Double-A Chain

A Next-gen public chain commits to Metaverse and Web3.

Version 1.0

2022/02

Content

1. Background

2021 is the start year of the Metaverse. Facebook, Nvidia, Microsoft, Apple, and many other technology giants have announced they will participate in exploring the Metaverse. Their participation will undoubtedly provide a more reliable hardware and software foundation for the Metaverse ecology and accelerate the rapid development of the Metaverse.

However, at the same time, the blockchain network, which is the underlying infrastructure of the Metaverse, has still failed to achieve large-scale acceptance due to scalability and unsatisfactory user experience, which seriously affects the development of the Metaverse. Even on Ethereum, the most widely used smart contract, there are not many Metaverse projects and DAPPs widely adopted. On the other hand, some Metaverse applications can gain a large number of users in a short time but will cause network congestion and high latency or even lead to the paralysis of the entire network, resulting in poor user experience. Furthermore, the increasingly high gas fee also seriously hinders the expansion of the Metaverse user base.

Whether Metaverse or Web3, one of the biggest features is decentralization. Although the PoW consensus mechanism can well ensure the decentralization of the network, it has been criticized more and more because it is not environmentally friendly enough, which shows that the PoW consensus mechanism is not the best choice for the Metaverse and Web3. On the other hand, a few public chains have higher transaction throughput but compromise on decentralization to increase transaction speed.

Certainly, the high throughput, high transaction speed, low cost, and reasonable degree of decentralized blockchain network determine the development speed and market breadth of the Metaverse and Web3.

2. Design Principles

In the future, Metaverse and Web3 applications will face users on a global scale and generate a massive amount of data and digital assets far exceeding the current Internet generation. Therefore, as a next-gen chain focusing on Metaverse and Web3, Double-A Chain has been committed to developing functions such as high transaction speed, high throughput, and low fees as main objects. In addition, it also adheres to the following design principles:

1. Independent blockchain: Technically speaking, Double-A Chain is an independent blockchain, not a second-layer solution.
2. Ethereum Compatibility: Ethereum is still the most practical and widely used smart contract. To make better use of Ethereum's relatively mature applications and communities, Double-A Chain chose to be compatible with the existing Ethereum mainnet. Compatible with the existing Ethereum mainnet means that most of the DApps, ecosystem components, and tools in the blockchain world will be interoperable with Double-A Chain. This strategy can make it possible for Double-A Chain to catch up with Ethereum in performance, ecology, and other aspects.
3. Consensus and Governance: To be more environmentally friendly and leave a more flexible governance mechanism for the community, Double-A Chain will adopt a consensus mechanism based on Staking. Based on DPoS and combined with other governance methods, Double-A Chain will achieve better network performance and complete community governance while achieving faster transaction speed and higher transaction capacity.
4. Cross-chain communication: Double-A Chain will support cross-chain communication between different blockchains in the future. The communication protocol of the Double-A Chain is bidirectional, decentralized, and trustless. It will focus on the circulation of digital assets between the Double-A Chain and other blockchains.

3. Overview

Double-A Chain is a next-gen public chain featuring decentralization, high transaction speed, and low handling fees. Double-A Chain is compatible with smart contracts and supports high-performance transactions. Its token is AAC, and it adopts the APoS consensus mechanism. Double-A Chain focuses on building a high-performance blockchain, empowering the Metaverse and Web3.0 ecosystem, and promoting the rapid implementation and sustainable development of GameFi, NFT, and other application scenarios.

Double-A Chain is committed to leveraging the existing developer community and ecosystem to solve the scalability and usability issues blockchains currently face without compromising decentralization. Double-A Chain will enable developers to design, implement and migrate DApps built on platforms such as Ethereum to Double-A Chain by providing more wallet support, payment APIs and SDKs, products, identity solutions, and other support in the future.

Double-A Chain has always been committed to providing Metaverse, Web3 application development, and ecological landing solutions, providing scalability and excellent user experience for DApps and Web applications in Metaverse, Web3.

4. Manifesto & Vision

4.1 Manifesto

To assist developers to participate in every construction stage of Metaverse and Web 3.0.

