Ambr, the unlimited scalable platform with asynchronous smart contract and cross-chain support

Draft 0.6.1

Ambr Team

Support@ambr.org

Summary

Ambr is an asynchronous cross-chain platform that supports smart contracts. The platform ensures security while providing a high degree of flexibility, scalability, and asynchronous concurrency in real time. Unlike most other platforms, the Directed Acyclic Graph (DAG) data structure is used, instead of a blockchain, and along with the Casper protocol make up the basis of Ambr. Each account here has a private chain and verification chains are used to confirm transactions between numerous account chains without affecting overall performance. And upon the completion of the Casper consensus, transaction fees are distributed accordingly. This approach grants Ambr powerful data processing capabilities and guarantees security.

Ambr's virtual machine (AVM) is partially compatible with some EVM features and is based on the ETH smart contract but improved to be more efficient and practical. AVM solves the problem of consistency within the DAG platform to enable the usage of smart contracts. Ambr's account chains and relay chains work together to conduct secure and fast transactions with other currencies. This cross-chain technology enables not only cross-link transfer, but also allows tranfering of multi-currency within the chain.

In addition, the decoupling protocol allows Ambr to have unlimited transaction processing capabilities. Unlike the Lightning Network and normal status channels, the observer role is introduced to punish the malicious to ensure the interests of both parties in high-frequency secondary delinking transactions.

I. Introduction

The traditional centralized model of electronic money transfer has many problems, such as high maintenance costs, reversible information processing, no privacy protection, and low efficiency. The new distributed architecture has greatly improved the processing abilities compared to more traditional models.

In 1998, Wei Dai's b-money had realized this idea very well, but the consensus issue was not well resolved. Hal Finney pioneers the Proof of Work (PoW) concept in 2005. Following encryption technology and based on the PoW mechanism, Satoshi Nakamoto created, in 2009, a widely recognized digital cryptocurrency known as Bitcoin.

Bitcoin guarantees currency security through the concept of chain structure and PoW, and with the development of the Internet, is becoming more widely accepted as a payment option. However, with the increase in the number of transactions, Bitcoin's chained structure can no longer meet the exponential growth of transactions quantities, because each transaction takes too long. Although many digital currencies with different consensus mechanisms came after, such as using PoS, DPoS, etc., none fixed the problems with the chain structure.

This article will introduce a new type of data structure based on DAG and a new consensus protocol, Casper, to achieve the lag-less cryptocurrency Ambr while guaranteeing security.

II. Background

Blockchain is one of the most significant developments in information technology over the past few years. Considered to be a subversive innovation of the computing model following mainframes, personal computers, and the Internet, the technology is on track to be the next big advancement.

The United Nations, the International Monetary Fund, and countries around the world are all paying close attention to the development of blockchain and actively exploring its applications.

The main features of the blockchain are:

1. Decentralization

The use of distributed accounting and storage results in no centralized hardware or management organization. The rights and obligations of all nodes are equal. Data blocks in the system are maintained by nodes with maintenance functions for the entire system.

2. Openness

The system is completely open. While the private information of the parties to the transaction is encrypted, the data of the blockchain is open to all. And anyone can query the blockchain data to develop related applications through the open interface. The entire system information is highly transparent.

3. Autonomy

The blockchain uses consensus-based specifications and protocols (such as a set of open and transparent algorithms) to allow all nodes in the entire system to exchange data freely and securely in a trusted environment. Aa an added benefit, this system is difficult to attack.

4. Information cannot be modified

All information successfully verified and added to the blockchain will be stored permanently. Unless a party simultaneously control more than 51% of nodes in the system that are active in the PoW consensus, modifications to the database on a single node are not valid. Therefore, the system has a high level of data stability and reliability.

5. Anonymity

Since the exchange between nodes follows a fixed algorithm, the data exchange is trust-free (The procedural rules in the blockchain will automatically determine if the activity is valid). The counterparty can become trusted while remaining anonymous.

Ambr aims to create a set of distributed systems with new architectures and rules.

The above features are the characteristics that blockchain technology must have. In addition, Ambr will introduce large-scale innovations based on these foundations.

III. Assumption and Prospect

Purpose

With the increasing demand for information sharing, the industry requires a more secure decentralized platform. Bitcoin and Ethereum emerge as the times require, but the design of the chain structure creates inefficiencies. With disadvantages causing the current platforms to no longer meet people's growing needs, a new system is needed.

Ambr was designed as a common decentralized application platform that uses the DAG's book structure. Transactions in the books are grouped by account. Using the layered consensus algorithm, the writing and confirmation of transactions are decoupled, and the high performance and scalability of the system can be guaranteed. Ambr's virtual machine part uses Ethereum's smart contract language and goes hand in hand with Solidity.

Proper extensions were made available to provide even more powerful narration capabilities. In addition, the new message-driven asynchronous architecture greatly increases system throughput and scalability. Ambr has built-in digital tokens to create a healthy and transparent community environment for better service to users. This document details the architectural guidelines and system implementation details of Ambr.

Assumptions

Ambr is defined to fulfill the following requirements:

- A complete non-disruptive DAG-based smart contract system
- Ability to execute smart contracts both independently and in parallel
- High degree of network adaptability
- Quantum security
- Rapidly confirmed consensus system
- High-performance, storage-friendly system
- Green consensus algorithm/verification protocol

The PoW consensus algorithm has good security and has been fully validated in Bitcoin and Ethereum. However, this algorithm needs to consume a large amount of computing resources to solve the mathematics problem and cause energy waste. At present, Ethereum's overall TPS is only about 15, completely unable to meet the needs of decentralized applications.

Building high quality software is very challenging, and has led to issues with subsequent maintenance, software upgrades and others. The Ambr project will be developed using a modular model, leading to an abstraction layer. The boundaries of the system will be strictly defined, and maintain synchronization with the community to achieve a continuous and iterative development model. The development of quality software systems is highly supported.

In order to meet the above requirements, the design focuses on the following aspects:

- A well-layered software architecture

- A complete simulation test system and mathematical model to verify at any time
- Use functional programming paradigm and object-oriented programming paradigm
    - One is to make improvements to better handle distributed and parallel processing
    - The other is to increase code maintainability and readability
- Good code review mechanism
- High performance, robustness, scalability

Outlook

- Ambr will position itself as a leader in future blockchain technology and will continue to maintain a continuous investment in technology
- Ambr will inject a total of 30% tokens into the community in the future to create a strong, united and positive developer community

IV. Consensus

Blockchain system is a typical distributed system. In a fully decentralized distributed system, the consensus is equivalent to the entire system infrastructure, which largely determines the security and efficiency of the system, as well as the degree of resource consumption.

A. Consensus mechanism

The four mainstream consensus mechanisms are PoW, PoS, DPoS, and DAG. The following is a brief analysis:

1) **PoW:** Proof of Work, this is a decentralized system that distributes rights according to the power of the mining machine. The disadvantages are:

a) Consumption of social resources. The calculated results of computational power are useless and consume a large amount of resources.

b) Inefficiency. The blocking rate is inversely proportional to security, and is no longer useful to the further developments of decentralized blockchain applications.

c) Centralization. The characteristics of decentralization are being lost due to mining pools concentrating resources.

d) Barrier to entry. The gathering of computational power is not friendly to new public chains, and PoW based public chains have a difficult time obtaining large amounts of computational power to maintain system security in the early stages.

2) **PoS:** Proof of Stake, in order to deal with the problems PoW faces, PoS uses the shares to decentralize the system. Assumes that the more shares owned, the more vested interest in the system. Some problems for this system:

a) Save social resources, do not need dependence on computing power, and eliminate waste of resources

b) The rights and interests are more aligned. The person who owns the greater right also has more shares at the same time.

c) Transfer of equity remains slow. Since everyone has the right to vote, the overall confirmation rate is still slow.

d) The cost of being malicious is low. Although the interest is relatively strong, due to lack of a punishment mechanism, the cost of being malicious is not so high.

3) **DPoS:** Delegated Proof of Stake. In order to pursue efficiency, DPoS sacrifices the decentralized nature. Due to the super node's processing power being very strong, the overall system is fast and validation efficiency is very high. However, centralization is prevalent while using this mechanism. Due to the different channels of communication, the super node has the strongest ability to spread on the market. Ordinary nodes have no bookkeeping rights and are forced to be represented by super nodes. In this case, conflicts can form between rights and interests. And decisions end up based on the interests of select individuals.

a) Decentralized only on the outside. Similar to a presidential election: one person, one vote, and each person may be elected as the president. However, because the barrier to entry is too high, the information distribution channel is limited, and ordinary users join the competition and the probability of victory is almost zero, making the super node become a few people's games.

b) Agency problems. Having a president does not mean that the interests of the country take the lead. When the interest of the president conflicts with the interests of the country, will the president really sacrifice his or her personal interests for the benefit of the country?

c) Very suitable for new blockchain projects but also very dangerous. The super-nodes in the new project are very powerful and for a long time, all super-nodes are in the hands of only one person.

4) **DAG**: directed acyclic graph. Due to the single-chain structure of traditional blockchain projects, speed of transactions is low. DAG will bring the blockchain technology to an asynchronous era. The following are some features:

a) The throughput is huge and far exceeds DPoS.

b) The structure is complex and error-prone. When the transaction volume is large, due to the complex structure of the chain, the historical consensus is likely inconsistent, or the system cannot serialize the transaction. Resulting in some transaction never being confirmed.

Some projects have to rely on various centralized organizations to ensure the stable operation of the project, creating centralization issues.

c) No transaction fees. On the surface, seems like an advantage, but is actually a drawback. Without the fee, the attacker can send a large number of costless transactions to occupy the resources.

Most DAGs without fees also have shortcomings in data structures and consensus issues. No fees means no need to mine, and no be inflation. But, expect a lack of interests from the people who maintain the normal operation of the system.

d) The ecological construction is difficult. Based on the above, misusing the system is possible without fees, and without profit to incentivize people to maintain the system. This system will not function.

B. Consensus design principles

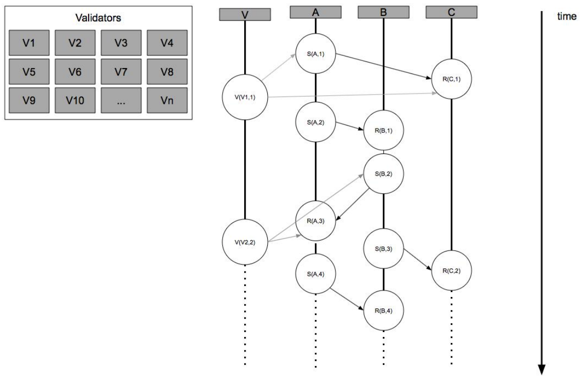
Security: security is the most important feature of all blockchain systems. Ambr hopes to be an absolutely secure and fully decentralized consensus system.

Performance: Under the premise of ensuring security, the system should have ultra-high transaction speeds.

C. System Overview

Unlike the traditional blockchain consensus, Ambr uses Casper and DAG as the basis for consensus. This combo allows the security and decentralization features of Casper, and the high transaction speeds of a DAG.

```
Unit{
   version:0000...0001,
   type:send,
   balance:9999,
   previous:DC32...1CD1,
   destination:zefd...sieokdf,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
   }
```



[FIGURE 1 Consensus system overview]

The structure of the entire system is shown in the figure above, and consists of verifiers (V1~Vn), account chain (A, B, C...), verification chain V, transaction units (S, R), verification units (V1, V2. ..).

