockNonce

LockNonce of a Passport records the total number of times the current Passport has changed its Lock. LockNonce itself is not a component of the transaction. When signing and verifying transactions, LockNonce will be superimposed as dependency-injected data behind the original data of the transaction and treated as a whole for signing and verification.

The value starts at 0 and increases by 1 after each Passport lock change.

2.2.6 About Lock

Lock is used to safeguard and control the ownership of the Passport, ensuring that only its owner can use it to send transactions, thus preventing it from being illegally stolen by others. Locks are implemented based on various encryption technologies such as Elliptical curve cryptography (ECC) and SHA-3. Lock has a Type, Body and possibly other 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.

safeguard

锁基于各种加密技术（比如 Elliptical curve cryptography (ECC) 和 SHA-3）实现。

允许用户保护和控制他的 Passport 确保只有他本人可以使用它发送交易，从而避免它被非法使用。

用户使用锁证明他对特定护照的所有权和保护和控制它确保只有他本人可以使用它发送交易，从而避免它被非法使用。

锁被用来保护和控制用户的护照的所有权确保只有他本人可以使用它发送交易，从而避免它被他人非法窃取。

和允许用户使用 Key 解锁来使用 Passport

对他的交易进行签名从而证明这是其本人的操作。

Lock 基于私钥公钥加密理论，Lock 的拥有者通过他的私钥签名证明 Lock 的所有权

用户可以使用他的 Key 和相关交易数据验证他的身份证明他对护照的所有权从而可以使用它发送交易。

Passport owner uses Lock to control its relevant Passport

and assets 的使用权.

Locks are implemented based on various encryption technologies such as Elliptical curve cryptography (ECC).

encryption technology

encryption technique

2.2.6.1 About Type

它代表锁的类型。它是 EQCBits 类型。

2.2.6.2 About Body

Lock body is implemented based on various encryption technologies. It is the main part of the lock. Different types of locks may have different lock body implementation methods.

2.2.6.3 Elliptical curve cryptography (ECC) based Lock

根据 Publickey 制作 Lock 的时候可以将当前 Passport 的 Identity 和 LockType 注入一起构造成锁。从而哪怕同一个公钥 safeguard 多个 Passport 也不会重复!将 LockingNonce 也注入(同时在 Lock 中增加一个 LockingNonce 状态对象存储部署此 Lock 时 Passport 的 Nonce）从而 Nonce 复用问题也解决了。从而每次换锁时都可以置零 Nonce 了。

PassportIdentity+LockingNonce+LockType+Publickey

签名交易的时候也可以将 Passport 的 Identity 和 LockType 注入。

交易 MPT 基于 PassportIdentity 设计如何？这样的话会更好呀！因为基于 StateRoot 马上就可以确定当前区块有没有特定的 Passport的交易。客户端在发送交易时需要携带其PassportIdentity 和 LockType。但是在区块中则没必要存储 PassportIdentity（因为存储在 MPT 中了）。不行的 MPT 本身不是元数据，它是元数据驱动的状态数据，区块中并不包含有 MPT 初 root 之外的 data，所以这会导致 PassportIdentity 丢失。从而交易本身依然必须携带 PassportIdentity。

交易元数据规范：

要不要注入包括 LockType?

交易倾向于注入 Nonce、Salt（LockType 和 LockLockNonce）。

2.2.6.3.1 T0 lock

T0 lock's lock type is 0 and use secp256r1(NIST P-256) elliptic curve to safeguard and control the ownership of relevant Passport. T0 lock has a lock type 0, a SHA3-256 public key hash of secp256r1(NIST P-256) elliptic curve. It has self-check-based error correction capabilities that can detect its character errors caused by reading, input or network transmission, thereby ensuring its correctness.

2.2.6.3.2 T1 lock

T1 lock's lock type is 1. It can provides multi-party safeguard and control the ownership of relevant Passport and 0-trust-based Passport security control services against loss and damage of private keys. T1 lock has a LockType, a Status, a SHA3-256 hash of the relevant redundant lock pairs. It has self-check-based error correction capabilities that can detect its character errors caused by reading, input or network transmission, thereby ensuring its correctness.

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.

2.2.6 About Key

Key has a Status, one or more PublicKeys that used to verify Passport relevant digital signatures. Because the corresponding PublicKeys is stored in relevant Passport when the lock is unlocked for the first time, so that it can resist preimage attacks.

restrict access to the full functionality or data of (a computer, mobile phone, file, etc.), especially by requiring a user to verify their identity with a passcode or other form of authentication.

2.2.7 SmartContract

2.2.7.1 About 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 SmartContract 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 has a Status, an Identity, a Balance, a Nonce, a CodeHash and a StateRoot. 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。智能合约 nonce。

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.