4.2 Vision

Technological innovation is the ongoing mission of the blockchain industry, and it is also the driving force behind the digitalized world. We have witnessed the explosion of NFTs and the beginning of the Metaverse. When Ethereum and other chains face problems such as network congestion and rising gas fees, Double-A Chain believes that establishing an efficient and friendly blockchain infrastructure will accelerate the success of the Metaverse, Web3.0, and other digital revolutions. Therefore, the mission of Double-A Chain is not only to be a public chain but also to focus on discovering and supporting high-potential Metaverse, Web3.0 developers, and innovative projects. Relying on the Double-A Chain ecosystem, Double-A Chain is committed to becoming the birthplace of blockchain innovation, especially around Metaverse and Web3.0. Furthermore, to build a complete ecological cycle of technology development, application promotion, and asset trading.

5. Consensus & Governance

5.1 Consensus

Double-A Chain adopts the APoS consensus mechanism combining PoA+DPoS, jointly governed by verification nodes and voters. As a result, it has the characteristics of democracy, efficiency, low transaction cost, low transaction delay, and high transaction concurrency.

Although Proof of Work (PoW) has been recognized as a practical mechanism to implement a decentralized network, it is not environmentally friendly. It requires participants to invest a lot of hardware, capital, etc., to maintain network security. Ethereum and other networks are also experimenting with Proof of Authority (PoA) or its variants in different scenarios, including testnets and mainnets. PoA provides some defense against 51% attacks, improving efficiency and tolerance for certain levels of Byzantine players.

At the same time, PoA protocols have been questioned for being less decentralized than PoW since validators, the nodes which take turns producing blocks, have all authority and are prone to corruption and security attacks. Some blockchains have introduced a different type of Delegated Proof of Stake (DPoS) to allow token holders to vote and elect validators (supernodes). This mechanism achieves decentralization and facilitates community governance.

Double-A Chain proposes to combine DPoS and PoA to achieve a new consensus, the APoS consensus mechanism. It has the characteristics of low transaction cost, low latency, high transaction concurrency, and supports up to 21 validators (validation nodes).

5.2 Validators

5.2.1 Definition

Double-A Chain's validators (also known as supernodes) refer to the nodes that govern and obtain rights and interests of Double-A Chain and are responsible for producing, validating blocks, and processing transactions to drive the normal operation of the blockchain. Validators also participate in Double-A Chain governance and enjoy income distribution. Becoming a validator requires staking a specified amount of AAC and voting by the community.

5.2.2 Type and Amount

Double-A Chain has two types of nodes, validators and validator candidates. The specific rules and rights are as follows：

Type	Amount	Responsibilities	Token
Validators	21	Handling fee+Prize pool reward for nodes	AAC
Candidates	11（In the future, it will be decided whether to increase the number of candidates based on community voting）	Prize pool reward for nodes.	AAC

5.5.3 How to become a validator?

(1)Stake AAC to qualify for node election

Any address can stake more than 10 million AAC and get votes from more than 100 other voting addresses to qualify for validator election (become a candidate). When the node gets the top 21 votes, it will become an active validator for the next epoch. Votes ranked 22-32 become candidate nodes.

If less than 21 nodes meet the conditions, all valid nodes become validators.

If less than 11 nodes meet the conditions, the contract will select nodes according to the actual number.

Exit mechanism: If a validator decides to stop maintaining the network and no longer wants to be a validator, it can request exit from the governance contract. The validator staked tokens will be automatically unlocked after 15 days, thereby exiting.

(2)Voting

Chain users can vote for validator candidates by staking AAC to a Double-A Chain address designated by the node. Each node candidate must obtain at least 100 user votes to qualify for election (i.e., 100 voting addresses).

The top 21 votes become validators, and the 22nd to 32nd is the candidate nodes.

-Voting prerequisite

The minimum number of votes for a user to vote for a validator candidate is 10,000 AAC.

Voting rules

- One AAC, one vote, the more AAC staked, the more votes candidate get.
- One address can only vote for one node
- Addresses of validator candidates cannot participate in voting
- To vote, you must first convert AAC to AACV (AAC Vote), AACV can be converted back to AAC when you withdraw (you need to wait five days)

(3)Exit the voting mechanism

- The AAC staked by users for voting can be retrieved after this round of the election.
- If the user retrieves the vote, the contract will reduce the number of votes obtained

by the corresponding node. The actual number of votes obtained by the node is calculated based on the snapshot of the current round of election deadline.
- After retrieving the vote, the user can continue to vote AACV to other nodes or apply to exchange AACV back to AAC. It takes five days to retrieve back to AAC.

