Aurei

Aurei White Paper

Aurei

Contents

Aurei（ARI）：
The next generation smart contract and decentralized

application platform

Table of contents

Miscellaneous and Concerns

- Implementation of the Improved Ghost Protocol
- cost
- Computation and Turing completeness
- Currency and issuance
- Release Breakdown
- Centralization of Mining
- Scalability

Overview: Decentralized Applications

in conclusion

Notes and Further Reading

- annotation
- Further Reading

Aurei

Aurei（ARI）: The next generation smart contract and decentralized application platform

When Satoshi Nakamoto launched the Bitcoin blockchain in January 2009, he simultaneously introduced two revolutionary new and untested concepts to the world. The first was bitcoin, a decentralized peer-to-peer online currency that maintains value without any asset backing, intrinsic value, or central issuer. So far, bitcoin has attracted a great deal of public attention, both politically as a currency without a central bank and with wild price swings. However, there was another part of Satoshi's grand experiment that was just as important as bitcoin: the proof-of-work blockchain concept that allowed people to reach consensus on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if someone has 50 BTC and sends it to both A and B at the same time, only the transaction that is confirmed first will take effect. There was no inherent way to determine which of the two transactions came first, and this problem hampered the development of decentralized digital currencies for many years. Satoshi's blockchain was the first reliable decentralized solution. Now, developers' attention is rapidly turning to the second part of bitcoin technology: how blockchain can be used for areas other than currency.

Commonly mentioned applications include using on-chain digital assets to represent custom currencies and financial instruments (colored coins), ownership of some underlying physical device (smart assets), non-fungible assets like domain names (namecoins), and more advanced applications such as decentralized exchanges, financial derivatives, peer-to-peer gambling, and on-chain identity and reputation systems. Another important

area that is often asked about is "smart contracts" - systems that automatically transfer digital assets according to arbitrary rules set in advance. For example, one might have a storage contract of the form "A can withdraw up to X coins per day, B can withdraw up to Y coins per day, A and B can withdraw at will, A can stop B's withdrawal rights". The logical extension of this type of contract is decentralized autonomous organizations (DAOs) - long-term smart contracts that contain an organization's assets and encode the organization's rules. Aurei's goal is to provide a blockchain with a built-in mature Turing-complete language that can be used to create contracts to encode arbitrary state transition functions, allowing users to create all of the above mentioned systems and many others that we have not yet imagined by simply implementing the logic in a few lines of code.

History of Aorecoin

Italy is an ancient European civilization with a long history, outstanding art and unique customs. It is also a developed capitalist country, one of the four largest economies in Europe and the eighth largest economy in the world. Due to its developed small and medium-sized enterprises , it is also known as the "Kingdom of Small and Medium-sized Enterprises". It has a total of 55 UNESCO World Heritage Sites , such as the Arena, Pantheon, Arc de Triomphe and many other ancient buildings. It is ranked with China as the country with the most World Heritage Sites in the world. And these heritages can be traced back to historical periods.

Rome is the capital of Italy, and it can also be said to be the predecessor of Italy. Why do we say that? This is because ancient Rome refers to the civilization that emerged in the central part of the Italian Peninsula (i.e. the Apennine Peninsula) in the early 9th century BC. Ancient Rome went through three stages: the Roman Kingdom (753-509 BC), the Roman Republic (509-27 BC), and the Roman Empire (27-476 BC/1453). During the Roman Republic and the Roman Empire, the capital was located in Rome, which lasted from 509 BC to 1453 AD. During this period, it was divided into the Eastern Roman Empire and the Western Roman Empire in 395 AD. The Western Roman Empire continued to set its capital in Rome, while the Eastern Roman Empire had little to do with Italy. The Western Roman Empire fell in 476 AD, and the Middle Ages in Europe began. It lasted until the fall of the Eastern Roman Empire in 1453, and the Middle Ages ended. Therefore,

there was no country called Rome on Italian soil during the Middle Ages. This period was a long period of division and separatism in Italy, and it was finally unified by the Kingdom of Saturn in the 19th century. From the founding of the Roman Republic to the unification of Italy, the Italian nation has played an extremely important and indispensable position, so it can be said that Rome is the predecessor of Italy.

As a tool for measuring prices and a medium for purchasing goods, currency has also been given different storage values in different periods. . The currency of ancient Rome can be traced back to the end of the 7th century BC, when some Greek cities in Asia Minor pressed gold and silver alloys. The main ones are cadras, sestels, semis, aures, duponti, dinars, and ass. . Gold coins have been issued since the Second Punic War of ancient Rome. As a gold coin: aure, (Aureus), the most valuable currency in Roman currency, was cast in gold. In the era of Caesar, an aure weighed about 8 grams. In 24 BC, Octavian established the Roman coinage system, which was based on 4 metals: gold, silver, brass and bronze. Designed for different categories of use. Gold and silver were used officially. It was also during the Octavian period that the aure was set at 1/40 Roman pound. It has been quite scarce since the mid-2nd century AD, but it was still the standard gold coin in the early and middle stages of the empire. Therefore, the value is also high. Italy launched the Aurel Coin, which is derived from blockchain technology and will be issued globally. The issuance of Aurel Coin has also been recognized and agreed upon by many countries. The reason why it has attracted so much attention and influence is that it is a digital cryptocurrency issued using blockchain 3.0 technology. The 3.0 technology is based on blockchain technology 1.0 and 2.0, and has made new innovations and improvements and fully realized decentralized issuance.