D. Related concepts

Account chain: Each account has its own chain. All data in the chain is related to the account. Only the owner of the account can add numbers to the chain.

The trading unit of S(A,n) U R(A,n) as shown in the figure constitutes A's account chain.

Transaction unit: The transaction unit is divided into a transaction sending unit S and a transaction receiving unit R. A pair of S and R constitute a complete transaction. The transaction sending unit deducts the amount

```
Unit{

  version:0000...0001，
  type:send，
  balance:9999，
  previous:DC32...1CD1，
  destination:zefd...sieokdf，
  public_key:12ea...df94，
  signature:84ec...edf6，
  hash:57da...96c2，
}
```

from the sender's balance and the transaction receiving unit sends the transaction to the sending unit. The sending amount is added to the receiving unit.

[FIGURE: Transaction sending unit]

```
Unit{
   version:0000...0001,
   balance:9999,
   type:receive,
   previous:DC32...1CD1,
   source:32C1...5DB4,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
}
```

[FIGURE: Transaction receiving unit]

The structure of the sending and receiving units of the transaction is shown in detail above.

Version: the version number used for future upgrades, used in identification

Balance: The current account balance, which will be included in each unit point

Type: transaction type

Previous: The transaction unit of the previous current account

Destination: This field is valid when the type is a transaction sending unit and represents the address of the account to be sent to

Source: This field is valid when the type is a transaction receiving unit, indicating the transaction sending unit paired with the transaction receiving unit

Public_key, signature: Sign the unit to ensure that the unit is sent by the account owner

Hash: A hash operation is performed on the entire transaction unit to ensure that the transaction unit is not modified during network transmission

E. Verification chain

In order to serialize the DAG and to make the DAG form an unmodifiable stable point at the end, a verification chain was designed to assist the DAG in serialization and stability.

Several basic issues that need to be addressed in designing this chain.

1. Who is allowed to publish units on the verification chain?

Since the witness chain is one of the most important components of the maintenance of the entire system, the person who issues the unit on the verification chain should have a greater interest correlation with the entire system. Those who have such conditions and promise to maintain the system are the verifiers.

2. When to publish the unit on the verification chain?

The verifiers are responsible to maintain the stability and efficiency of the entire system. The relationship between verifiers needs to be cooperative not competitive, and should never result in internal conflict.

3. How to ensure that the right person publishes the right node at the right time?

Verifiers who have contributed to the system should be rewarded, while the ones that cause damage to the system will be removed. In addition, the system will heavily discourage those who are lazy or are unable to contribute to the system due to technical reasons.

4. This verification chain is so central, how to ensure decentralization?

Based on the above three issues, the following are the design goals for the verification chain:

1) The verifier needs to hold more tokens to ensure the interest correlation, and in order to enhance this correlation, the tokens needs to be held for a period before becoming effective.

2) The verification unit should have the verifiers take turns instead of competing.

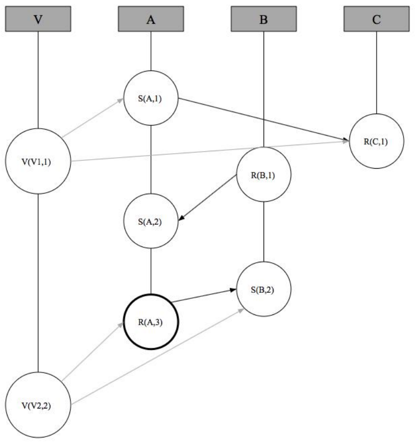
3) The verifier needs to provide the security deposit, which will be penalized when not responding in a timely fashion or being malicious. To encourage the verifier to verify more transactions, fees are rewarded to the verifier for verifying transactions.

4) The set of verifiers is dynamically changing, the number is variable, and the threshold is low to ensure the decentralization of the system.

After the 2 verification units after the deposit is paid, the verification group is automatically added. The verifier produces the block, verifies all the unverified transactions, and adds the hash value of the end unit that owns the chain of unverified transaction accounts. Then invites all the verifiers to vote, and if the vote is passed, the publisher obtains transaction fees and share the verification with all validators. Skip if not enough votes, and if a malicious verifier is detected at this time, all his or her security deposit will be penalized.

The following definition defines malice and punishment:

1. The verifier currently with the publishing unit rights did not fulfill its obligations and verified the illegal transactions.



[FIGURE 2 Malicious Verifier]

As shown in the figure, R(A,3) is an invalid unit point (eg, S(B,2) only transferred 10 tokens to him, R(A,3) said he received 20 tokens), But the verifier V2 thinks R(A,3) is fine/valid, and initiated the vote to validate R(A,3). In this case, the guarantor V2's deposit will be fully penalized due to his or her main action.

2. The verifier possessing the current publishing unit rights has issued two verification units in a row, or has issued units when not necessary.

Assuming that the unit issued by the verifier is not received during a time interval, then either:

a) The verifier does not act and does not fulfill its obligations.

Or

b) Due to network delay, no main action is detected.

The impact on the entire system is not significant. And if the problem persists, the system will automatically remove the verifier from the set of verifiers.

If the verification unit is issued at an inappropriate time disregarding the consensus, the verifier will be deemed malicious and punished by a penalty (lose the security deposit).

F. Verification unit

The verification unit is issued by the verifier in turn. Its main role is as follows:

1) Validation of all unverified transactions.

2) Collect votes for the last verification unit.

3) If the previous verification unit was malicious, issue a penalty. Its structure is as follows:

```
Verification{
   version:0000...0001,
   previous:DC32...1CD1,
   heigth:999,
   verify:...,
   punishment:...,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
}
```

Version: indicates the version number

Previous: points to the previous verification unit

Height: indicates the height of the current verification unit

Verify: Verified transactions and votes

Punishment: Proofs and penalties are given to non-voting verifiers

Public_key, signature: Name of the unit to ensure origin

Hash: A hash operation is performed on the entire unit to ensure that the transaction unit is not modified during network transmission

G. Related logic

The following conditions should be met before joining and leaving the collection of verifiers:

1) The verifier needs to have a certain amount of tokens to ensure that the Ambr system and the verifier are sufficiently relevant.

2) The verifier has to have the ability to maintain Ambr's system. In other words, have a high-performance server with public IP, which is online for a long time to ensure that Ambr can operate safely and stably.

Becoming a verifier is relatively simple. Simply add a request to join the verifier unit in the account chain (of course, need to pay the deposit), in the unit after the validation chain confirmation of the two verification unit after being added to the collection.

The specific unit structure is as follows:

```
Unit{
   version:0000...0001,
   type:JoinValidators,
   balance:9999,
   previous:DC32...1CD1,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
}
```

Similar to becoming a verifier, leaving the verifier set only requires adding a message to leave the set on the face of the account. After confirmation of two verification points, the set will no longer contain the verifier.

The specific unit structure is as follows:

```
Unit{
   version:0000...0001,
   type:LeaveValidators,
   balance:9999,
   previous:DC32...1CD1,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
}
```

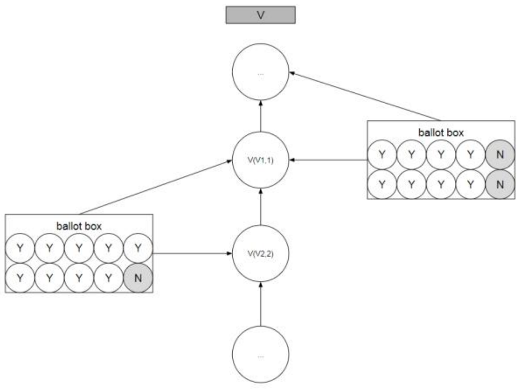
H. Verifier's Responsibilities

The responsibilities of the verifier are to maintain the safe, stable, and high-speed operation of the entire system so as to achieve the purpose of maintaining its own assets and gaining profits. In order to achieve these goals, the verifier should perform the following responsibilities according to the following rules.

When the verifier discovers a new verification unit on the verification chain during the term of office, the information is synchronized, and after the verifier determines the correctness of the verification unit, a vote of his own signature is cast. When issuing a verification unit, two things need to be done.

1. Collect a vote on the previous verification unit. If the previous verification unit did not receive enough votes and has evidence of being malicious, then collect evidence, punish the previous verification unit, put the result into the current verification unit, and wait for the vote to be reviewed.

2. Make judgments on the transactions received at present and act as witnesses for transactions that comply with requirements.



[FIGURE 3: Vote Chart]

Note: Since the system is decentralized, more people are needed as verifiers. However, the above figure shows that in order to punish evildoers, the evidence of wrongdoing must be kept for a period of time. However, storing such a huge amount of information can be a burden on the system. To alleviate this issue, all witnesses are encouraged to dynamically save newer voting information for use as evidence, and earlier voting information can be deleted.

I. Maintenance Ambr consensus principles and reasons:

Principle 1: Every time a verification unit is generated, the verifier should obtain a gain that is proportional to the margin.

Reason 1: In order to ensure the safety, stability and efficiency of the system, more people and funds are required to become verifiers.

Principle 2: The income of each verification unit should be prorated.

Reason 2: In order to ensure that the relationship between each verifier is cooperation rather than competition, thus saving social resources, and not forming an interest group for competition.

Principle 3: The more people and funds become verifiers, the higher the original verifier.

That is, the p % verifiers that participate in the consensus game will get f(p) ⩽ p % gains.

Reason 3: To ensure that the entire set of verifiers is developed and decentralized, and not controlled by a few interest groups.

Principle 4: Those who act maliciously in the main actions will be severely punished. Those who do nothing are also considered to be malicious and will also need to receive punishment.

Reason 4: Being malicious can be not recognizing history or verifying illegal transactions. Doing nothing is also considered malicious, for example, not releasing the verification unit on time, and not voting.

Principle 5: More votes equals more earnings.

That is, the p % verifiers that participate in the vote will get f(p) ⩽ p % of the return. If 100% of the people participate in the vote, the gain is 100%.

Reason 5: Try to ensure that all verifiers participate in the voting, and punish those who do not.

V. Smart contract

In some high-level language systems (such as java and .net), for the sake of high platform compatibility and portability, and high running performance, most introduce an intermediate encoding, assembly language, and then the execution is interpreted by the virtual machine. Like Ethereum's Solidity, RChain's RHO, and ECO's WASM.

After comprehensive evaluation of various platforms, Ambr decided to use the Solidity language as a contract writing language. The simple, proven and expressive language and has accumulated a large number of community developers due to the use of the Ethereum community. With the powerful JIT interpreter developed by the C language to interpret the execution, the performance functionality is very high.

And can use the existing development tools of the community to make the contract preparation easier and more convenient.