5.5.4 Block

All active validators (validation nodes) are ordered according to predefined rules and take turns to package blocks. Suppose the validator fails to pack a block in time in its round. In that case, the active validator who did not participate in the past n/2 (n is the number of active validators) blocks will randomly execute the block. The normal operation of Double-A Chain requires at least n/2+1 active validators to function properly.

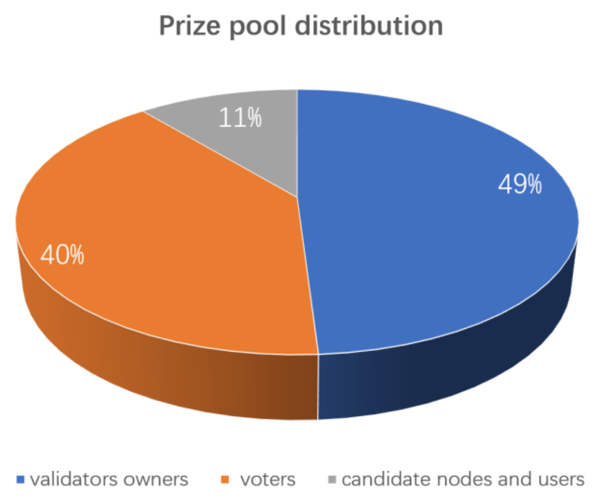
5.5.5 Node Reward

（1）Handling fee

The fees for packaging transactions are all charged by the validator of the packaging blocks.

（2）Prize pool

All AACs supplied every year are set up as a prize pool for nodes.
49% of the pool will be distributed to node owners according to the proportion of votes obtained by validators;
40% of the pool will be distributed to voter according to the proportion of votes obtained by validators.
11% of the pool, divided equally among all candidate nodes and users. If there are less than 11 candidate nodes, the total reward for this part will be reduced to { prize pool amount x 1% x the actual number of candidate nodes }. (To prevent only one candidate node available and the node takes all 11%)



6. Tokenomics

6.1 Supply Mechanism

The total supply of AAC is 12 billion initially. After that, the contract will supply an additional 5% (600 million) every year on top of 12 billion coins.
AAC's additional supply mechanism adopts a hard fork; that is, at a certain point (specified block height), 600 million coins are added to the staking contract. To separate a hard fork requires more than half of the nodes to agree.

6.2 Distribution Mechanism

1. Project party will use the initial supply of 12 billion AAC to exchange old tokens, as follows

（1）Huobi users exchange at 1:10, and the total supply is about 5.45 billion.

（2）OKx users exchange at market price, and the total supply is about 2 billion.

（3）Supply about 4.55 billion to other exchanges and holders of the Ethereum.

2. The 600 million AAC that increases the supply every year will be set up as a node reward pool, and the specific distribution mechanism is following the rules in "Node Rewards."

6.3 Function and Use

AAC is the token of Double-A Chain, which can be used for staking, voting governance, and a future payment method within the ecosystem. AAC provides gas and secures the network. When users send AAC or use Double-A Chain applications, users pay a small fee in AAC, which motivates nodes to process and validate user transactions.

In addition, AAC will serve as a medium for transactions in the future on-chain ecological market. AAC will be used as a settlement tool for Double-A Chain's future Metaverse, Web3.0, GameFi, and other scenario applications and circulation markets. Double-A Chain will set up an AAC Incubation Fund to support the development and operation of Metaverse projects through investment, technical and market support.

7. Performance

- TPS： 1000+
- Average block interval： 3s
- Gas fee： 3gwei

8. Technical Features

8.1 Technical Highlights

- An open and decentralized network to maintain the security of the network and assets.
- Faster transaction speed, lower transaction fee.
- Supports EVM programmability smart contract compatibility to reduce migration costs for developers.
- Support cross-chain asset transfer to optimize users' experience.

8.2 Smart Contract

8.2.1 ARC-20 Token Standard

ARC-20 is an API that implements tokens in smart contracts. It provides multiple functions. For example, transfer tokens from one account to a different account, thereby obtaining the current balance of the account and the total supply of tokens available on the network. In addition to this, it has other features like approving the spending of tokens into third-party accounts.

If a smart contract implements the following methods and events, it can be referred to as an ARC-20 token contract, which, once deployed, will be responsible for tracking tokens created on the Double-A Chain.