The concept of decentralized digital currency, like alternative applications such as property registries, has been proposed for decades. Most of the anonymous electronic cash protocols of the 1980s and 1990s were based on Chaumian blinding. These electronic cash protocols provided highly private currencies, but none of them became popular because they all relied on a centralized intermediary. In 1998, Wei Dai's b-money first introduced the idea of creating money by solving computational problems and decentralized consensus, but the proposal did not provide a specific method for how to achieve decentralized consensus. In 2005, Hal Finney introduced the concept of "reusable proofs of work", which used both the ideas of b-money and the computationally difficult Hashcash problem proposed by Adam Back to create cryptocurrencies. However, this concept once again lost itself in idealism because it relied on trusted computing as a backend.

Because money is a first-come-first-served application, the order of transactions is crucial, so decentralized money needs to find a way to achieve decentralized consensus. The main obstacle encountered by all electronic money protocols before Bitcoin is that although research on how to create secure Byzantine-fault-tolerant multi-party consensus systems has lasted for many years, the above protocols only solve half of the problem. These protocols assume that all participants in the system are known and produce security boundaries in the form of "if there are N parties participating in the system, then the system can tolerate N/4 malicious participants." However, the problem with this assumption is that in the case of anonymity, the security boundaries set by the system are vulnerable to Sybil attacks, because an attacker can create thousands of nodes on a server or botnet to unilaterally ensure that they have a majority share.

Satoshi Nakamoto's innovation was to introduce the idea of combining a very simple node-based decentralized consensus protocol with a proof-of-work mechanism. Nodes gain the right to participate in the system through the proof-of-work mechanism, packaging transactions into "blocks" every ten minutes, thereby creating an ever-growing blockchain. Nodes with a lot of computing power have greater influence, but it is much more difficult to gain more computing power than the entire network than to create a million nodes. Although the Bitcoin blockchain model is very simple, it has proven to be useful enough in practice. In the next five years, it will become the cornerstone of more than two hundred currencies and protocols around the world.

Bitcoin as a state transition system

Technically speaking, the Bitcoin ledger can be thought of as a state transition system that includes all existing Bitcoin ownership states and a "state transition function". The state transition function takes the current state and a transaction as input and outputs a new state. For example, in a standard banking system, the state is a balance sheet. A request to transfer X dollars from account A to account B is a transaction. The state transition function will subtract X dollars from account A and add X dollars to account B. If the balance of account A is less than X dollars, the state transition function will return an error message. So we can define the state transition function as follows:

APPLY(S,TX) > S' or ERROR

In the banking system mentioned above, the state transition function is as follows:

APPLY({ Alice: $50, Bob: $50 },"send $20 from Alice to Bob") = { Alice: $30,Bob: $70 }

but:

APPLY({ Alice: $50, Bob: $50 },"send $70 from Alice to Bob") = ERROR

The “state” of the Bitcoin system is the set of all mined, unspent bitcoins (technically called “unspent transaction outputs” or UTXOs). Each UTXO has a value and an owner (defined by a 20-byte address which is essentially a cryptographic public key[1]). A transaction consists of one or more inputs and one or more outputs. Each input contains a reference to an existing UTXO and a cryptographic signature created by the private key corresponding to the owner’s address. Each output contains a new UTXO that is added to the state.

In the Bitcoin system, the state transition function APPLY(S,TX)->S' can be roughly defined as follows:

1.Each input of the transaction:

- oIf the referenced UTXO does not exist in the current state (S), an error message is returned
- oIf the signature does not match the UTXO owner's signature, an error message is returned

2.If the total value of all UTXO inputs is less than the total value of all UTXO outputs, an error message is returned.

3.Return to the new state S' , in which all input UTXOs are removed and all output UTXOs are added.

The first part of the first step prevents the sender of the transaction from spending non-existent bitcoins, and the second part prevents the sender of the transaction from spending other people's bitcoins. The second step ensures value conservation. Bitcoin's payment protocol is as follows. Suppose Alice wants to send 11.7BTC to Bob. In fact, Alice cannot have exactly 11.7BTC. Suppose, the minimum amount of bitcoins she can get is: 6+4+2=12. Therefore, she can create a transaction with 3 inputs and 2 outputs. The face value of the first output is 11.7BTC, and the owner is Bob (Bob's Bitcoin address), and the face value of the second output is 0.3BTC, and the owner is Alice herself, that is, change.

ARI output mechanism

If we had a trusted centralized service, the state transition system could be easily implemented, and the above functions could simply be encoded accurately. However, we want to build the Bitcoin system into a decentralized currency system, and in order to ensure that everyone agrees on the order of transactions, we need to combine the state transition system with a consensus system. Bitcoin's decentralized consensus process requires nodes in the network to continuously try to package transactions into "blocks". The network is designed to produce a block approximately every ten minutes, and each block contains a timestamp, a random number, a

reference to the previous block (i.e., a hash), and a list of all transactions that have occurred since the previous block was generated. In this way, a continuously growing blockchain is created over time, which is constantly updated to represent the latest state of the Bitcoin ledger.