A. Virtual machine

AVM (Ambr Virtual Machine - AVM) Based on the stack structure, the machine's word size (and the size of the data in the stack) is 256 bits. Mainly to facilitate the implementation of Keccak-256 hash and elliptic curve calculations. The memory model is based on word addressed byte data. The running model is based on a thread stack with a maximum stack depth of 1024. AVM also has a separate memory model; similar to memory but more like a byte array, a word-based word array. Unlike variable memory, storage is non-volatile and is maintained as part of the system state. All memory and storage data is initialized to 0. AVM is not a standard Von Neumann structure. The program code is saved in a virtual, interactive ROM through a special instruction, instead of in general accessible memory or storage. AVM can cause exceptions in some situations, including stack overflows and illegal instructions. When an exception occurs, AVM immediately stops and informs the execution agent (the transaction's processor, or execution environment) to handle the exception separately.

Bytecode

The bytecode can be considered as the instruction code of the virtual machine, or an intermediate code designed to speed up the execution of the runtime. Similar to assembly, these virtual instruction and will not be directly on the CPU, and running requires AVM to execute.

The following are some advantages:

- Reduced file size
- Faster generating function prototypes
- Increase the difficulty of being cracked
- Slight optimization of the source code

B. Features of smart contracts

The smart contract was introduced by Ethereum, and runs on a distributed system that uses p2p network communication. This running platform gives these codes the characteristics of immutability, determinism, distributed and self-checking. The storage of state during code execution is immutable. Everyone can open a private node, replay the entire system, and get the same results.

In Ambr, each contract has a unique address to identify itself. The client can interact with this address, send and receive Ambr coins, call functions, query the current status, and so on.

Smart contracts are essentially two elements: the code, and the state stored in the blockchain after the code is run.

C. Turing Complete

In computability theory, a Turing complete system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is one that can be used to simulate a single-band Turing machine. Although physical storage capacity in an limitation, Turing completeness usually refers to a "universal physical machine or programming language with unlimited storage capacity," and Solidity is a Turing-complete language.

D. Transaction fee (Gas)

In order to motivate the nodes in the network to actively participate in the calculation, on the other hand, in order to avoid wasting of resources on malicious codes, each piece of code needs to provide "Gas" charges when computing in the network.

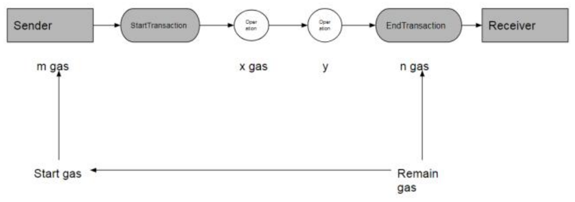
Fees charged are reflected in the following three areas:

1) The universal instruction calculates the fee. When using the computational resources (CPU, GPU, etc.) in the network, the bytecode instruction will determine the price;

2) Fees for calling sub-contracts and creating contracts;

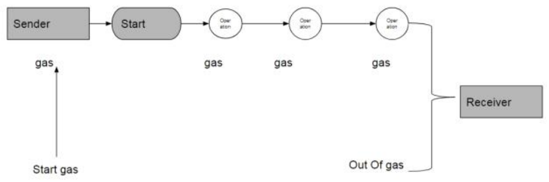
3) Increased memory usage in dynamic use. For the execution of an account, the price is in multiples of 32 bytes, and memory access and addressing are also based on 32-byte alignment. Accessing an address larger than 32 bytes of the index will require additional memory usage. The address can easily exceed the 32-byte limit.

To sum up, these possible events must be managed. Storage costs have another purpose - in order to minimize storage costs (directly with a state data communication on all nodes), clearing the stored execution cost instructions is not only recommended, but will also return some expenses; because of the initial storage often costs more than actually used, a return fee will be issued after the storage space is cleared, this returned fee is part of the pre-paid fee.



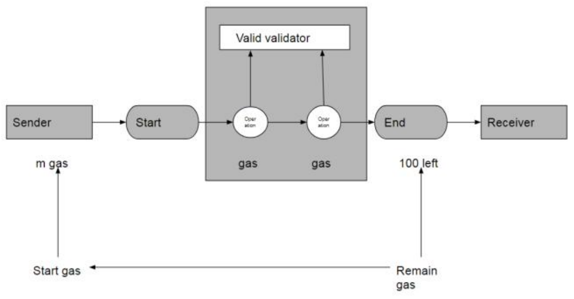
[FIGURE 4: Fuel usage process]

When the sender sends enough Gas to execute the contract, excess fuel will be returned to the sender, and simultaneously synchronize execution structure to recipient account status.



[FIGURE 5: Exhausted fuel]

When the sender does not send enough gas to pay for running this contract, the virtual machine will run the abort instruction and contract.



[FIGURE 6: Transaction fees]

The fuel consumed in the above chart is the processing fee, and will eventually be sent to the verifier.

```
Unit{
   version:0000...0001,
   language:solidity
   type:contract,
   previous:DC32...1CD1,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
   creator:43ef...999,
   entropy: 2334,
   entropyprice: 2344,
   value: “233“,  //ambr number
   initcode: “abef”,
   contract: “efdae...ef32”,

   //this is optional
   metadata: {
      sources:
      {
         “filename.sol”: {
            “keccak256”: “0x123…”,
            “location”: “ef23….”,
         },
         “filename2.sol”: {
          “keccak256”: “0x123…”,
            “content”: “something….”,
         },
      },
       //defines the varients for EVM
       settings:
       {
          libraries: {
             “lib1”: “hashaddress”,
             “lib2”: “hashaddress”,
          },
          target: {
             “filename”: “myfile.exe”,
             “filename2”:
“myfile2.exe”,
          },
          parameters: {
             “optimize”: “true”,
         },
       },
       output:
```

E. Contract introduction

Ambr's smart contract is deployed on the DAG and is synchronized to other nodes in the network through consensus protocols and p2p networks. This system allows contracts to be executed correctly on every complete node.

Smart contracts are written in the Solidity language and then compiled by the compiler into a binary bytecode file. Finally, the AVM interpretation of the respective node is performed.

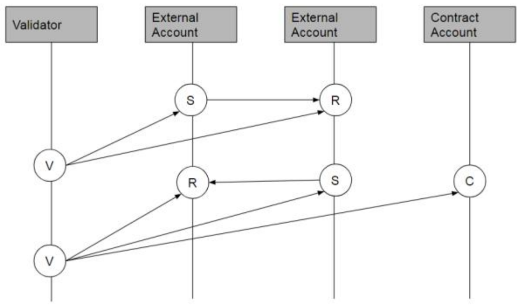
Smart contract example:

The smart contract is compiled, then sent to the network for storage. A new contract account will be generated for the smart contract, and then a transaction node will be

```
        {
            abi: “”,
            doc: “”,
         },
      },

    },
 }
```

added based on the account, as shown in the following figure.



[FIGURE 7: Contract creation process]

[FIGURE: Unit{}] Activation contract

The storage structure is as follows

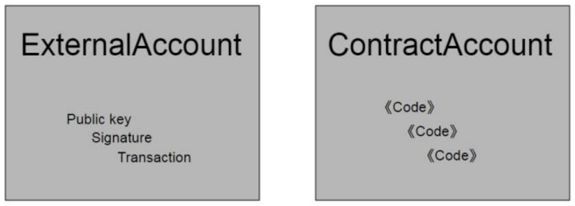
```
params:[{
    "from": "0xxxxxxx",
    "to": "0x33333",
    "entropy": 4455555,
    "entropyPrice": 234524,
    "value": "xxxxx",
    "data": "0x3ader33",
}]
```

F. Accounts and Status

Ambr has 2 built-in accounts:

External Account: has a private key

Contract Account: owns the code



[FIGURE 8: Account and status]

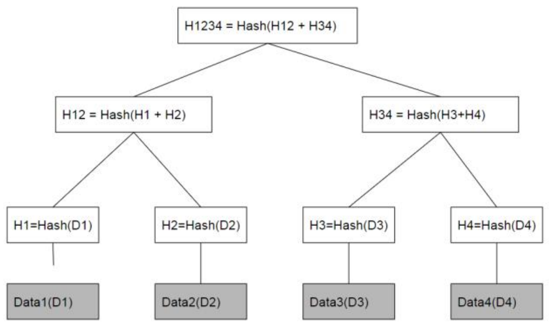
An external account has a private key, can own a coin, and can send a message to another externally owned or contracted account by creating and using its own private key to sign the transaction. The message sent between two externally owned accounts is just a simple value transfer. Also the external account can send messages, trade to the contract account, thus activating the contract account code, allowing the performance of various actions. (Such as transferring tokens, writing to internal storage, dug out a new token, performing some calculations, creating a new contract, etc.)

A contract account cannot initiate a transaction on its own. Instead, the contract account only triggers a transaction in response to the transaction after receiving a transaction (from an externally owned account or another contract account). An asynchronous messaging system will be explained in the "Asynchronous Message Drivers" section to show how to achieve communication between contracts.

In the system, all transactions and messaging are initiated by external accounts.

World state

In Ambr's verification chain global state machine, the status is a Merkle tree structure consisting of addresses and corresponding states.



[FIGURE 9: Merkle Tree]

As shown in the above figure, the Merkle tree is a binary tree consisting of a root node, a set of intermediate nodes, and a set of leaf nodes. The lowermost leaf node contains the stored data or its hash value, each intermediate node is the hash value of its two children's nodes, and the root node consists of the hash values of its two child nodes.

For an ordinary external account, every time the status is the account of the parties involved in the transaction, the main account, the address of the issued account, and the corresponding status (balance, etc.) are all stored. The contract account must save more data, including:

1. The details of the account of the departure transaction,

2. The status of the execution result of the contract account,

3. The status result of the influence of the contract code,

4. Status Result of Fee Recipient Account

The Merkel tree is characterized by any change in the underlying data that is passed to its parent node, up to the root of the tree.

Typical application scenarios of the Merkel tree include:

Quickly compare large amounts of data: Two Merkel roots being the same means that the data represented must also be the same.

Fast way to track modifications: In the above example, if the data in D1 is modified, N1, N4, and Root will be affected. Therefore, using Root -> N4 -> N1, the modification to D1 can be seem easily;

Zero-knowledge proofs: how to prove a data, for example, (D0...D3) includes the given content D0. Construct a Merkel tree and publish 0, N1, N4, Root. D0 owners can easily detect the existence of D0, but do not know about other content.

According to the above statement, the Merkle tree can be used in the SPV light node wallet, which can reduce the download amount. On the basis of no need to download all the data, the account information data can be easily compared and verified, and data transmission can be reduced when the P2P network is synchronized.

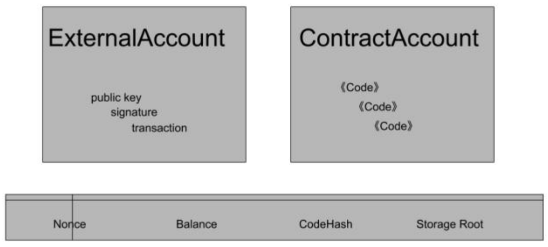
Account Status

Nonce: If the account is an externally owned account, nonce represents the transaction number sent from this account's address. If the account is a contract account, nonce represents the contract sequence created for this account number.

Balance: account balance

StorageRoot: The root hash value of the Merkle tree. The Merkle tree encodes the Hash value of this account's stored content. The default is the null

CodeHash: the hash value of this account code. For a contract account, this is the code that is hashed and saved as a CodeHash. For externally owned accounts, the CodeHash domain is an empty string Hash value.