Method

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
  function transfer(address _to, uint256 _value) public returns (bool success)
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
  function approve(address _spender, uint256 _value) public returns (bool success)
  function allowance(address _owner, address _spender) public view returns (uint256 remaining)
```

Event

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256
_value)

```

Example

Let's see how the ARC-20 standard enables us to check any ARC-20 token contract on the Double-A Chain. We only need the contract's application binary interface (ABI) to create an ARC-20 token interface. Next, we will use a simplified ABI to make an example.

Web3.py example

First, confirm you have installed Web3.py library

```
$ pip install web3

```

```
from web3 import Web3


w3 = Web3(Web3.HTTPProvider("https://cloudflare-eth.com"))

dai_token_addr = "0x6B175474E89094C44Da98b954EedeAC495271d0F"
  # DAI
weth_token_addr = "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2"
# Wrapped ether (WETH)

acc_address = "0xA478c2975Ab1Ea89e8196811F51A7B7Ade33eB11"
  # Uniswap V2: DAI 2

  # This is a simplified Contract Application Binary Interface (ABI) of an ARC-20
  Token Contract.
  # It will expose only the methods: balanceOf(address), decimals(), symbol()
  and totalSupply()
  simplified_abi = [
  {
  'inputs': [{'internalType': 'address', 'name': 'account', 'type': 'address'}],
  'name': 'balanceOf',
  'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
  'stateMutability': 'view', 'type': 'function', 'constant': True
  },
  {
  'inputs': [],
            'name': 'decimals',
            'outputs': [{'internalType': 'uint8', 'name': '', 'type': 'uint8'}],
            'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
            'inputs': [],
            'name': 'symbol',
            'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
            'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
            'inputs': [],
            'name': 'totalSupply',
            'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
            'stateMutability': 'view', 'type': 'function', 'constant': True
        }
    ]

    dai_contract =
    w3.eth.contract(address=w3.toChecksumAddress(dai_token_addr),
    abi=simplified_abi)
  symbol = dai_contract.functions.symbol().call()
    decimals = dai_contract.functions.decimals().call()
  totalSupply = dai_contract.functions.totalSupply().call() / 10**decimals
  addr_balance = dai_contract.functions.balanceOf(acc_address).call() /
    10**decimals

    #   DAI
    print("===== %s =====" % symbol)
    print("Total Supply:", totalSupply)
    print("Addr Balance:", addr_balance)

    weth_contract =
    w3.eth.contract(address=w3.toChecksumAddress(weth_token_addr),
    abi=simplified_abi)
    symbol = weth_contract.functions.symbol().call()
    decimals = weth_contract.functions.decimals().call()
    totalSupply = weth_contract.functions.totalSupply().call() / 10**decimals
    addr_balance = weth_contract.functions.balanceOf(acc_address).call() /
    10**decimals

    #   WETH
    print("===== %s =====" % symbol)
    print("Total Supply:", totalSupply)
    print("Addr Balance:", addr_balance)

```

8.2.2 ARC-721 Non-Fungible Token Standard

What is ARC-721?

ARC-721 introduces a standard for NFTs; in other words, this type of token is unique and may have a different value than another token from the same smart contract.

All NFTs have a uint256 variable named tokenId, so for any ARC-721 contract, the pair of values contract address, tokenId must be globally unique. A DApp can have a "converter" that uses the tokenId to input and output some interesting images.

Overview

ARC-721 provides functions like transferring tokens from one account to another, getting the current token balance of an account, getting the owner of a specific token, and the total supply of the token available on the network. Besides these, it also has some other functionalities like to approve that an amount of token from an account can be moved by a third party account.

If a Smart Contract implements the following methods and events, it can be called an ARC-721 Non-Fungible Token Contract and, once deployed. It will be responsible for keeping track of the created tokens on Double-A Chain.