According to this paradigm, the algorithm for checking whether a block is valid is as follows:

1.Checks whether the previous block referenced by the block exists and is valid.

2.Check if the block's timestamp is later than the previous block's timestamp and earlier than 2 hours in the future [2].

3.Checks whether the block's proof of work is valid.

4.Assign the final state of the previous block to S[0] .

5.Assume that TX is the transaction list of the block, which contains n transactions. For all i belonging to 0...n-1, perform state transition S[i+1] = APPLY(S[i],TX[i]) . If any transaction i fails in the state transition, exit the program and return an error.

6.Returns correct, state S[n] is the final state of this block.

Essentially, every transaction in a block must provide a correct state transition. Note that the "state" is not encoded in the block. It is purely an abstract concept remembered by the validation nodes. For any block, you can start from the genesis state and add every transaction in every block in order to calculate the current state. In addition, you need to pay attention to the order in which miners include transactions in the block. If there are two transactions A and B in a block, and B spends the UTXO created by A, if A is before B, the block is valid, otherwise, the block is invalid.

The interesting part of the block validation algorithm is the concept of “proof of work”: Each block is hashed with SHA256 and the resulting hash is treated as a 256-bit value that must be less than a dynamically adjusted target value, which at the time of this writing is approximately 2^190. The purpose of proof of work is to make block creation difficult, thereby preventing Sybil attackers from maliciously regenerating the blockchain. Because SHA256 is a completely unpredictable pseudo-random function, the only way to create a valid block is to simply try and error, increasing the value of the random number and seeing if the new hash value is less than the target value. If the current target value is 2^192, this means that on average it takes 2^64 attempts to generate a valid block. Generally speaking, the Bitcoin network resets the target value every 2016 blocks, which guarantees that a block is generated every ten minutes on average. To reward miners for their computational work, each miner who successfully generates a block is entitled to include in the block a transaction that sends them 25 BTC out of thin air. In addition, if the input of the transaction is greater than the output, the difference is paid to the miner as a “transaction fee”. By the way, rewards to miners are the only mechanism for issuing Bitcoins, and there are no Bitcoins in the genesis state.

To better understand the purpose of mining, let's analyze what happens when a malicious attacker appears on the Bitcoin network. Because Bitcoin's cryptographic foundation is very secure, the attacker will choose to attack the part that is not directly protected by cryptography: the transaction sequence. The attacker's strategy is very simple:

1.Send 100 BTC to the seller to purchase the product (especially electronic products that do not require mailing).

2.Wait until the item is shipped.

3.Create another transaction to send the same 100 BTC to your own account.

4.Make the Bitcoin network believe that the transaction sent to your account was sent first.

5.Aurei (ARI) has completed the above-mentioned production process, and 100 million pieces have undergone currency code audit and total quantity audit, and can be paid and transferred normally.

Merkle Tree

Figure 1: Providing only a few nodes on the Merkle tree is enough to provide legal proof of a branch.

Figure 2: Any attempt to change any part of the Merkle tree will eventually lead to an inconsistency somewhere on the chain.

An important scalability feature of the Bitcoin system is that its transactions are stored in a multi-level data structure. The hash of a block is actually just the hash of the block header, which is a piece of data about 200 bytes long that contains a timestamp, a random number, the hash of the previous block, and the root hash of the Merkle tree that stores all the block transactions.

A Merkle tree is a binary tree consisting of a set of leaf nodes, a set of intermediate nodes, and a root node. The bottom, a large number of leaf nodes, contain the basic data, each intermediate node is the hash of its two children, and the root node is also the hash of its two children, representing the top of the Merkle tree. The purpose of a Merkle tree is to allow block data to be transmitted piecemeal: a node can download a block header from one source, and the rest of the tree from another source, and still be able to confirm that all the data is correct. This is possible because the hash propagates upward: if a malicious user tries to add a fake transaction lower in the tree, the resulting change will cause changes to nodes higher up in the tree, and to nodes higher up, and eventually to the root node and the block hash, so that the protocol will record it as a completely different block (almost certainly with an incorrect proof of work).

The Merkle tree protocol is critical to the long-term sustainability of Bitcoin. In April 2014, a full node in the Bitcoin network - a node that stores and processes all the data for all blocks - took up 15GB of storage space, and it is growing at a rate of more than 1GB per month. Currently, this storage space is acceptable for desktop computers, but mobile phones are no longer able to handle such a large amount of data. In the future, only commercial organizations and hobbyists will act as full nodes. The simplified payment confirmation (SPV) protocol allows for another type of node to exist, such a node is called a "light node", which downloads the block header, confirms the workload proof with the block header, and then downloads only the Merkle tree "branches" related to its transactions. This allows light nodes to safely determine the status of any Bitcoin transaction and the current balance of an account by downloading only a small part of the entire blockchain.

Other blockchain applications

The idea of applying the idea of blockchain to other areas has long been around. In 2005, Nick Szabo proposed the concept of "title property with ownership", describing how the development of replicated database technology can enable blockchain-based systems to be applied to register land ownership, creating a detailed framework that includes concepts such as property rights, illegal encroachment, and Georgia land taxes. Unfortunately, however, there were no practical replicated database systems at the time, so the protocol was not put into practice. However, since the successful development of the decentralized consensus of the Bitcoin system in 2009, many other applications of blockchain have begun to emerge rapidly.