[FIGURE 10: Account Composition]

Asynchronous message driver

Ambr uses RPC to communicate and the data content format uses the Json format.

RPC (remote process call), name remote procedure call, is two servers with different physical locations. One application of the server wants to call a function or method of an application on another server. The apps are not in the same memory space, and cannot be called directly. Therefore, express the semantics and incoming parameters through the network is necessary.

RPC is a cross-operating, cross-programming language network communication method. The method specifies that parameters and return values are serialized into binary data during network transmission. This process is called serialization or marshalling. The serialized binary is sent to another server via addressing and transmission. Another server receives the binary data and deserializes it, reverts to the memory expression, and then finds the corresponding method call. The returned value is still returned to the first server in binary form, and then the deserialized read return value.

The Ambr project uses Google's GRPC framework, which is based on http2 for powerful performance and extensibility. The Json format is in plain text format for easy reading and understanding.

In the Bitcoin implementation, a set of JsonRPC is used. Compared with JsonRPC, GRPC has Google's strong community support, and is based on Http2 and has a unified implementation of various platforms, especially between various devices. The seamless interaction, network reuse of Http2, enables the GRPC to reuse the existing TCP connection when frequently called many times, which greatly improves the efficiency. Because the establishment and release of TCP connections is very resource-consuming. GRPC also provides more powerful support for various data formats, like Json, ProtoBuffer, XML, etc., being crucial to the upgrade, maintenance and expansion of subsequent systems.

Contract creation

In Ambr's system, creating a contract essentially means creating a special transaction whose content is a piece of code whose purpose is actually to create a new contract account.

The process of creating a contract account in Ambr is to use a special formula to declare the address of the new account, and then use the following method to initialize an account:

- Set nonce to 0
- If the sender sends a certain amount of Ambr as the initial balance of the account through the transaction
- Set storage to 0
- Set the contract's CodeHash to an empty string Hash value

After the account is initialized, the contract is initialized using the Init code attached to the transaction. The execution of Init code is various and can, depending on the constructor, update the account's storage, or create of another contract account, etc.

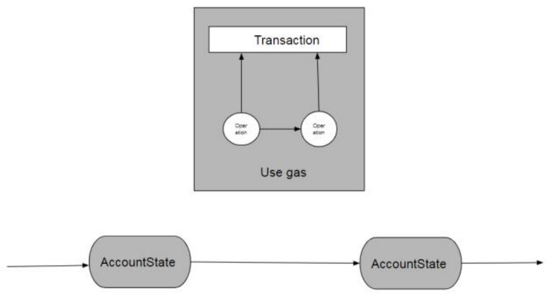
Gas is used when the code of the initialization contract is executed. The transaction does not allow the gas to exceed the remaining gas. When using gas that exceeds the remaining gas, an out of gas anomaly (OOG) occurs and exits. If a trade exits due to a lack of gas, the status will immediately return to a point before trading. The sender also does not get a refund of the gas spent before running out.

However, if the sender sends Ambr as the transaction (see Json message structure Unit-Value), even if the contract is created, Ambr will be returned. If the successful execution of the initialization code is completed, the cost of the final contract creation will be paid. These are storage costs that are proportional to the size of the contract code that is created (again, no free lunch). If the amount of gas is insufficient, the transaction will once again announce a gas shortage and interrupt the exit.

If nothing happens without any exception, any remaining unused gas will be returned to the original transaction sender, and the changed status is now allowed to be permanently saved.

Execution model

Execution of the transaction:



[FIGURE 11: Execution of transactions]

First, the virtual machine performs a series of pre-verifications before executing the transaction:

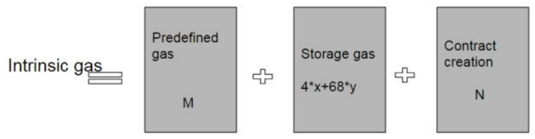
- The transaction code is a valid instruction set that can be executed in compliance with the predefined rules
- Valid transaction signature
- The valid transaction number. The nonce in the account is the count of transactions sent from this account. The transaction number must be equal to the nonce in the sending account

A transaction’s estimated gas must be equal to or greater than the intrinsic gas used by the transaction. The intrinsic gas includes:

1) Execute transaction booking fee

2) The gas cost of the data sent with the transaction (the cost per byte of data or code 0 is 4gas, and the data or code cost per non-zero byte is 68gas)

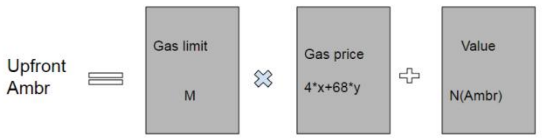
3) If a contract is being created, additional contract creation gas is also required



[FIGURE 12: Intrinsic Fuel]

The balance of the sending account must have enough Ambr to pay for the "pre-" gas charges. The calculation of the gas cost in the early stage is relatively simple.

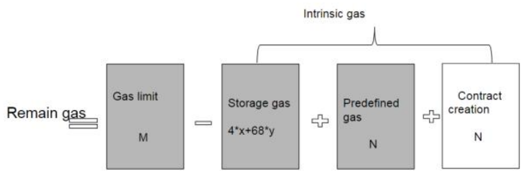
First, the gas cost of the transaction is multiplied by the gas price of the transaction to get the maximum gas cost. Then, the maximum gas cost plus the total value sent from the sender to the receiver.



[FIGURE 13: Front Cost]

If the transaction meets all the above requirements, then proceed with the following steps.

1) Deduct the upfront cost of execution from the sender's balance and add 1 to the nonce of the sender's account for the current transaction. At this point, calculate the remaining gas and subtract the intrinsic gas used from the total gas traded.



[FIGURE 14: Fuel Surplus]

2) Start executing the transaction. The system will keep track of "sub-states" throughout the transaction execution. The sub-state is a way of recording the information generated in the transaction and is immediately needed when the transaction is completed.

Specifically:

- Self-destroyed sets: sets of accounts (if any) that are discarded after the transaction is completed
- Log Series: archiving and retrievable checks of code execution for virtual machines
- Point refund balance: The total amount that needs to be returned to the sending account after the transaction is completed. Recall that the storage in the previous system requires payment, and the sender will have a refund if the memory is cleared. Use refund counts to track refund balances. The refund count starts at 0 and increases each time the contract deletes something in the store.

3) Various calculations required for the transaction begin to be processed. When all the steps required for the transaction are completed and no invalid state has occured, the final state is determined by determining the amount of unused gas that is returned to the sender.

Once the sender gets a refund:

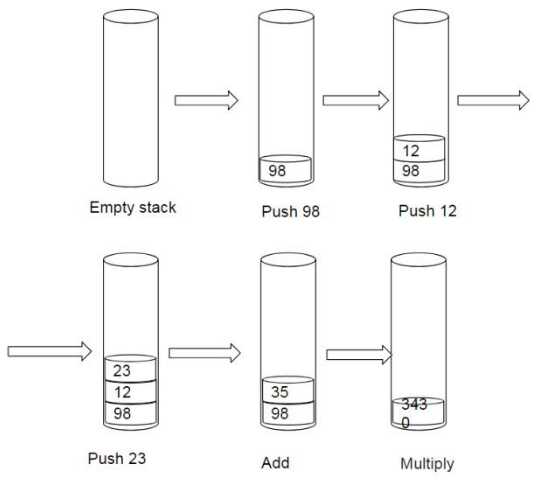
- Gas costs will be sent to the verifier
- The gas used by the transaction will be added to the gas count of the block (The count keeps track of the total amount of gas used by all transactions in the current block, which is very useful for verifying blocks.)
- All accounts in the self-destructed group (if any) will be deleted
- A new state and a series of logs created by the transaction.

In summary, these constitute the state machine of Ambr.

AVM execution model and memory layout

Above was the basics of transaction execution, the following looks at some of the differences between contract creation transactions and message communications. A series of steps must be followed to execute the transaction from the beginning to the end. Here is how the transaction is executed in Ambr's Virtual Machine (AVM).

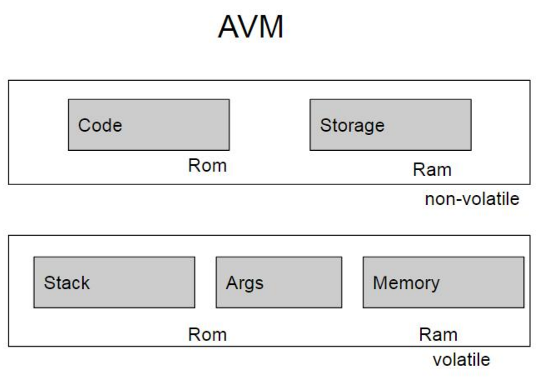
The actual transaction processing part of the protocol is run in the virtual machine, and the virtual machine interprets and executes. EVM is the Turing complete virtual machine. The only limitation is that the AVM exists but a typical Turing complete machine does not exist because the AVM is inherently bound by gas. Therefore, the total amount of calculations that can be completed is essentially limited by the total amount of gas supplied.



[FIGURE 15: Instruction execution]

In addition, AVM has a stack-based architecture. A stack machine is a computer that uses the last-in first-out to save temporary values. Each stack item in AVM has a size of 256 bits and the stack has a maximum size of 1024 bits. The EVM has memory and items are stored in an array of addressable bytes. Memory is volatile, that is, data is not persistent.

AVM also has a memory. Unlike EVM, AVM is non-volatile and is maintained as part of the system state. AVM saves program code separately and can only be accessed in the virtual ROM by special instructions. In this case, AVM differs from the typical von Neumann architecture in that the program's code is stored in memory or memory.



[FIGURE 16: Memory Model]

AVM also has its own language: "AVM bytecode". A programmer usually uses a high-level language to encode. However, in order to be efficient when the code is running, a compiled byte code is needed. The code is usually written in a high-level language such as Solidity, then compiled into AVM bytecode that AVM can understand.

Implementation

Before performing a specific calculation, the processor determines whether the following information is valid and accessible:

System status:

- The remaining gas used for the calculation
- Account address with execution code
- The address of the sender of the transaction that originally triggered this execution
- The address of the account that triggered the code execution (may be different from the original sender)
- The price of the gas triggering this execution
- The input data for this execution
- Value is passed to this account as part of the current execution
- Machine code to be executed
- Block header of the current block
- At the beginning of the deep execution of the current message communication or contract creation stack, the memory and stack are empty, and the program counter is 0

1PC: 0 STACK: [] MEM: [], STORAGE: {}

AVM then begins recursive execution of the transaction, calculating the system state and machine state for each cycle. The system state is also the global state of Ambr.