2.2.7.2 About SmartContract

Each smart contract is assigned a unique identifier when it is deployed, which is its Identity. The associated smart contract can be referenced using its Identity. It consists of two adjacent state objects. The first state object is Passport Identity, which is the Identity of the Passport where the smart contract is

deployed, and the second state object is SmartContractID, 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.

2.2.7.2.1 About Status

2.2.7.2.2 About Balance

Balance: This represents the balance of the account in Wei.

2.2.7.2.3 About Nonce

Nonce of a SmartContract records the total number of transactions that have been sent and confirmed by this SmartContract. The value starts at 0 and increases by 1 with each sent and confirmed transaction.

2.2.7.2.4 About CodeHash

CodeHash: The hash of the smart contract code is stored in this field.

2.2.7.2.5 About StateRoot

Storage: This field contains the data of the storage variables within the smart contract.

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

2.3 About EQC

EQC is an abbreviation for "EQcoin". It is the original

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

commodity of EQcoin ecosystem. It is a cryptocurrency. Users need to pay EQC 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 at least 21,000,000 EQCs but no more than 70,000,000 EQCs, and then will issue 2,102,400 or 4,204,800 EQCs every year.

2.5 About 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 About 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 About 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.

2.7 About EQswap

EQswap is a decentralized exchange that uses an order book system to facilitate the trading of digital assets on the EQcoin blockchain. As a decentralized exchange, EQswap is permissionless, allowing everyone to trade digital assets or create a new market for exchanging a new pair of digital assets.

2.8 About 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, deploy smart contract, or execute a function within a smart contract.

Transaction has a Status, a its Passport Identity, 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.

2.8.1 About Transaction Nonce

Transaction Nonce is a number that is used only once in a specific transaction. EQcoin network stipulates that the nonce of a new transaction must be the current Nonce of its Passport plus one. Therefore, each new Transaction Nonce must be set to the current Nonce of its Passport plus one.

Transaction Nonce enables preventing replay attacks, which involve a malicious user trying to resend a previous transaction. It ensures that each transaction can be uniquely identified, ordered correctly, processed and verified accurately within the EQcoin block chain.

2.8.2 About 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.

2.8.3 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 69 (because the size of the smallest transaction is 69 bytes)) and V(value) state object(the transaction body).

2.9 Transaction use case

2.9.1 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 ID: 0 Nonce: 0 Balance: 2100000000000000	
LockMate	Lock: 0aa...aa Publickey: null

Transaction sent by Adam to issue Passport：

Transaction	
Status[2]: 0000000[1][3]	
Passport ID: 0	
Nonce: 0	
IssuePassportTxOut	Status[4] : 0000000[5]0[6]
	Lock: bb...bb[7] Value: 10100000000
Signature: xx...xx	

The size of the transaction is 105 bytes.

After Adam sends the transaction:

Adam's Passport
Status: 0

---

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

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

[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.

ID: 0 Nonce: 1 Balance: 2099989800090000	
LockMate	Lock: 0aa...aa Publickey: xx...xx

Eve's Passport	
Status: 0 ID: 1 Nonce: 0 Balance: 10100000000	
LockMate	Lock: 0bb...bb Publickey: 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 Nonce: 1 Balance: 2099989800090000	
LockMate	Lock: 0aa...aa Publickey: xx...xx

Transaction sent by Adam to issue Passports:

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

[8] Indicates IssuePassportTxOut includes multiple sub-IssuePassportTxOuts.

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

	Value: 10000000000
Signature: xx...xx	

The size of the transaction is 144 bytes.

After Adam sends the transaction:

Adam's Passport	
Status: 0 ID: 0 Nonce: 2 Balance: 2099969800080000	
LockMate	Lock: 0aa...aa Publickey: xx...xx

Moses' Passport	
Status: 0 ID: 2 Nonce: 0 Balance: 10000000000	
LockMate	Lock: 0cc...cc

	Publickey: null

Noah's Passport	
Status: 0 ID: 3 Nonce: 0 Balance: 10000000000	
LockMate	Lock: 0dd...dd Publickey: null

2.9.2 Transfer

1. Adam transfers 101 EQCs to Moses.

Before Adam sends the transaction:

Adam's Passport
Status: 0 ID: 0 Nonce: 2 Balance: 2099969800080000

LockMate	Lock: 0aa...aa 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 Value[13]: 10100000000
Signature: xx...xx	

The size of the transaction is 72 bytes.

After Adam sends the transaction:

---

[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.