Namecoin - Created in 2010, it is billed as a decentralized name registry. Decentralized protocols like Tor, Bitcoin, and BitMessage require some way to identify accounts so that others can interact with them. However, the only available identifiers in all existing solutions are pseudo-random hashes like 1LW79wp5ZBqaHW1jL5TciBCrhQYtHagUWy . Ideally, one would like to have an account with a name like "george". However, the problem is that if someone can create an account called "george", then anyone else can also create an account called "george" to pretend to be someone else. The only solution is a first-to-file system, where only the first registrant can successfully register, and a second person cannot register the same account again. This problem can be solved by using Bitcoin's consensus protocol. Namecoin is the earliest and most successful name registry system to use blockchain.

Colored coins - The purpose of colored coins is to provide a service for people to create their own digital currency on the Bitcoin blockchain, or, more importantly, currency in general - digital tokens. According to the colored coins protocol, people can issue new currency by specifying a color for a particular Bitcoin UTXO. The protocol recursively defines other UTXOs to be the same color as the transaction input UTXO. This allows users to keep UTXOs that only contain a certain color, and send these UTXOs just like sending normal Bitcoins, by backtracking the entire blockchain to determine the color of the received UTXO.

Metacoins - The idea of Metacoins is to create a new protocol on top of the Bitcoin blockchain, using Bitcoin transactions to store Metacoin transactions, but with a different state transition function APPLY'. Because the Metacoin protocol cannot prevent invalid Metacoin transactions on the Bitcoin blockchain, a rule is added that if APPLY'(S,TX) returns an error, the protocol will default to APPLY'(S,TX) = S. This provides a simple solution for creating arbitrary, advanced cryptocurrency protocols that cannot be implemented in the Bitcoin system, and the development cost is very low, because the mining and network issues are already handled by the Bitcoin protocol.

Therefore, in general, there are two ways to build a consensus protocol: building a separate network and building a protocol on top of the Bitcoin network. While applications such as Namecoin have been successful using the first approach, it is difficult to implement because each application requires the creation of a separate blockchain and the building and testing of all state transitions and network code. In addition, we predict that the

adoption of decentralized consensus technology will follow a power-law distribution, with most applications being too small to secure a free blockchain, and we also note that a large number of decentralized applications, especially decentralized autonomous organizations, require interaction between applications.

On the other hand, the Bitcoin-based approach has the disadvantage that it does not inherit Bitcoin's property of being able to perform Simplified Payment Verification (SPV). Bitcoin can achieve SPV because it can use blockchain depth as a proxy for validity. At a certain point, once a transaction's ancestors are far enough away from the present, they can be considered part of the legal state. In contrast, metacoin protocols based on the Bitcoin blockchain cannot force the blockchain to not include transactions that do not conform to the metacoin protocol. Therefore, SPV for secure metacoin protocols requires scanning all blocks backwards to the beginning of the blockchain to confirm whether a transaction is valid. Currently, all "light" implementations of metacoin protocols based on Bitcoin rely on trusted servers to provide data, which is a rather suboptimal result for a cryptocurrency where one of the main purposes is to eliminate the need for trust.

script

Even without extending the Bitcoin protocol, it can implement "smart contracts" to a certain extent. Bitcoin's UTXO can be owned not only by a public key, but also by a more complex script written in a stack-based programming language. In this model, to spend such a UTXO, data that satisfies the script must be provided. In fact, the basic public key ownership

mechanism is also implemented through scripts: the script takes an elliptic curve signature as input, verifies the transaction and the address that owns this UTXO, and returns 1 if the verification is successful, otherwise it returns 0. More complex scripts are used for other different applications. For example, people can create scripts that require two of the three private keys to be collected for transaction confirmation (multi-signature), which is very useful for corporate accounts, savings accounts and certain commercial agents. Scripts can also be used to send rewards to users who solve computational problems. People can even create a script like "If you can provide a simplified confirmation payment proof that you have sent a certain amount of Dogecoin to me, this Bitcoin UTXO is yours." In essence, the Bitcoin system allows decentralized exchange of different cryptocurrencies.

However, the Bitcoin scripting language has some serious limitations:

Lack of Turing completeness – This means that while the Bitcoin script language can support many computations, it cannot support all computations. The main missing piece is loop statements. The purpose of not supporting loop statements is to avoid infinite loops when confirming transactions. In theory, this is an obstacle that can be overcome for script programmers, because any loop can be simulated by repeating if statements many times, but doing so will lead to inefficient use of script space. For example, implementing an alternative elliptic curve signature algorithm may require 256 repeated multiplications, each of which needs to be coded separately.

Value-blindness . UTXO scripts cannot provide fine-grained control over the withdrawal amount of an account. For example, a powerful application of oracle contracts is a hedging contract, where A and B each send $1,000 worth of bitcoin to the hedging contract. After 30 days, the script sends $1,000 worth of bitcoin to A and the remaining bitcoin to B. Although implementing a hedging contract requires an oracle to determine how much a bitcoin is worth in dollars, this mechanism has made great progress in reducing trust and infrastructure compared to the current fully centralized solution. However, because UTXO is indivisible, the only way to implement this contract is to use many UTXOs with different denominations (for example, there is a 2^k UTXO for each k with a maximum value of 30) and let the oracle pick the correct UTXO to send to A and B.