Machine status includes:

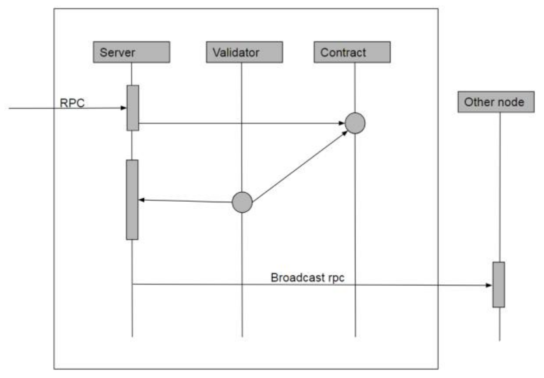
- Available gas
- Program counter
- Content of memory
- The number of active words in memory
- The contents of the stack
- Items in the stack are deleted or added from the far left of the series
- Each cycle, the remaining gas will be reduced by the corresponding amount, and the program counter will increase.
- At the end of each cycle, three scenarios are possible:
  1. The machine reached an abnormal state (such as insufficient gas, invalid instruction, insufficient stack entry, stack item overflow, invalid JUMP/JUMPI destination, etc.) so stop and discard all changes

2. Enter the next cycle of subsequent processing
3. The machine reaches a controlled stop (at the end of the execution process). Assuming that the execution does not encounter an abnormal state, a "controllable" or normal stop is reached and the machine will produce a composite state, the remaining gas after the execution, the resulting child status, and combined output.

VI. DAG and contract

Contract creation

The contract is created by the user to send a contract creation message, then create a contract account using this message, and then initialize the account with the initialization code contained in the message. If successful, the account will be synchronized to other distributions through the p2p network nodes.



[FIGURE 17: Contract process]

```
Unit{
   version:0000...0001,
   language:solidity
   type:contract,
   public_key:12ea...df94,
   signature:84ec...edf6,
   hash:57da...96c2,
   creator:43ef...999,
```

[FIGURE: Unit{}] Contract creation message

The above represents a contract creation instruction in json format.

Version: the version number used for future upgrades.

Language: The contract programming language, because the virtual machine executes the bytecode, so theoretically as long as the corresponding compiler is developed, any language will be supported.

Type: Contract type

Public_key: The public key of the creator account, used to verify the validity of the data

Signature: Unit signature

Hash: The sha256 string of the unit structure

Creator: The creator's account address

Gas: Estimated gas value for the contract

Gasprice: The price of gas that the sender is willing to pay (Ambr measurement)

Value: The number of Ambr

Initcode: Initial account code

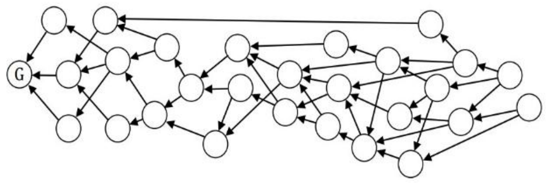
Contract: Solidity binary code of compiled smart contract bytecode

Consistency issues for DAG+ contracts

```
entropy: 2334,
 entropyprice: 2344,
 value: “233“， //ambr number
 initcode: “abef”,
```

Some products are based on DAG technology, such as IOTA and Byteball, but none support smart contracts. Because of a problem with the DAG architecture itself - no guarantees of transaction status, atomicity or unity. In terms of time, specific nodes (such as remote nodes) may only exist temporarily. So the time for confirming a transaction cannot be estimated.

From the node point of view, a node in the entire network node may not be able to update the transaction information at a certain moment, that is, the node is not broadcasted to the transaction information at a certain moment. These conditions are problematic for many of the cryptocurrency platforms out there.



[FIGURE 18: Traditional DAG]

As shown in the above chart, all trading units in the DAG structure are ordinary trading units. Normal cells contain signatures, transactions, parental hash values, and each transaction is a delayed acknowledgment due to the uncertainty in the architecture implementation, namely the uncertainty of the transaction confirmation. Nodes are not fully synchronization due to network delays, and so confirmation is done by subsequent nodes. But because the insertion order of the subsequent nodes on different network nodes cannot be guaranteed, the implementation of the smart contract becomes a very difficult challenge.

However, Ambr has successfully responded to this challenge and solved the problem. Ambr introduced the DAG architecture based on the multi-chain structure of accounts, making the basic units of participants in the network independent from the accounts. At the same time, a validation chain was introduced and Casper was adopted. The algorithmic election succeeded in solving the above-mentioned uncertainties, realized smart contracts, and keeps the smart contracts and common trading solutions consistent.

Contract verification

The consensus agreement relies on parallel verification chains to assist in transaction confirmations.

When a smart contract is created, an account chain with one node is formed in the DAG graph. After the node is created, the verification is done by the verifier on the current verification chain and generated. A new DAG node, at this time, will be broadcasting to other network nodes according to the rules defined by the consensus protocol. After the broadcast message is sent to other nodes through the p2p network, the node will get the public_key, unithash, and signature inside the message for verification. If the verification passes, the node will be inserted into the DAG, and the contract is verified.

The verification chain contract verification rule obtains the hash value of sha256 by calculating the key field of unit, then compare the unithash fields and check for modifications. Apply public ellipse to public_key to decrypt the signature and then compare with the unithash field to check if the sender is trusted.

Through the creator value, the balance of the creator account is obtained, and the value of the received message is compared. The system then judges whether or not the user has a sufficient balance, and the remove the contract code of the account if funds are insufficient. Next, the rule integrity verification is performed on the code to see if the instructions are valid for a contract.

VII. Decoupling protocol

The blockage of the Bitcoin network has caused non-essential users to avoid transferring funds to Bitcoin on the chain as much as possible. Ethereum also had a game application that resulted in a serious blockage of transactions.

People today expect things to be instant and without delay. However, transaction speeds of traditional decentralized blockchains ranges from a few seconds to a few days, and is a severe limiting factor.

Bitcoin and Ethereum have separately delisted protocols such as Lightning Networks and Status Channels for immediate access and large-scale concurrent demand support.

The basic logic of Lightning Network and Status Channel is as follows

1) A, B respectively put funds into the pool of funds on the chain.

2) Trading at high frequency under the A and B chains requires A, B double signatures each time, and the serial number attached to the transaction is incremented by one at a time.

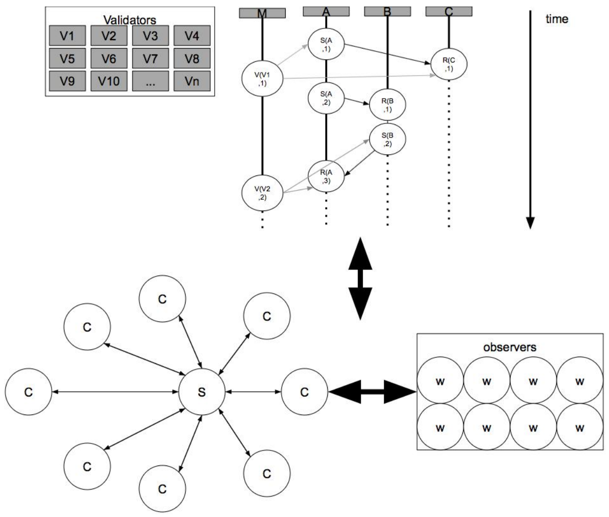
3) If either A or B wants to close the channel, the highest serial number transaction evidence signed by both parties is needed. If the other party does not provide a higher serial number (that is, a later consensus) within a certain period of time, then follow the highest transaction serial number and allocate funds for the pool.

4) Multiple parties involved in the transaction will be connected in pairs to form a long channel for trading.

The advantage of this agreement is that the process of frequent transactions will be performed out of the main chain, thereby reducing the pressure from the amount of transactions, leading to higher transaction speeds. However, the disadvantages of this agreement are also quite obvious:

1) When one party closes the trading channel, the other party must observe whether the evidence provided by the other party is the highest transaction number within a certain period of time. If not, then corresponding evidence must be provided. Otherwise, the evidence provided by the other party will be used, which can result in a loss.

2) Another very important use scenario for high-frequency transactions is divergent high-frequency trading around businesses. The lightning network and status channel support for this type of use scene is insufficient. To this end, Ambr has designed the decoupling protocol as follows:



[FIGURE 19: Unlinking transaction diagram]

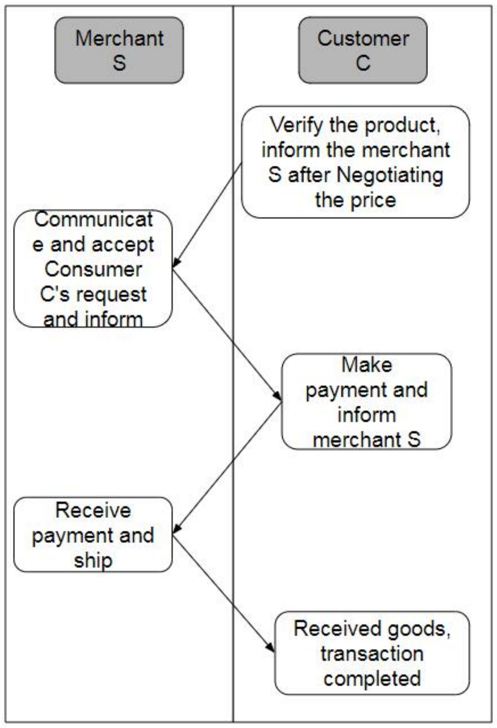
1) The observer is added. When the channel is established, a party can pay the observer a small fee to help while he or she is offline. The observer can perform proofs on the chain to ensure the channel's interests.

2) Merchants may be serving different customers at the same time. If discovered cheating, all the security deposits in all channels will be penalized.

Through the decoupling protocol, a large number of high-frequency small transactions can be put out of the chain, and only the initial and final results are saved on the chain, thereby greatly extending the throughput of the system.

Assume a scenario in which two people who do not know each other conduct transactions and need to go through processes such as communication, payment, and delivery. Each process has evidence on both sides, and the law requires that in the event of a disagreement, sufficient evidence must be provided. And unless the other party provided a more valid proof, then proceed to a judgment in the court according to the claim of one party's evidence.

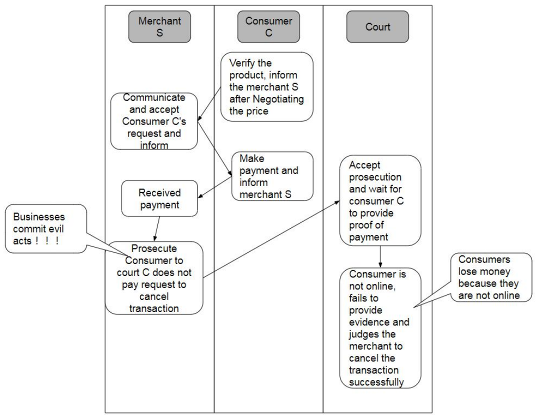
The normal transaction process is shown in the figure below.



Merchant S
Customer C
Verify the product, inform the merchant S after Negotiating the price
Communicate and accept Consumer C's request and inform
Make payment and inform merchant S
Receive payment and ship
Received goods, transaction completed

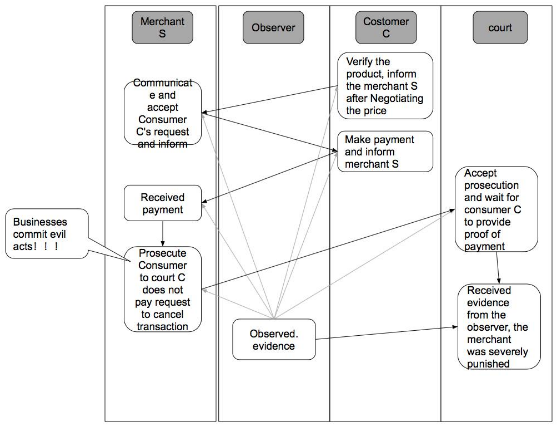
[FIGURE 20: Normal trading diagram]

Consumers are more likely to buy cheaper things, and after the purchase, will not be online to follow the transaction. At this time, the business can be malicious, as shown in the following figure.