Adam's Passport	
Status: 0 ID: 0 Nonce: 3 Balance: 2099959700070000	
LockMate	Lock: 0aa...aa Publickey: xx...xx

Moses' Passport	
Status: 0 ID: 2 Nonce: 0 Balance: 20100000000	
LockMate	Lock: 0cc...cc Publickey: null

2. Adam transfers 101 EQCs to Moses and Noah.

Before Adam sends the transaction:

Adam's Passport

Status: 0 ID: 0 Nonce: 3 Balance: 2099959700070000	
LockMate	Lock: 0aa...aa Publickey: 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 Value: 10100000000
	Passport ID: 3 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 ID: 0 Nonce: 4 Balance: 2099939500060000	
LockMate	Lock: 0aa...aa Publickey: xx...xx

Moses' Passport	
Status: 0 ID: 2 Nonce: 0 Balance: 30200000000	
LockMate	Lock: 0cc...cc Publickey: null

Noah's Passport	
Status: 0 ID: 3 Nonce: 0 Balance: 20100000000	
LockMate	Lock: 0dd...dd Publickey: null

2.9.3 Change lock

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

Before Adam sends the transaction:

Adam's Passport	
Status: 0 ID: 0 Nonce: 4 Balance: 2099939500060000	
LockMate	Lock: 0aa...aa

	Publickey: xx...xx

Transaction sent by Adam to change lock:

Transaction	
Status: 000000[16]1[16]0	
Passport ID: 0	
Nonce: 4	
OPTxOut	Status[17]: 0000000[18]0[19] 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 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 Balance: 2099939500050000	
LockMate	Lock: 0bb...bb Publickey: null

2.9.4 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 ID: 0 Nonce: 5 Balance: 2099939500050000	
LockMate	Lock: 0bb...bb 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] Smart contract ID: 1002.2 Function ID: 4[26] 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 ID: 0 Nonce: 6 Balance: 2099939500039949	
Bethard token : 20100000000	
LockMate	Lock: 0bb...bb Publickey: xx...xx

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, 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 ID: 0 Nonce: 6 Balance:

2099939500039949	
Bethard token : 20100000000	
LockMate	Lock: 0bb...bb 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 Smart contract ID: 1002.2 Function ID: 4 Value: 51
	Status: 00011010 Smart contract ID: 1002.3 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 Winner: 9
Signature: xx...xx	

The size of the transaction is 83 bytes.

After Adam sends the transaction:

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

2. 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 ID: 0 Nonce: 7 Balance: 2099919400029898	
Bethard token : 40200000000	
LockMate	Lock: 0bb...bb Publickey: xx...xx

Transaction sent by Adam to execute smart contract:

Transaction	
Status: 00100000	
Passport ID: 0	
Nonce: 5	
SmartContractTxOut	Status: 00001[31]011[32] Smart contract ID: 1002.1 Function ID: 1 Passport ID: 1 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 Value: 100 Passport ID: 3 Value: 100
Signature: xx...xx	

The size of the transaction is 77 bytes.

After Adam sends the transaction:

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

Eve's Passport
Status: 0 ID: 1

Nonce: 0 Balance: 10100000000	
Bethard token : 10000000000	
LockMate	Lock: 0bb...bb Publickey: null

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

Noah's Passport
Status: 0 ID: 3 Nonce: 0 Balance: 20100000000

Bethard token : 10000000000	
LockMate	Lock: 0dd...dd Publickey: null

2.9.5 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 ID: 0 Nonce: 8 Balance: 2099919400019898
Bethard token : 10200000000

LockMate	Lock: 0bb...bb Publickey: xx...xx

Complex Transaction sent by Adam:

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

[33] Indicates whether transaction specifies a higher power price, 0: default, 1: higher.

Signature: xx...xx

The size of the transaction is 151 bytes.

After Adam sends the transaction:

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

Moses' Passport	
Status: 0 ID: 2 Nonce: 0 Balance: 40300000000	
Bethard token: 10000000000	
LockMate	Lock: 0cc...cc

	Publickey: null

Amon's Passport	
Status: 0 ID: 4 Nonce: 0 Balance: 5100000000	
LockMate	Lock: 0ee...ee Publickey: null

2.10 About MerklePatriciaTrie

2.10.1 About 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.

2.10.1.1 About 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).

2.10.1.2 About 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.

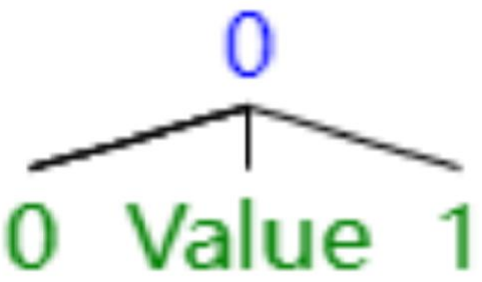
[43] Indicates which state object has the smallest UpdateNonce, 0: ZeroNode, 1: OneNode, 2: Value.