Lack of state - UTXOs can only be spent or unspent, which leaves no room for multi-stage contracts or scripts that require any other internal state. This makes it difficult to implement multi-stage options contracts, decentralized exchange offers, or two-stage cryptographic commitment protocols (necessary to ensure computational rewards). It also means that UTXOs can only be used to build simple, one-off contracts, rather than more complex stateful contracts such as decentralized organizations, making meta-protocols difficult to implement. Binary state combined with value blindness means that another important application - withdrawal limits - is impossible to implement.

Blockchain-blindness - UTXO cannot see the blockchain data, such as random numbers and the hash of the previous block. This defect deprives the scripting language of its potential value based on randomness, severely limiting its application in other fields such as gambling.

We have examined three approaches to building advanced applications on top of cryptocurrencies: building a new blockchain, using scripts on the Bitcoin blockchain, and building a metacoin protocol on the Bitcoin blockchain. The new blockchain approach gives you the freedom to implement any features you want, at the cost of development time and nurturing effort. The script approach is very easy to implement and standardize, but it is limited in its capabilities. The metacoin protocol, while very easy to implement, suffers from poor scalability. In the Aurei system, our goal is to build a general framework that can take advantage of all three approaches.

Aurei

Aurei aims to integrate and improve upon the concepts of scripting, altcoins, and on-chain meta-protocols, enabling developers to create arbitrary consensus-based, scalable, standardized, feature-rich, easy-to-develop, and interoperable applications. Aurei does this by building the ultimate abstract base layer - a blockchain with a Turing-complete programming language built in - enabling anyone to create contracts and decentralized applications with their own custom ownership rules, transaction methods, and state transition functions. The main framework for Namecoin can be implemented in two lines of code, and other protocols such as currencies and reputation systems can be implemented in less than twenty lines of code. Smart contracts - encrypted boxes that contain value and can only be opened when certain conditions are met - can also be created on our platform, and are much more powerful than what Bitcoin Script can provide due to the added power of Turing-completeness, value-awareness, blockchain-awareness, and multi-state.

Aurei Account

In the Aurei system, the state is composed of objects called "accounts" (each account consists of a 20-byte address) and state transitions that transfer value and information between two accounts. Aurei accounts consist of four parts:

- A random number, a counter used to ensure that each transaction can only be processed once

- The current ARI balance of the account
- The account's contract code, if any
- Account storage (empty by default)

ARI coin (Ether) is the main crypto fuel inside Aurei and is used to pay transaction fees. Generally speaking, Aurei has two types of accounts: externally owned accounts (controlled by private keys) and contract accounts (controlled by contract code). Externally owned accounts have no code, and people can send messages from an external account by creating and signing a transaction. Whenever a contract account receives a message, the code inside the contract is activated, allowing it to read and write to internal storage, send other messages or create contracts.

Messages and Transactions

Aurei messages are somewhat similar to Bitcoin transactions, but there are three important differences between the two. First, Aurei messages can be created by external entities or contracts, while Bitcoin transactions can only be created from the outside. Second, Aurei messages can optionally contain data. Third, if the recipient of an Aurei message is a contract account, it can choose to respond, which means that Aurei messages also contain the concept of functions.

A "transaction" in Aurei is a signed data package that stores messages sent from external accounts. A transaction contains the recipient of the message, a signature to confirm the sender, the ARI coin account balance, the data to be sent, and two values called STARTGAS and GASPRICE. In order to prevent

exponential explosion and infinite loops of code, each transaction needs to limit the number of computational steps caused by executing the code - including the initial message and all messages caused by execution. STARTGAS is the limit, and GASPRICE is the fee that needs to be paid to miners for each computational step. If the gas is "used up" during the execution of the transaction, all state changes are restored to the original state, but the transaction fees paid cannot be recovered. If there is still gas left when the execution of the transaction is aborted, then the gas will be refunded to the sender. There are separate transaction types and corresponding message types for creating contracts; the address of the contract is calculated based on the hash of the account random number and the transaction data.

An important consequence of the messaging mechanism is Aurei's "first class citizen" property - contracts have the same rights as external accounts, including the right to send messages and create other contracts. This allows contracts to play multiple different roles at the same time, for example, a user can make a member of a decentralized organization (one contract) an intermediary account (another contract) that acts as an intermediary between a paranoid individual using a custom quantum proof-based Lamport signature (a third contract) and a co-signing entity that itself uses an account secured by five private keys (a fourth contract). The power of the Aurei platform is that decentralized organizations and proxy contracts do not need to care what type of account each party to the contract has.

Aurei state transition function

Aurei's state transition function: APPLY(S,TX) -> S' can be defined as follows:

1.Checks whether the transaction is in the correct format (i.e. has the correct value), whether the signature is valid, and whether the random number matches the random number of the sender's account. If not, returns an error.

2.Calculate the transaction fee: fee=STARTGAS * GASPRICE and determine the sender's address from the signature. Subtract the transaction fee from the sender's account and add the sender's random number. If the account balance is insufficient, return an error.

3.Set the initial value of GAS = STARTGAS and subtract a certain amount of gas value according to the number of bytes in the transaction.

4.Transfer value from the sender's account to the receiver's account. If the receiving account does not exist yet, create it. If the receiving account is a contract, run the contract's code until the code finishes running or the gas runs out.