Merchant S
Consumer C
Court
Verify the product, inform the merchant S after Negotiating the price
Communicate and accept Consumer C's request and inform
Make payment and inform merchant S
Received payment
Accept prosecution and wait for consumer C to provide proof of payment
Businesses commit evil acts ! ! !
Prosecute Consumer to court C does not pay request to cancel transaction
Consumer is not online, fails to provide evidence and judges the merchant to cancel the transaction successfully
Consumers lose money because they are not online

[FIGURE 21: Business being malicious]

Under these circumstance, in order to protect personal interests while offline, the consumers need to have someone observe the details of the transactions to help determine whether or not the business is being malicious.


Merchant S
Observer
Costomer C
court
Verify the product, inform the merchant S after Negotiating the price
Communicate and accept Consumer C's request and inform
Make payment and inform merchant S
Received payment
Accept prosecution and wait for consumer C to provide proof of payment
Businesses commit evil acts! ! !
Prosecute Consumer to court C does not pay request to cancel transaction
Observed. evidence
Received evidence from the observer, the merchant was severely punished

[FIGURE 22: Business being malicious, plus observer]

In summary, the interests of both parties in the high-frequency de-chaining transaction can be guaranteed by increasing the observer and the penalty for the fraud.

VIII. Cross-chain protocol

Since 17 years ago, with a large number of blockchain projects being developed and promoted, more and more people and funds have been invested in these projects.

The number of blockchain projects of various sizes and types is now numerous, but along this development path, a series of problems have also emerged. For example, the number of blockchain projects is growing, but communication in between is nonexistent. As a result, each of the blockchains is a separate "island of information", which greatly limits the application space of the blockchain. Therefore, effective cross-chain technology is needed to solve these problems.

Cross-chain technology can save a large number of blockchain projects from being blind, and build a bridge to allow communication with each other. Cross-chain technology is a way to expand the blockchain, which largely determines the upper limit of the development of blockchain projects. The cross-chain technology used by Ambr is designed based on security, efficiency, and difficulty of implementation.

About the history of cross-chain technology: Early cross-chain technology focused more on asset transfer, represented by Ripple and BTC Relay; existing cross-chain technology focuses more on cross-chain infrastructure, represented by Polkadot and Cosmos. The newly emerging FUSION has realized multi-currency smart contracts, and can produce multiple cross-linked financial transactions in the market.

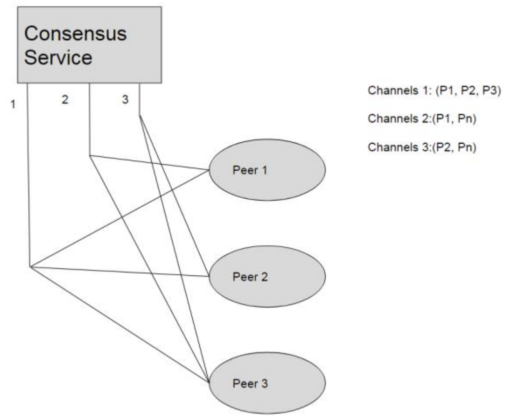
A. Notary Mechanism

The main representative is the Ripple Interledger protocol, which applies to all bookkeeping systems and aims to achieve a globally uniform payment standard.

B. Corda

Corda is a "blockchain" technology architecture. Both sides of the transaction jointly select a notary with high degree of feasibility so that the notary can verify the validity and uniqueness of the data. If the notary confirms that the deal is feasible, the deal will be reached. At this point the transaction notary's books will be synchronized. The advantage of doing this is to make the transaction more efficient under the conditions of ensuring security.

Fabric: The newly defined concepts include chain, peer, channel, and consensus services. Peers can participate in multiple books so that the fabric has scalability, peer transaction isolation, account isolation and etc.



[FIGURE 23: Fabric Channel]

C. Cosmos

Cosmos: (Interchain Foundation's cross-chain open source project) focused on solving cross-chain asset transfer. The blockchain network is mainly composed of Zone and Hub.

1) The Cosmos Zone is a separate blockchain space.

2) The Cosmos Hub Center is a multi-asset equity certificate cryptocurrency network.

The Hub is a relay chain that is debited by a decentralized certifier group. A Hub communicates with multiple Zones. Each Hub has billing information for multiple associated Zones to generate a multiple asset center book. Hub guarantees asset transfer at different zones

In the process, the total assets inside remain unchanged.

The Cosmos process is as follows:

First, the cross-link communication between Hub and Zone is achieved by the IBC protocol. Assume that Zone1 wants to perform cross-chain transactions with Zone2.

1) Zone1 generates and publishes transaction information on the Hub.

2) The Hub generates and publishes a proof of the presence of Zone1's cross-chain packets on Zone2.

3) Zone2 receives the message package and publishes the received certification information on the Hub.

4) The Hub gives proof that Zone2 has received the proof and sends the message on Zone2.

The advantages of Cosmos include the following:

1) The transfer of tokens in each Zone will be through the connected Hubs, so assets in each Zone will be recorded.

2) If a Zone fails, other valid Zones will not be affected

3) The newly added Zone can easily be added to the Hub Center

D. Cross-chain transactions

Cross-chain technology can save a large number of blockchain projects from being blind, and build a bridge to allow communication with each other. Cross-chain technology is a way to expand the blockchain, which largely determines the upper limit of the development of blockchain projects.

Regarding cross-chain technology, the current mainstream cross-chain technologies include: Notary mechanism (Notary Schemes), Sidechains/relays, hash lock (Hash-locking) and distributed private key control. The main cross-chain technology differences are as follows:

Cross-chain technology	Notary technology	Relay and side chain technology	Hash-locking	Distributed Key Control Technology
Interoperability	All	All	Only cross-dependence	All
Trust Model	A variety of notary public honest	Chain will not fail or be "51% attacked"	Chain will not fail or be "51% attacked"	Chain will not fail or be "51% attacked"
Applicable for cross-chain exchange	Support	Support	Support	Support
Applicable to cross-chain asset transfer	Support	Support	Not Support	Support
Applicable for cross-chain Oracles	Support	Support	Not directly supported	Support
Applicable for cross-chain asset mortgage	Support	Support	Most support but with difficulty	Support
Realization difficulty	Medium	Difficult	Easy	Medium
Multi-currency smart contracts	Difficult	Difficult	Not support	Support
Implementation case	Ripple	BTC Relay/Polkadot/COSMOS	Lightning network	Wanchain/FUSION

E. Ambr cross-chain transactions

Ambr's roadmap includes the use of relay technology and future protocols such as Polkadot and Cosmos to support cross-chain transactions between different cryptocurrencies. The cross-chain technology adopted by Ambr is designed based on the difficulty of achieving security and high efficiency.

The core technology of the Ambr cross-chain protocol is the Ambr relay chain. This technology enables Ambr to not only have the scalability and scalability of Polkadot, but also have the characteristics of Cosmos compatible future blocks.

Ambr's cross-chain protocol supports cross-link transactions between different currencies. Users will be able to make transactions between Ambr coins and BTC, ETC, ZCash and other tokens, breaking the barriers between exchanging currencies. The Ambr relay chain plays a huge role in facilitating transaction and exchange between different blockchain currencies, and provides very novel technical and ideological guidance to emerging blockchain related companies.

Ambr cross-chain protocol design principles:

1) **Security**: This is the centerpiece of Ambr's cross-chain design, and Ambr must have absolute security while implementing cross-links. The historical data generated during the cross-connection process is extremely difficult to modify.

2) **Performance**: The efficiency of cross-linking is also an important consideration for Ambr. Under the condition of ensuring security, increase Ambr's throughput and the speed of cross-chain confirmation. In other words, as the number of transaction transactions handled by Ambr across the chain grows, users enjoy a better trading experience.

The Ambr relay chain is designed using a cross-chain protocol similar to the Polkadot relay chain and Cosmos. The Ambr relay chain mainly plays a role in recording the transaction address and transaction amount and verifying the validity of the transaction. Multiple trading units exist on a trading chain. Each cross-chain transaction must be recorded and verified by at least one transaction unit. Contracts generated in each foreign currency that are linked to the Ambr relay chain must be reviewed by the relay chain relay unit. And the transaction address of a node in each foreign currency must have a mapped address on the relay chain. Each transaction amount is stored in the relay unit in the relay chain. The relay chain provides technical and platform support for solving cross-chain transactions in Ambr and foreign currencies, while also providing space and opportunities for cross-chain transactions between different foreign currencies.

```
Transfer
{
    version:0000...0001,
    previous:DC32...1CD1,
    height:999,
    verify:...,
    punishment:...,
    direction: output/input,
    sourcelink:Ambr,
    targetlink:ETH,
    amount:99ETH,
    public_key:12ea...df94,
    signature:84ec...edf6,
    hash:57da...96c2,
}
```

Version: indicates the version number

Previous: points to the previous transfer unit

Height: indicates the height of the current transfer unit

Verify: verified transactions and votes

Direction: indicates direction, sub-output and input

Sourcelink: indicates the source chain. Input can be ETH, BTC, etc. Output can only be Ambr

Targetlink: Indicates the target chain. Output can be ETH, BTC, etc. Input can only be Ambr

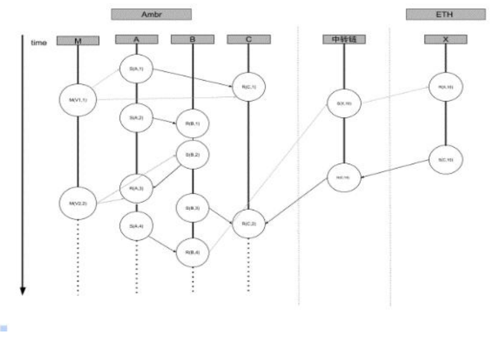
Amount: indicates the amount, in a defined unit

Punishment: proofs and penalties are given to non-voting verifiers

Public_key, signature: sign the unit to ensure the origin is the verifier

Hash: a hash operation is performed on the entire unit to ensure that the transaction unit is not modified during network transmission

Example of cross-chain trading between Ambr and Ethereum (ETH):



[FIGURE 24: Ambr and ETH cross-chain transactions]

Input transaction:

1) Suppose an account on Ethereum X needs to pay 10 ETH to Ambr account C. X needs to apply for a transit contract on Ethereum. The transit contract should contain the Amber account C address and the amount sent, signed by account C.

2) The relay chain actively monitors that ETH has an Ambr account-related contract.

3) A transfer unit in the relay chain records the review of the contract and is verified by the verifier's collection of numbers.

4) If the digital verification is passed, record of account C is added to the transaction and the 10 ETH is locked and disabled in ETH; if not, the contract is void and the transaction fails.

5) Finally, a verification unit in the AMBR verification chain authenticates the end node of account C and achieves AMBR consensus on the entire network.