Method

```
function balanceOf(address _owner) external view returns (uint256);
function ownerOf(uint256 _tokenId) external view returns (address);
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
function approve(address _approved, uint256 _tokenId) external payable;
function setApprovalForAll(address _operator, bool _approved) external;
function getApproved(uint256 _tokenId) external view returns (address);
function isApprovedForAll(address _owner, address _operator) external view returns (bool);

```

Event

```
event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
```

```
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

```

Web3.py example

First， confirm you have installed the Web3.py Python library.

```
$ pip install web3

```

```
from web3 import Web3
from web3._utils.events import get_event_data


w3 = Web3(Web3.HTTPProvider("https://cloudflare-eth.com"))

ck_token_addr = "0x06012c8cf97BEaD5deAe237070F9587f8E7A266d"    # CryptoKitties Contract

acc_address = "0xb1690C08E213a35Ed9bAb7B318DE14420FB57d8C"    # CryptoKitties Sales Auction

 # This is a simplified Contract Application Binary Interface (ABI) of an ARC-721 NFT Contract.
 # It will expose only the methods: balanceOf(address), name(), ownerOf(tokenId), symbol(), totalSupply()
 simplified_abi = [
 {
        'inputs': [{'internalType': 'address', 'name': 'owner', 'type': 'address'}],
 'name': 'balanceOf',
 'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
 },
 {
        'inputs': [],
        'name': 'name',
        'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [{'internalType': 'uint256', 'name': 'tokenId', 'type': 'uint256'}],
        'name': 'ownerOf',
        'outputs': [{'internalType': 'address', 'name': '', 'type': 'address'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
    },
    {
        'inputs': [],
        'name': 'symbol',
        'outputs': [{'internalType': 'string', 'name': '', 'type': 'string'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
     },
     {
        'inputs': [],
        'name': 'totalSupply',
        'outputs': [{'internalType': 'uint256', 'name': '', 'type': 'uint256'}],
        'stateMutability': 'view', 'type': 'function', 'constant': True
    },
]

ck_extra_abi = [
{
        'inputs': [],
        'name': 'pregnantKitties',
        'outputs': [{'name': '', 'type': 'uint256'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
},
{
        'inputs': [{'name': '_kittyId', 'type': 'uint256'}],
        'name': 'isPregnant',
        'outputs': [{'name': '', 'type': 'bool'}],
        'payable': False, 'stateMutability': 'view', 'type': 'function', 'constant': True
}
]

ck_contract = w3.eth.contract(address=w3.toChecksumAddress(ck_token_addr), abi=simplified_abi+ck_extra_abi)
name = ck_contract.functions.name().call()
symbol = ck_contract.functions.symbol().call()
kitties_auctions = ck_contract.functions.balanceOf(acc_address).call()
print(f"{name} [{symbol}] NFTs in Auctions: {kitties_auctions}")

```

```
    pregnant_kitties = ck_contract.functions.pregnantKitties().call()
    print(f"{name} [{symbol}] NFTs Pregnants: {pregnant_kitties}")

   # Using the Transfer Event ABI to get info about transferred Kitties.
    tx_event_abi = {
    'anonymous': False,
    'inputs': [
            {'indexed': False, 'name': 'from', 'type': 'address'},
            {'indexed': False, 'name': 'to', 'type': 'address'},
            {'indexed': False, 'name': 'tokenId', 'type': 'uint256'}],
    'name': 'Transfer',
    'type': 'event'
    }

    # We need the event's signature to filter the logs
    event_signature = w3.sha3(text="Transfer(address,address,uint256)").hex()

    logs = w3.eth.getLogs({
    "fromBlock": w3.eth.blockNumber - 120,
    "address": w3.toChecksumAddress(ck_token_addr),
    "topics": [event_signature]
    })

    # Notes:
    #     - 120 blocks is the max range for CloudFlare Provider
    #     - If you didn't find any Transfer event you can also try to get a tokenId at:
    # https://etherscan.io/address/0x06012c8cf97BEaD5deAe237070F9587f8E7A266d#events
    #             Click to expand the event's logs and copy its "tokenId" argument

    recent_tx = [get_event_data(w3.codec, tx_event_abi, log)["args"] for log in logs]

    kitty_id = recent_tx[0]['tokenId'] # Paste the "tokenId" here from the link above
    is_pregnant = ck_contract.functions.isPregnant(kitty_id).call()
    print(f"{name} [{symbol}] NFTs {kitty_id} is pregnant: {is_pregnant}")

```

8.2.3 ARC-1155 Multi-token Standard

ARC-1155 is a standard interface for managing contracts of multiple token types. A single deployed contract may include any combination of fungible tokens, non-fungible tokens, or other configurations such as semi-fungible tokens.

What is meant by Multi-Token Standard?
The idea is simple and aims to create a smart contract interface that can represent and control any number of fungible and non-fungible token types. In this way, ARC-1155 tokens can perform the same functions as ARC-20 or ARC-721 tokens, or even both. Most importantly, the functionality of both standards has been improved to increase efficiency and correct apparent implementation errors on the ARC-20 and ARC-721 standards.