5.If the value transfer fails because the sender's account does not have enough money or the code execution runs out of gas, the original state will be restored, but the transaction fee must still be paid, and the transaction fee is added to the miner's account.

6.Otherwise, all remaining gas is returned to the sender, and the consumed gas is sent to the miner as a transaction fee. For example, suppose the contract code is as follows:

```
if not self.storage[ calldataload ( 0 )]:
self.storage[ calldataload ( 0 )] = calldataload ( 32 )
```

It is important to note that in reality the contract code is written in the underlying Aurei Virtual Machine (EVM) code. The above contract is written in our high-level language Serpent, which can be compiled into EVM code. Assume that the contract memory is empty at the beginning, a value of 10ARI coins, gas is 2000, gas price is 0.001ARI coins and 64 bytes of data, the first thirty-two bytes of the block represent the number 2 and the second represents the word CHARLIE . After the transaction is sent, the state transition function is processed as follows:

1.Check that the transaction is valid and in the correct format.

2.Check if the sender of the transaction has at least 2000*0.001=2 ARI coins. If so, subtract 2 ARI coins from the sender's account.

3.The initial setting of gas=2000, assuming the transaction length is 170 bytes, the fee per byte is 5, minus 850, so there is 1150 left.

4.Subtract 10 ARI coins from the sender's account and add 10 ARI coins to the contract account.

5.Run the code. In this contract, the code is simple: it checks if the contract storage at index 2 is used, notices that it is not, and sets its value to CHARLIE. Assume that this consumes 187 units of gas, so the remaining gas is 1150 - 187 = 963.

6.Add 963*0.001=0.963 ARI coins to the sender's account and return the final state.

If no contract receives the transaction, then all transaction fees are equal to GASPRICE multiplied by the byte length of the transaction, and the data of the transaction has nothing to do with the transaction fee. In addition, it is important to note that messages initiated by contracts can assign gas limits to the computations they generate, and if a subcomputation runs out of gas, it simply reverts to the state it was in when the message was sent. Therefore, just like transactions, contracts can also protect their computational resources by setting strict limits on the subcomputations it generates.

Code Execution

The code of Aurei contracts is written in a low-level stack-based bytecode language, called "Aurei virtual machine code" or "EVM code". The code consists of a series of bytes, each of which represents an operation. Generally speaking, code execution is an infinite loop, and an operation is executed every time the program counter increases by one (initial value is zero) until the code is executed or an error, STOP or RETURN instruction is encountered. Operations can access three types of storage space:

- A stack is a last-in-first-out data storage where 32-byte values can be pushed into and popped from the stack.
- In-memory , infinitely scalable queue of bytes.

Long-term storage of the contract , a key/value storage, where the key and value are both 32 bytes in size. Unlike the stack and memory that are reset when the calculation is completed, the storage content will be maintained for a long time.

The code can access values like block header data, sender and received message data, and the code can also return a byte array of data as output.

The formal execution model of EVM code is surprisingly simple. When the Aurei virtual machine is running, its complete computational state can be defined by the tuple (block_state, transaction, message, code, memory, stack, pc, gas) , where block_state is the global state containing all account balances and storage. At each execution round, the current instruction is found by calling the pc (program counter)th byte of the code, and each instruction has its own definition of how it affects the tuple. For example, ADD pops two elements from the stack and pushes their sum, decrements gas by one and increments pc by one, and SSTORE pops the top two elements from the stack and inserts the second element into the contract storage location defined by the first element, again reducing the gas cost by up to 200 and incrementing pc by one. Although there are many ways to optimize Aurei through just-in-time compilation, a basic implementation of Aurei can be implemented in a few hundred lines of code.

Blockchain and Aurei’s output mechanism

Although there are some differences, Aurei's blockchain is similar to the Bitcoin blockchain in many ways. The difference in their blockchain architecture is that Aurei blocks contain not only transaction records and recent states, but also block numbers and difficulty values. The block confirmation algorithm in Aurei is as follows:

1.Checks whether the previous block referenced by the block exists and is valid.

2.Checks if the timestamp of the block is greater than the previous block referenced and is less than 15 minutes old.

3.Checks that block numbers, difficulty values, transaction roots, uncle roots, and gas limits (many low-level concepts unique to Aurei) are valid.

4.Checks whether the block's proof of work is valid.

5.Assign S[0] to the STATE_ROOT of the previous block .

6.Assign TX to the transaction list of the block, with a total of n transactions. For i belonging to 0...n-1 , perform state transition S[i+1] = APPLY(S[i],TX[i]) . If any transition fails, or the gas spent by the program to this point exceeds GASLIMIT , an error is returned.

7.Assign S[n] to S_FINAL and pay the block reward to the miner.

8.Check if S_FINAL is the same as STATE_ROOT . If so, the block is valid. Otherwise, the block is invalid.