Output transactions:

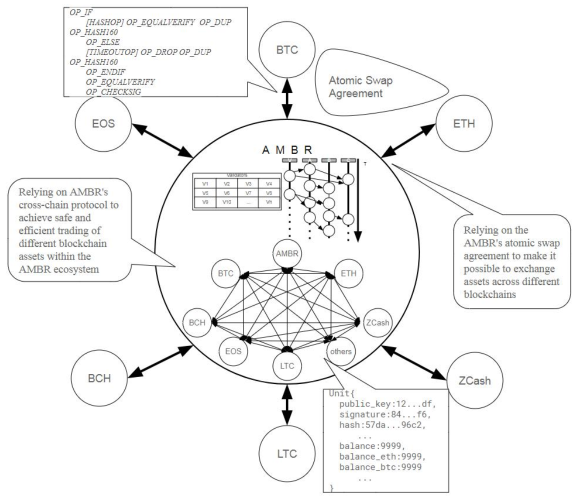
1) Assume Ambr account B needs to pay 10 ETH to Ethereum account X. Account B needs to send a transaction request to the Ambr relay chain. The transaction request should contain the address and amount of ETH account X and be signed by account B.

2) The relay link receives a request from Ambr's account B and records it.

3) A relay unit in the relay chain records the audit of the transaction request, and is digitally verified and signed by the verifier.

4) If the verification is passed, then the end node of account B adds a transaction record deducted by 10 ETH, and the relay chain converts the address of account X in the mapping of ETH and releases any 10 ETH locked in ETH.

5) The 10 ETH is ultimately sent by the contract to account X in the ETH.



```
OP_IF
    [HASHOP] OP_EQUALVERIFY  OP_DUP
OP_HASH160
    OP_ELSE
    [TIMEOUTOP] OP_DROP OP_DUP
OP_HASH160
    OP_ENDIF
    OP_EQUALVERIFY
    OP_CHECKSIG
```

```
Unit{
  public_key:12...df,
  signature:84...f6,
  hash:57da...96c2,
      ...
  balance:9999,
  balance_eth:9999,
  balance_btc:9999
      ...
}
```

BTC
Atomic Swap Agreement
EOS
ETH
A M B R
Validators
V1 V2 V3 V4
V5 V6 V7 V8
V9 V10 ... Vn
Relying on AMBR's cross-chain protocol to achieve safe and efficient trading of different blockchain assets within the AMBR ecosystem
Relying on the AMBR's atomic swap agreement to make it possible to exchange assets across different blockchains
AMBR
BTC
ETH
BCH
ZCash
EOS
LTC
others
BCH
ZCash
LTC

[FIGURE 25: Cross-chain technology overview map]

Ambr's cross-chain trading technology is not limited to a single currency. Ambr's relay chain can be connected with BTC, ZCash, EOS and other derivative spaces. In other words, the block space derived from most currencies in the market can now be connected to Ambr's relay chain. And any newly born currency can be quickly and easily added to the Ambr relay chain simply by signing a contract with Ambr and reaching a consensus. This system will enable Ambr to achieve a very large scale of unlimited expansion, to meet the needs of the global transaction.

Moreover, most currencies on the market today can even be traded directly via the Ambr relay chain. For example, BTC and ETH can trade directly on the Ambr relay chain, eliminating the need to directly establish channels or platforms in BTC and ETH, and saving significant technical and labor costs. Ambr relay chain can be used as a cross-chain platform in the market to achieve transactions between different currencies.

To ensure the speed and security of cross-chain transactions on the Ambr relay chain, Ambr atomic swap implementations will be adopted. Ambr Atomic Swap is a new technology that allows Ambr to achieve trustless peer-to-peer transactions between other types of digital assets. This transaction can be completed in an instant and neither party has the opportunity to violate the agreement. And when one of the parties withdraws from the middle of the transaction, the digital assets will be returned to both parties at a specified time.

This technology has great significance for the future of encrypted digital currency, because this seamless cross-blockchain encryption digital currency exchange capability opens up many new applications. Ambr atomic swaps can open up trade barriers between various encrypted digital currencies to ensure that transactions are correct. If users want to trade between Ambr coins and other types of encrypted digital currency, the technology allows users to fully control these funds.

How Ambr Atomic Swap Works

Let M and N be the parties to the digital asset transaction, M has its own account at Ambr, and N has its own account at Ethereum and Ambr. Say M and N have negotiated a deal by telephone, network, etc., and also know that both parties are in the right Ambr accounts. According to the deal reached, M prepares to transfer 100 Ambr on Ambr to N under Ambr's account, and N will pay 200 ETC to M through Ambr.

In order to complete the transaction of Ambr coins and ETCs, the parties in turn perform the following steps:

1. In order to complete the transaction atomicity, first N customizes a command X and computes V:=Hash(X). X now knows only N himself.

2. N posted a transfer transaction on Ambr and conditionally transferred 200 ETC to M. However, unlike ordinary transfer transactions, the transaction is accompanied by a hash lock condition: M can only present an instruction X' satisfying Hash(X') == V to Ambr within 4,000 seconds to enter the 100 ETC into his account (No essential difference between the

model or UTXO model). If M fails to receive ETC on time, N can return 100 ETC to his account by initiating a refund transaction on Ambr. Both the V and the timeout in the hash lock condition are public, and M can certainly see it.

3. M now sees N on Ambr launching such a transaction, but he does not know what the unlock command X is, so he must pay 100 Ambr to N through Ambr to buy this command X. M then sends an instruction transfer to N with the same hash lock on Ambr for a period of 2000 seconds. If the timeout is not received, the transfer amount will be automatically refunded. This hash-locked instruction transfer is, in principle, easy to implement. The logic is that when N clicks on the instruction, a dialog pops up asking N to enter an instruction X' that satisfies Hash(X') == V. The Ambr coins in the instruction transfer will be transferred to the account of N on Ambr. At the same time, Ambr will send a reply to M to inform M that the transfer has been received, and the N input command X is also displayed on the reply. If N enters the wrong instruction X’, N cannot receive the transfer amount.

4. Now N receives the order transfer, clicks on the order transfer promptly, and enters the command X that he knows. Because V == Hash(X), N successfully got 100 Ambr. According to the program logic, Ambr sends a reply to inform M that the transfer has been charged, and that the command entered by N is X, so that M knows instruction X.

5. Because M now knows instruction X, he can now use instruction X on Ambr to retrieve the 200 outstanding ETCs. M took the time to get 200 ETC on Ambr.

6. At this point, M got 200 ETC and N got 200 Ambr. Ethereum and Ambr did not need to communicate with each other during the transaction, but the atomic exchange between ETC and Ambr is ensured.

The above is the normal process, and the atomicity of the exchange is still established under the abnormal flow. For example, in step 3 above, M did not issue a transfer via Ambr. Since N does not see the transfer, instruction X is not sent to M. M cannot get instruction X and cannot take ETC on Ambr. The atomicity of the transaction is guaranteed. Hash locking mechanisms are easy to implement on blockchains such as Ethereum. It is not difficult for Ambr to implement the above hashed lock transfer.

In addition, since both transfers are designated counterparty transfers, the program can be designed to be unlocked by third party instructions, but the assets are still in the same way as previously specified. This design allows users to suspend wallets temporarily. So unlocking instructions will not compromise security, but have additional benefits.

IX. Security

Through the above-mentioned consensus mechanism, a legitimate transaction must have the following characteristics:

1) The transaction issued by the account cannot be a confirmed transaction. Can be determined by the hash value to determine if this transaction exists.

2) Each transaction must have a valid account signature.

3) The transactions in the account must be chained, i.e. each transaction must have prior transaction information.

4) Since Ambr is an asynchronous model, the Receive module can wait for the signer's signature to take effect. However, the Sender module takes effect immediately. That is, after the Sender module broadcasts, the sender's balance will be immediately deducted from the corresponding part of the Send module. As the number of currency types increases, so does the type of attack. Therefore, possible attack methods will be explored from a security perspective. Below are some of the possible attack methods and how Ambr will respond to these types of attacks.

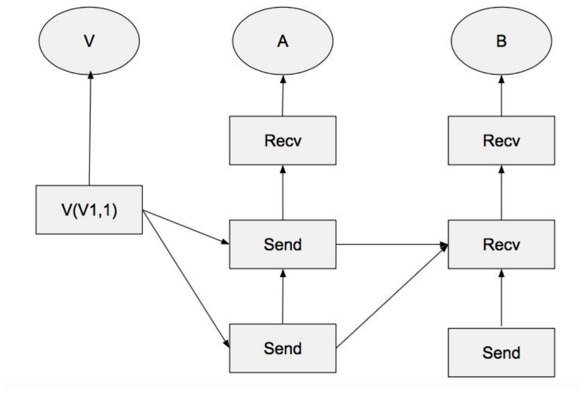
a. Double spend attack

Typical examples of double spend attacks are as follows:

1) Assume that Alice (Attacker) is the buyer, Bob is the seller, Alice buys a digital item from Bob through Bitcoin, and Bob confirms sending the item to Alice.

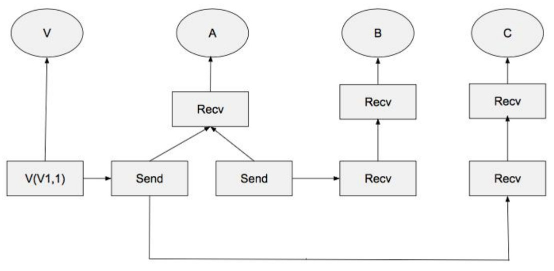
2) At this point Alice purchased a digital item from Charlie. Alice used the same payment method and Charlie confirmed the item. Assuming that Alice only had 10 yuan in the above case, the price of the commodity was also 10 yuan, and Alice bought two items with 10 yuan, which caused a double spending problem.

In the Bitcoin system, after Bob confirms receiving the payment, Bob needs to wait for several generated Blocks before sending items to Alice. This mechanism solves the security issues, but creates delays. Ambr's design pattern perfectly avoids this problem. Each block of Ambr's account chain shows the account balance. Assume that Alice wants to achieve double spending, and its module chain must be one of the following:



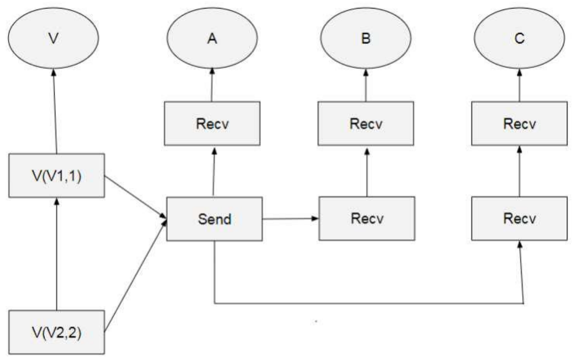
[FIGURE 26: Double spending scenario 1]

As shown in the figure, two consecutive Send modules are under A. Both Send modules are valid in this case. Since the Send module is effective immediately, the account balance will change immediately. Assume that Alice's balance after payment is not enough to pay for the second Send module. The system will not generate the second Send.



[FIGURE 27: Double spending scenario 2]

As shown in the figure, a fork occurs in the A trading chain, and the two Send modules point to one Receive module in the A chain at the same time. Ambr will use the verifier of the account to determine the effectiveness of the two branches. The method is for the verifier to broadcast two branches to the network and vote from the responses received to determine which branch is valid. Regardless of the voting result, the branch will always retain one and give up other branches. So the double spend attack will fail in the Ambr system.