[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).

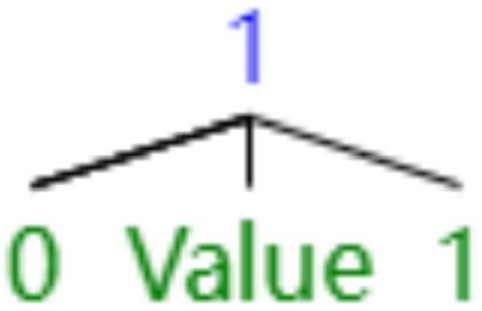
2.10.1.3 About ZeroNode

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



2.10.1.4 About OneNode

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



2.10.1.5 About 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.

2.10.2 About 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.

2.10.2.1 About 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).

2.10.1.2 About 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.

2.10.2.3 About 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.

2.10.2.3.1 About 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.

2.10.2.3.2 About 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.

2.10.2.3.3 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.

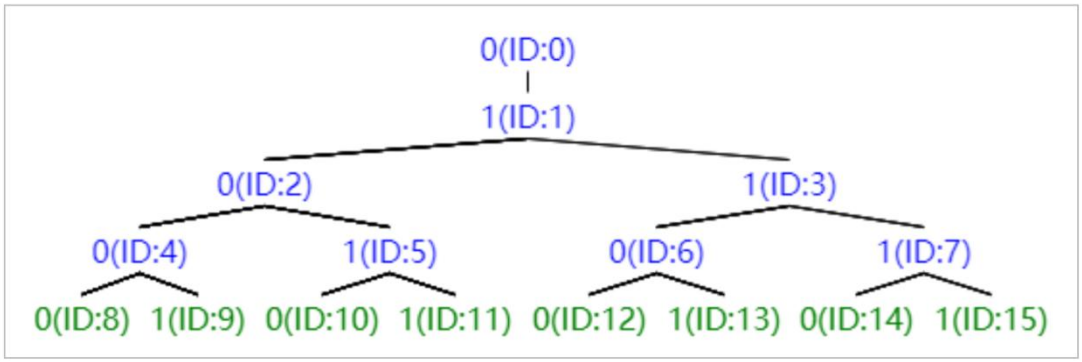
When performing a delete operation, if the current HashKey

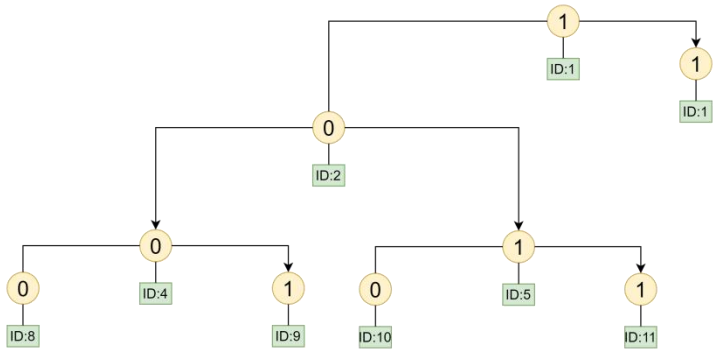
exists, and if the current LeafNode has a HashKey collision, then compare the UUID of the state object stored in the current StateObjectMateArray one by one with the UUID of the state object to be deleted. If the UUID of a state object stored in the current StateObjectMateArray is consistent with the 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.

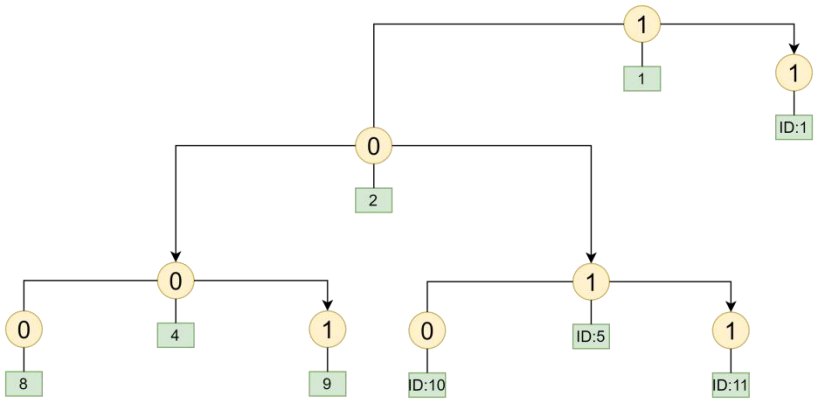
注：这里操作的对象应该统一是 StateObjectMate 而不是 StateObject，从而支持对 StateObject 的数据治理操作。