ARC-1155 Functions and Feathers
Batch Transfer: Transfer multiple assets in one call.
Batch Balance: Get the balances of multiple assets in one call.
Batch Approval: Approve all tokens to one address.
Hooks: Receive token hook.
NFT Support: If the supply is only 1, it is considered an NFT.
Secure Transfer Rules: A set of secure transport rules.

Batch Transfer
Batch transfers are very similar to regular ARC-20 transfers. Let's look at the regular ARC-20 transferFrom function:

```
// ARC-20
function transferFrom(address from, address to, uint256 value) external
returns (bool);

// ARC-1155
function safeBatchTransferFrom(
address _from,
address _to,
uint256[] calldata _ids,
uint256[] calldata _values,
bytes calldata _data
  ) external;

```

The only difference with ARC-1155 is that we pass the values as an array and an array of ids. For example given ids=[3, 6, 13] and values=[100, 200, 5], the resulting transfer would be:
Transfer 100 tokens with id 3 from _from to _to.
Transfer 200 tokens with id 6 from _from to _to.
Transfer 5 tokens with id 13 from _from to _to.
In ARC-1155, we only have transferFrom, no transfer. To use it transfer as normal, just

set the from address to the address of the calling function.

Batch Balance

The respective ARC-20 balanceOf call likewise has its partner function with batch support. As a reminder, this is the ARC-20 version:

```
// ARC-20
function balanceOf(address owner) external view returns (uint256);

// ARC-1155
function balanceOfBatch(
address[] calldata _owners,
uint256[] calldata _ids
) external view returns (uint256[] memory);

```

Even simpler for the balance call, we can retrieve multiple balances in a single call. We pass the array of owners, followed by the array of token ids.

For example given _ids=[3, 6, 13] and _owners=[0xbeef..., 0x1337..., 0x1111...], the return value will be

```
[
balanceOf(0xbeef...),
balanceOf(0x1337...),
balanceOf(0x1111...)
]

```

Batch Approval

```
// ARC-1155
function setApprovalForAll(
address _operator,
bool _approved
) external;

function isApprovedForAll(
address _owner,
address _operator
) external view returns (bool);

```

Receive Hook

```
function onARC1155BatchReceived(
address _operator,
address _from,
uint256[] calldata _ids,
uint256[] calldata _values,
bytes calldata _data
) external returns(bytes4);

```

ARC-1155 supports receive hooks for smart contracts only. The hook function must return a magic predefined bytes4 value which is given as:

```
bytes4(keccak256("onARC1155BatchReceived(address,address,uint
 256[],uint256[],bytes)"))

```

When the receiving contract returns this value, it is assumed the contract accepts the transfer and knows how to handle the ARC-1155 tokens. Great, no more stuck tokens in a contract!

Secure Transfer Rule

We've touched on a few safe transfer rules already in the previous explanations. But let's look at the most important of the rules:

1. The caller must be approved to spend the tokens for the _from address or the caller must equal _from.
2. The transfer call must revert if
_to address is 0.
length of _ids is not the same as length of _values.
any of the balance(s) of the holder(s) for token(s) in _ids is lower than the respective amount(s) in _values sent to the recipient.
any other error occurs.

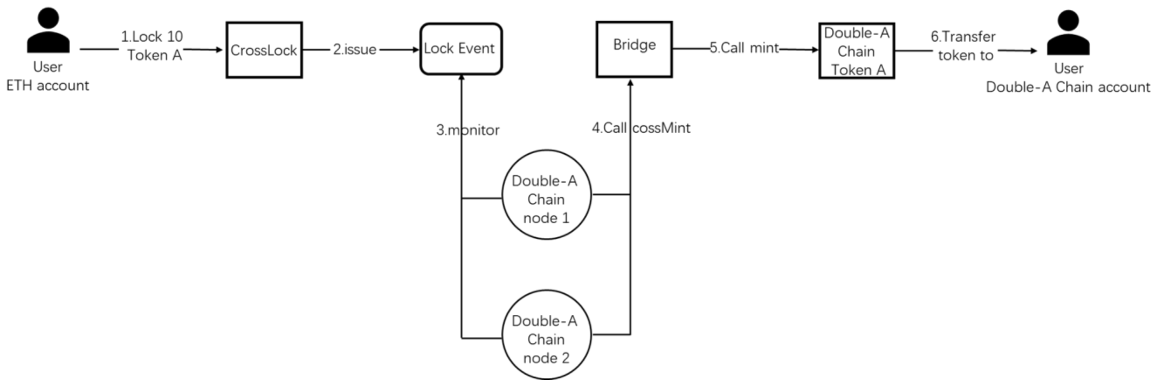
9. Bridge

Double-A Chain will use two EVM-compatible blockchain asset cross-chain solutions, including two types: Native Bridge and Peg Bridge.