[FIGURE 28: Double spending scenario 3]

3) Another double spending attack is shown in the figure. Assume that during the V(V1,1) verification transaction, M has verified the A-->B transaction and this transaction is valid. In the V(V2,2) verification, V(V2,2) verifies the A-->C transaction and marks it as valid. This is double spending. From A, B, and C, it is difficult to see which transaction is valid. At this time, it can be determined from the chain of verifiers whether or not V(V2,2) is cheating. The method is that in the process of the verifier voting, since V(V1,1) has verified the transaction, the V(V2,2) transaction will be opposed by the set of verifiers. After V(V3,3) collects the votes, it can be determined that the verified transaction of V(V2,2) is not a valid transaction, and V(V2,2) is proved. The system penalizes V(V2,2), confiscates its deposit, kicks it out of the verifier set, and is executed by on V(V2,2) after reverification.

b. Sybil attack

The Sybil attack was originally published by Douceur. That is, in a peer-to-peer network, a single node has multiple identities and the account generates a large number of nodes to influence the outcome of the voting. However, Ambr's equity allocation is allocated by the assets owned by the account, so the Sybil attack has no effect on the Ambr system.

c. Block Gap Synchronization

In Ambr's design mode, the Send and Receive modules are transmitted by UDP in order to keep the amount of data transmitted over the network to a minimum.

However, UDP transmission does not guarantee reliability, and the module does not guarantee secure reception. So the Ambr system node has the following two options:

1) Discard this module as spam sent by the attacking node.

2) The sender is required to retransmit, using secure and reliable TCP transmissions.

At this point, the attacker sends out a large number of junk modules, which will increase the number of TCP transmissions and increase the network load, which will affect the real-time and effectiveness of the Ambr system. Ambr responds as follows:

1) The spam module receives broadcasts from the verifier and counts the responses.

2) If the vote falls below the threshold, the system will discard the junk module.

d. Dust attack

A dust attack means that the attacker sends a large amount of unwanted but valid junk transactions between accounts. If the transaction amount is 0, etc., a large number of transactions will cause the network to become congested and thus affect Ambr. In Ambr design mode, the fee is designed to deal with this attack. When an attacker issues a transaction, he needs to pay a certain fee to issue the transaction. The result of this is that the attacker's attack cost is greatly increased so that the revenue is far less than the cost, greatly discouraging this type of attack.

e. Penny-spend attack

Penny-spend attack is when an attacker sends a large number of transactions to the same node, to use up that node's storage resources, and cannot handle other transactions (DDOS). This kind of attack is similar to the dust attack described above, with different attacks on the same node. However, similar to the dust attack, the attacker's cost is great. And from Ambr's design patterns, users do not need to care about the history of transactions, because the balance will change with the transaction. Therefore, the node can choose to delete some historical records to free up resources, making that the attacking cost too high to be worth the results.

f. > 50% attack

From Ambr's design pattern, votes are based on equity. Therefore, if the attacker owns more than 50% of the equity, he can freely manipulate the voting results. However, the attacker must have 50% of the following restrictions:

1) The Ambr system is based on equity voting, so those with more equity will not want the Ambr system to crash. If the Ambr system crashes, these people need to suffer large losses.

2) If an attacker wants to acquire shares, he or she must acquire more 50% of the shares in the Ambr system, and an increase in acquisition costs (barrier to entry) increases the risk of attacks.

3) Ambr's verifiers are diverse, which greatly reduces the probability of joint verifier attacks.

4) Ambr's design model will punish attackers and once confiscated attacker assets will be confiscated. Therefore, the attacker's attacking cost and risk will be greatly increased, thereby reducing the attacker's motive.

g. Byzantine Generals Problem

From the above consensus that the set of verifiers can form a distributed system. Assuming that the delay of the system network is L, the following conclusions can be drawn from the previous BFT study:

1) In a partially synchronized system (ie, L<d, d is a constant, but the boundary value of d is not known), if a node is less than 1/3 evil, or does not respond, the system can reach a consensus. For example, the PBFT algorithm can reach a consensus when less than 1/3 of the nodes are not responding.

2) The asynchronous model (that is, L is infinite) has no consensus method.

Assume that an attacker uses a network attack (DDoS) to make at least 1/3 of nodes unable to respond, so that a distributed system composed of verifiers cannot reach a consensus and the transaction can never be verified.

Based on the above assumption, Ambr sets a boundary value (1 hour or 1 day). The system will determine that the distributed system composed of the set of verifiers cannot reach consensus. Ambr's response strategy is as follows:

1) Ambr will reassemble the verifier collection, or replace some unresponsive nodes, or combine a new one.

2) All nodes in the system will receive instructions to inform the verifier that the set cannot reach consensus. At this time, the system sets a delay time, assume that this time is T.

3) The node starts the timer after receiving a new verification module. When the time t > T is received, the node can determine that the verification module is valid.

4) The system investigates the original set of verifiers. If the survey results indicate no cyber-attack, the verifier is not responding to its own cause, and the verifier will be punished accordingly. If the investigation results in an attacker launching a cyber-attack, all of the attacker's deposits will be forfeited.

h. Castastrophic Crashes

Since the weight in Ambr is determined by the assets owned by the voters. Assume that the percentage of votes required for a stable point pass is k %, that is, the proportion of the assets owned by the vote in the total assets is greater than k % to form a stable point. In this way, another attack method can be derived. As long as the attacker attacks nodes with more assets, these nodes cannot respond, and the assets owned by these nodes are greater than 1-k %. After the attack is successful, the vote will never pass, i.e., the stability point will not be generated again. At this time, the system will detect the reason why the vote cannot be passed. If the network cannot reach consensus, follow the above steps. If the vote fails, the response strategy is as follows:

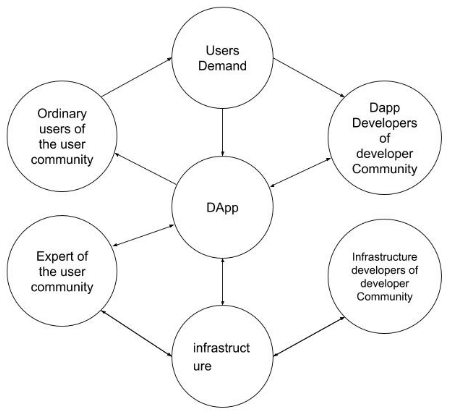
1) The system first detects unresponsive nodes in a set, and the nodes in the set are sorted according to the proportion of owned assets and total assets.

2) The system will set a delay time T. If the current time t > T, the system relocates the assets from the nodes (according to the order of the nodes in the collection) to the voting asset until the required asset.

3) The system checks the nodes in the collection. If the investigation result suggests no network attack, the verifier is not responding to its own reason, and the verifier will be subject to a penalty of taking out the corresponding percentage of the assets. If the investigation results in an attacker launching a cyber-attack, all of the attacker's deposits will be confiscated.

X. Ambr Community

Ambr will invest 30% of the tokens into the community, and the emphasis on the community will surpass any previous blockchain projects. The community will be where the most outstanding people in the entire blockchain industry are concentrated. Major development tasks, decisions, and directions will all depend on the community. So building an active, excellent, united, and interesting community will be the top priority. The financing used are all open and transparent, the evidence will be stored on the chain and published openly.


Users Demand
Ordinary users of the user community
Dapp Developers of developer Community
DApp
Expert of the user community
Infrastructure developers of developer Community
infrastructure

[FIGURE 29: Community map]

A. Developer community

The developer community includes developers of Ambr chain and ones related to DApps. Ambr will periodically release some questions to the community for help. Members who help solve the problem will be rewarded with tokens based on contributions. Any developer who can assist Ambr to maintain its leading technology is welcome to join.

In the future, Ambr will update frequently to integrate the results of the latest R&D into the system to ensure a technological advantage. Also, regardless of what is adopted, all contributing members will be compensated.

Developers are welcome to develop DApps based on Ambr, and will be supported with technical services. Tokens can be used to buy shares. If DApps are profitable, all profits will be attributed to Ambr's community funds, to keep the community active and healthy.

B. User community

In Ambr, users are just as important as the developers, and anyone is welcome to help the development process. Tasks such as analyzing and advising from all fields, including (but not limited to): philosophy, economics, law, sociology, political science, history, journalism, and mathematics, will help greatly. Contributions to Ambr will be rewarded accordingly.

Ambr hopes to be understood by more people, so those who can professionally promote Ambr can also earn tokens, depending on the circumstances (such as in academic journals or blogs).

Reference

[1] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,” Commun. ACM, vol. 21, no. 7, pp. 558–565, 1978.

[2] M Pease, R Shostak, L Lamport. Reaching Agreement in the Presence of Faults. Journal of the ACM, 1980, 27(2): 228-234.

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of Distributed Consensus with One Faulty Process,” J. ACM, vol.32, no. 2, pp. 374–382, 1985.

[4] L. Lamport, “The Part-Time Parliament,” ACM Trans. Comput. Sys-tems, vol. 16, no. 2, pp. 133–169, 1998.

[5] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” Proc. Symp. Oper. Syst. Des. Implement., no. February, pp. 1–14, 1999.

[6] Satoshi Nakamoto,Bitcoin: A Peer-to-Peer Electronic Cash System,https://bitcoin.org/bitcoin.pdf, 2008. A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, “Enabling Blockchain Innovations with Pegged Sidechains,” pp. 1–25, 2014.

[7] T. D. Joseph Poon, “The Bitcoin Lightning Network: Scalable Off-Chain Payments , http://lightning. network/lightning-network-paper.pdf,” pp. 1–59, 2016.

[8] Gentry C., Halevi S.,"Implementing Gentry’s Fully-Homomorphic Encryption Scheme". In: Paterson K.G. (eds) Advances in Cryptology – EUROCRYPT 2011. EUROCRYPT 2011. Lecture Notes in Computer Science, vol 6632. Springer, Berlin, Heidelberg.

[9] van Dijk M., Gentry C., Halevi S., Vaikuntanathan V., "Fully Homomorphic Encryption over the Integers". In: Gilbert H. (eds) Advances in Cryptology – EUROCRYPT 2010. EUROCRYPT 2010. Lecture Notes in Computer Science, vol 6110. Springer, Berlin, Heidelberg.

[10] López-Alt, Adriana, Eran Tromer, and Vinod Vaikuntanathan."On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption.". Proceeding STOC '12 Proceedings of the forty-fourth annual ACM symposium on Theory of computing, Pages 1219-1234.

[11] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous distributed e-cash from bitcoin,” Proc. - IEEE Symp. Secur. Priv., pp. 397–411, 2013.

[12] F. Reid and M. Harrigan, “An analysis of anonymity in the bitcoin system,” Secur. Priv. Soc. Networks, pp. 197–223, 2013.

[13] K. Bhargavan,A. Delignat-Lavaud,C. Fournet,A. Gollamudi, G. Gonthier, N. Kobeissi, A. Rastogi, T. Sibut-Pinote, N.Swamy, and S. Zanella-Béguelin, “Formal Verification of Smart Contracts,” 2016.