This confirmation method may seem inefficient at first glance, as it requires storing all the states for each block, but in fact Aurei's confirmation efficiency is comparable to that of Bitcoin. The reason is that the states are stored in a tree structure, and each additional block only requires changing a small part of the tree structure. Therefore, in general, most of the tree structure of two adjacent blocks should be the same, so that the data is stored once and can be referenced twice using a pointer (i.e., a subtree hash). A tree structure called a "Patricia Tree" can achieve this, which includes a modification of the Merkle tree concept that allows not only changing nodes, but also inserting and deleting nodes. In addition, because all state information is part of the last block, there is no need to store the entire block history - this method, if it can be applied to the Bitcoin system, has been calculated to save 10-20 times the storage space.

application

Broadly speaking, there are three types of applications on Aurei. The first are financial applications, which provide users with more powerful ways to manage and participate in contracts with their money. These include sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and even some types of comprehensive employment contracts. The second are semi-financial applications, where money is present but there are also heavy non-monetary aspects, a perfect example being self-enforced bounties for solving computational problems. Finally, there are completely non-financial applications like online voting and decentralized governance.

Token System

On-chain token systems have many applications, from sub-currencies representing assets like dollars or gold to company shares, individual tokens representing smart assets, secure unforgeable coupons, and even token systems for reward points that have no connection to traditional value at all. Implementing a token system in Aurei is surprisingly easy. The key point is to understand that all currencies or token systems are, at their core, a database with operations like: subtract X units from A and add X units to B, provided that (1) A had at least X units before the transaction and (2) the transaction was approved by A. Implementing a token system is a matter of implementing this logic into a contract.

The basic code to implement a token system in Serpent language is as follows:

```
def send (to, value):
    if self.storage[from] >= value:
        self.storage[from] = self.storage[from] - value
        self.storage[to] = self.storage[to] + value
```

This is essentially a minimal implementation of the "banking system" state transition functionality described further down in this article. Some additional code would need to be added to provide functionality for distributing coins initially and in some other edge cases, and ideally a function for other contracts to query the balance of an address. That would be enough. In theory, an Aurei-based token system that acts as a sub-currency could include an important feature that Bitcoin-based on-chain meta-coins lack: the ability to pay transaction fees directly with that currency. This capability would be achieved by maintaining an ARI coin account in the contract that is used to pay transaction fees for senders, and the contract would continually fund the ARI coin account by collecting the internal currency used to pay transaction fees and auctioning it off in a constantly running auction. This would require users to "activate" their accounts with ARI coin, but once the account has ARI coin it can be reused as the contract tops it up each time.

Financial derivatives and stable currencies

Financial derivatives are the most common application of "smart contracts" and one of the easiest to implement with code. The main challenge in implementing financial contracts is that most of them need to refer to an external price publisher; for example, a very popular application is a smart contract used to hedge the fluctuations of the price of ARI coins (or other cryptocurrencies) relative to the US dollar, but the contract needs to know the price of ARI coins relative to the US dollar. The simplest way to do this is through a "data provider" contract maintained by a specific institution (such as Nasdaq), which is designed so that the institution can update the contract as needed and provide an interface so that other contracts can send a message to the contract to get a reply containing price information.

When these key elements are in place, a hedge contract looks like this:

1.Waiting for A to input 1000ARI coins. .

2.Wait for B to input 1000ARI coins.

3.By querying the data provider contract, the dollar value of 1000 ARI coins, for example, x dollars, is recorded to storage.

After 30 days, either A or B is allowed to "reactivate" the contract to send $x worth of ARI coins (requery the data provider contract to get the new price and calculate) to A and send the remaining ARI coins to B.

Such contracts have extraordinary potential for crypto-commerce. One of the common criticisms of cryptocurrencies is their volatility; while a large number of users and merchants may want the security and convenience of crypto assets, they are unlikely to be happy with an asset losing 23% of its value in a single day. Until now, the most commonly recommended solution has been issuer-backed assets; the idea is that the issuer creates a sub-currency that they have the right to issue and redeem, giving one unit of the sub-currency to anyone who (offline) provides them with one unit of a certain underlying asset (e.g. gold, dollars). The issuer promises to return one unit of the underlying asset to anyone who returns one unit of the crypto asset. This mechanism enables any non-crypto asset to be "upgraded" to a crypto asset, if the issuer is trustworthy.

In practice, however, issuers are not always trustworthy, and in some cases the banking system is too fragile or dishonest to allow such services to exist. Financial derivatives offer an alternative. Instead of a single issuer providing reserves to back an asset, there is a decentralized market of speculators betting that the price of a crypto asset will rise. Unlike the issuer, speculators do not have bargaining power on their side, as the hedge contract freezes their reserves in the contract. Note that this approach is not fully decentralized, as a trusted source of price information is still required, although it is arguable that this is still a huge step forward in reducing infrastructure requirements (unlike an issuer, a price publisher does not require a license and would appear to be free speech) and reducing the risk of potential fraud.

Identity and Reputation Systems

The earliest altcoin, Namecoin, attempted to use a Bitcoin-like blockchain to provide a name registry system where users could register their names along with other data in a public database. The most common use case is a domain name system that maps a domain name like "bitcoin.org" (or in Namecoin's case, "bitcoin.bit") to an IP address. Other use cases include email verification systems and potentially more advanced reputation systems. Here is the basic contract in Aurei that provides a name registry system similar to Namecoin:

```
def register (name, value):
    if !self.storage[name]:
self.storage[name] = value
```

The contract is very simple; it's a database in the Aurei network that can be added to but not modified or removed. Anyone can register a name as a value that never changes. A more complex name registry contract would include "functionality clauses" that allow other contracts to query, and a mechanism for the "owner" of a name (i.e. the first registrant) to modify the data or transfer ownership. Even reputation and trust network features could be added on top of it.