9.1 Native Bridge

Native Bridge is a simple asset cross-chain solution. It does not need to convert to intermediate assets, and it can directly map the tokens on Ethereum to Double-A Chain.

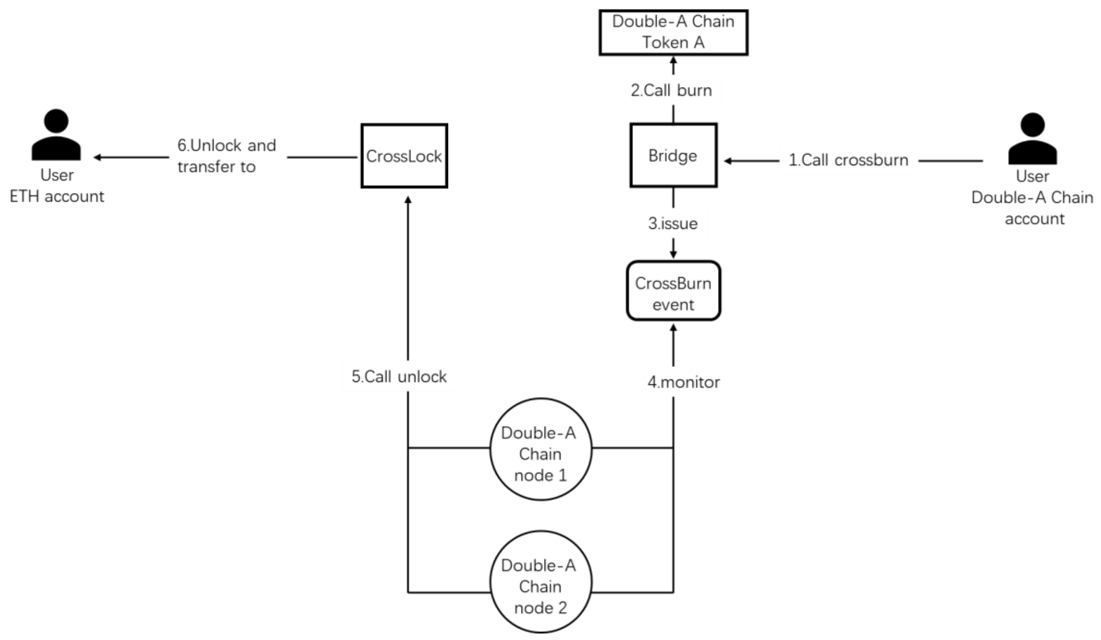
9.1.1 ETH to Double-A Chain



ETH to Double-A Chain

1. First, use the CrossLock feature on Ethereum.
2. The contract locks Token A in the contract and then emits a Lock Event. Double-A Chain runs node 1, and Token A's team runs node 2. These two nodes will monitor Lock Events on the Ethereum CrossLock contract.
3. The Bridge in the picture is a smart contract located on the Double-A Chain. When node1 and node2 find a Lock Event, they will request the crossMint function of the Bridge contract, respectively. crossMint uses openzeppelin's AcessControl for role assignment, and only Crosser can call crossMint.
4. The contract will assign Roles to node 1 and node 2. When nodes 1 and 2 calls crossMint, they vote on this cross-chain behavior. This behaviour needs both node1 and node2 to agree to pass.
5. After the vote is passed, the bridge will call the mint function of the Token A contract. Therefore, the Token A team needs to deploy the contract on the Double-A Chain in advance and allow the Bridge contract to mint or destroy Token A.
6. If the minting function of Token A is successfully performed, The contract will send token A to the user's address on Double-A Chain.