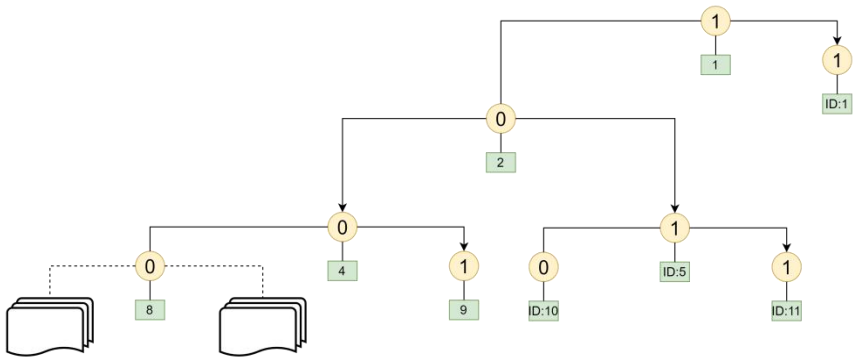
2.10.3 Passport/Transaction Global State chart

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







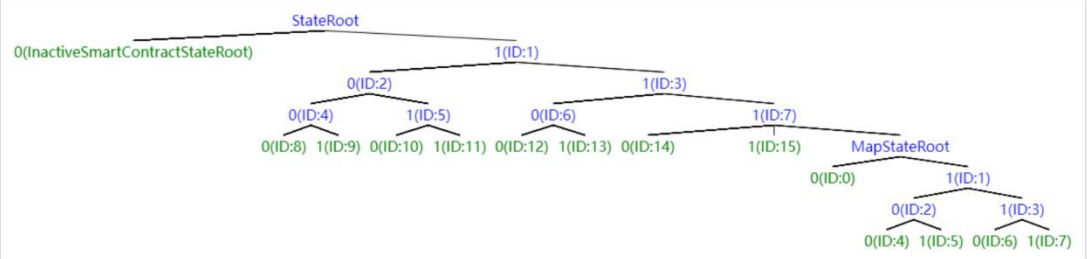


Note: The value of the node in the above figure is the value of the relevant Passport/Transaction that has been omitted in

this figure.

2.10.3 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.



2.11 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.

2.12 About EQCBlock

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

2.12.1 About SingularityBlock

The SingularityBlock is the first block of EQcoin, and the No. 1 to No. 1001 Passports will be issued in this block. Due to

"Without time at the moment," SingularityBlock doesn't have a timestamp.

2.12.2 About EQCBlockHeader

To Be Continued...

2.12.3 EQCBlock included transactions sorting priority design

EQCBlock first packages the transactions with a higher power price (sorted by power price from highest to lowest, and when the power prices are equal, sorted by Passport ID from smallest to largest) and then packages the transactions with the default power price (sorted by Passport ID from smallest to largest).

Each block can only include one transaction for a specific Passport, thus maintaining a balance that increases the accounting probability of high-ID Passport transactions.

2.12.4 EQCBlock included transactions packaging design

To Be Continued...

2.12.5 EQCBlock's block time interval and maximum TPS

EQcoin's block time interval is about 16 second, and the TPS is approximately 1000.

2.12.6 EQCBlock included transactions concurrent execution design

To Be Continued...

2.13 About EQcoinFBI

EQcoinFBI is an abbreviation for "EQcoin Federated Byzantine Intelligence". It is an intelligent federated Byzantine agreement. It is the consensus mechanism of EQcoin.

2.14 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.

2.15 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.

2.16 EQcoin GitHub

https://github.com/EQcoin

2.17 EQcoin 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.

2.18 EQcoin Thanksgiving Day

September 19th is EQcoin Thanksgiving Day.

To express gratitude for the contributions of Bitcoin's founder, Satoshi Nakamoto, a donation of 1,010,000 EQCs is being presented to him.

To express gratitude for the contributions of Ethereum's founders - Vitalik Buterin, Anthony Di Iorio, Charles Hoskinson, Mihai Alisie, Amir Chetrit, Joseph Lubin, Gavin Wood, and

Jeffrey Wilcke - a donation of 80,000 EQCs is being presented to them.

To express gratitude for the contributions of the EQcoin developer community, a donation of 1,180,000 EQCs is being presented to them.

2.19 Copyright

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 Xun Wang without prior written permission, Xun Wang reserves all rights to take any legal action and pursue any rights or remedies under applicable law.