9.1.2 Double-A Chain to ETH



1. User calls the cross burn function of the Bridge contract.
2. The Bridge contract calls the destruction function of the Double-A Chain Token A contract to destroy the user's token.
3. The Bridge contract emits a CrossBurn event.
4.Node1 and node 2 are run and controlled by Double-A Chain and Token A teams, who monitor CrossBurn events. When node1 and node2 find a CrossBurn event, they will call the unlock function of the CrossLock contract on Ethereum.
5. The caller's permission needs to be checked in the unlock function. Only node1 and node 2 can call the unlock function. When the unlock function is called, node1 and node2 will vote on this cross-chain behaviour. When both node1 and node2 agree, the cross-chain is passed for this time.
6. After the previous step is completed, Token A locked in the contract is unlocked and transferred to the user.

9.2 What should our partners do

1. Contact the Double-A Chain team for more details and request to deploy a smart contract.
2.Deploy your ARC-20 token on the Double-A Chain, implement the IToken interface, and give the Bridge contract the right to call mint and burn function.

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.0;

import "@openzeppelin/contracts/token/ARC20/IARC20.sol";

```

```
interface IToken is IARC20{
    function mint(address to, uint amount) external returns(bool);
    function burn(address from, uint amount) external
    returns(bool);
}
```

3. Obtain the Smart Bridge Node program from the Double-A Chain team and inform Double-A Chain of the Ethereum account address controlled by the node. Double-A Chain will grant permission to call the unlock function of the CrossLock contract and the crossMint function of the Bridge contract. At the same time, our partners must also provide an address to charge cross-chain fees.
4. Run the Smart Bridge Node, keep the node online, and ensure that the account controlled by the node has enough ETH and BNB to call the contract.

9.3 Costs and Cross-chain fees

Double-A Chain and partners need to run Smart Bridge Node and call-related contracts, which will incur costs. Therefore, the Native Bridge solution will charge cross-chain fees when users cross-chain and the charging behavior occurs in the Bridge contract. The specific amount of the fee is determined by Double-A Chain and partners.

10. Wallet

Wallet Support
AAC supports all wallets with custom networks such as Metamask, imToken, TokenPocket, and Coinbase Wallet.

11. Roadmap

【Ground】 2021 Q4——2022 Q2

The Initial version of Double-A Chain.
The system is stable and easy to use.
Developers can develop and promote DApps at a low cost.
Users can participate in DApps on Double-A Chain without hesitation.
Start Public Beta: Supported by blockchain browsers and wallets, the main chain achieves greater throughput, lower transaction costs, and faster transaction speeds.
Ecosystem Incubation: Technical service systemization, developer SDK, convenient asset transfer, etc.

【Sky】 2022 Q3——Q4

The protocol is further optimized.
Double-A Chain shoulder the mission of connecting Metaverse, web3.0, and Defi, allowing more users to join and use Metaverse and web3.0 applications.
Provide developer services:
Full developers' kit
Complete developer forum, blog, and FAQ information
The rapid development of on-chain ecological infrastructure
All-new Open ID
The personalized portal accurately matches DApps for users.

【Galaxy】 2023 Q1——Q2

Enable our Bridge and expand performance while retaining the decentralized advantages of distributed protocols.
Bridge:
Cross-chain interoperate contract
Cross-chain interoperability protocol
More cross-chain application
DAO Governance: Governance tools are further improved

【Universe】 2023 Q3——Q4

Landing of large-scale commercial applications.
Support a variety of traditional businesses to run smoothly on the chain

Full Technical Support：

Support variety of virtual machines
Multiple zero-knowledge proofs and privacy protection capabilities

Multiple signature capability
Storage compression and expansion solution
Multidimensional

12. Disclaimer

Nothing in this white paper is meant to recommend or invite users to buy any tokens. The sole purpose of Double-A Chain issuing this white paper is to receive feedback and opinions from the public. Suppose Double-A Chain is to sell any tokens (or "Future Token Simple Agreement") at any time. In that case, it will do so by issuing additional documents (including disclosure documents and risk factor documents). These authoritative documents should also include an updated version of this white paper, differing materially from the current version.

Nothing in this White Paper shall be deemed or warranted as a guarantee or promise of how Double-A Chain or the Tokens will develop or the utility or value of the Tokens. This white paper outlines the current plan, which may change at its sole discretion, and the success of its plan will depend on many factors beyond Double-A Chain's control, including market-based factors and a variety of factors within the data and cryptocurrency industry. Any statements regarding future events are based solely on Double-A Chain's analysis of the issues described in this white paper, and that analysis may prove to be incorrect.