Decentralized Storage

Over the past few years, a number of popular online file storage startups have emerged, most notably Dropbox, which seeks to allow users to upload backups of their hard drives, store the backups, and allow users to access them for a monthly fee. However, the file storage market is sometimes

relatively inefficient at this point; a cursory look at existing services shows that, particularly at the "uncanny valley" of 20–200GB where there is neither free space nor enterprise discounts, mainstream file storage costs per month mean paying for the cost of an entire hard drive in a month. The Aurei contract allows for the development of a decentralized storage ecosystem where users can reduce the cost of file storage by renting out their own hard drives or unused network space for a small profit.

The fundamental building block of such a facility is what we call a "decentralized Dropbox contract". This contract works as follows. First, someone breaks the data to be uploaded into chunks, encrypts each chunk for privacy, and builds a Merkle tree out of it. Then a contract is created with the following rules: every N chunks, the contract extracts a random index from the Merkle tree (using the hash of the previous chunk accessible to the contract code to provide randomness), and then gives the first entity XARI coins to back a proof of ownership of the chunk at that particular index in the tree with something like Simplified Payment Verification (SPV). When a user wants to re-download his file, he can use a micropayment channel protocol (e.g. 1 SAB per 32k bytes) to restore the file; the most cost-efficient way to do this is for the payer to not publish the transaction until the end, but to replace the original transaction with a slightly more cost-effective transaction with the same random number after every 32k bytes.

An important feature of this protocol is that, although it looks like a person trusts many random nodes who are not prepared to lose the file, he can divide the file into many small pieces through secret sharing, and then know through monitoring contracts that each small piece is still kept by a certain node. If a contract is still paying, it provides evidence that someone is still keeping the file.

Decentralized Autonomous Organization

The concept of a "decentralized autonomous organization" (DAO) is generally a virtual entity with a certain number of members or shareholders that decides to spend money and make code changes based on a majority of, say, 67%. The members collectively decide how the organization allocates funds. The method of allocating funds may be bounties, salaries, or more attractive mechanisms such as rewarding work with internal currency. This essentially replicates the legal structure of a traditional company or non-profit organization, using only cryptographic blockchain technology to enforce it. To date, much of the discussion around DAOs has been around the "capitalist" model of a "decentralized autonomous corporation" (DAC) with shareholders who receive dividends and tradable shares; as an alternative, an entity described as a "decentralized autonomous community" would give all members equal power in decision-making and require a 67% majority to add or remove members. The rule that each person can only have one membership would need to be enforced by the group.

Below is an outline of how to implement a DO in code. The simplest design is a piece of code that can modify itself if two-thirds of the members agree. Although the code is theoretically immutable, it is still easy to circumvent obstacles and make the code modifiable by placing the main body of the code in a separate contract and pointing the address of the contract call to a modifiable storage. In a simple implementation of such a DAO contract, there are three types of transactions, distinguished by the data provided by the transaction:

[0,i,K,V] registers a change proposal with index i to the contents of memory addresses indexed K to v.

[1,i] registers a vote for proposal i.

[2,i] If there are enough votes, confirm proposal i.

The contract would then have specific terms for each of these. It would maintain a record of all changes to open storage and a table of who voted. There would also be a table of all members. When a change to any storage content is approved by a two-thirds majority, a final transaction would execute the change. A more sophisticated framework would add built-in election functionality to enable things like sending transactions, adding and removing members, and even provide voting representation in a delegated democracy style (i.e. anyone can delegate to another person to vote on their behalf, and the delegation is transferable, so if A delegates to B and then B delegates to C then C will determine A's vote). This design would allow the DAO to grow organically as a decentralized community, allowing people to eventually delegate the task of selecting the right people to experts, unlike the current system where experts can easily come and go over time as community members change sides. An alternative model would be a decentralized company, where any account can own from zero to more shares, and decisions require a two-thirds majority of shares to agree. A complete framework would include asset management functionality - the ability to submit orders to buy or sell shares and the ability to accept such orders (assuming there is an order matching mechanism in the contract). Representation still exists in the form of delegated democracy, giving rise to the concept of a "board of directors".

More advanced organizational governance mechanisms may be implemented in the future; for now a decentralized organization (DO) can be described starting with a decentralized autonomous organization (DAO). The distinction between a DO and a DAO is fuzzy, a rough dividing line is whether governance can be achieved through a political-like process or an "automatic" process, a good intuitive test is the "no common language" criterion: can the organization function properly if the two members do not speak the same language? Obviously, a simple traditional holding company will fail, while something like the Bitcoin protocol is likely to succeed, and Robin Hanson's "futarchy", a mechanism for organizational governance through prediction markets is a real good example of what "self-governing" governance might look like. Note that one need not assume that all DAOs are superior to all DOs; autonomy is just a paradigm that has great advantages in some specific scenarios, but may not work in other places, and many semi-DAOs may exist.

Further applications

1.Savings wallet . Suppose Alice wants to ensure the safety of her funds, but she is worried about losing or having her private key stolen by a hacker. She puts ARI coins into a contract with Bob, as shown below. This contract is a bank:

Aurei can withdraw up to 1.2% of ARI coins per day and trade them on the exchange in seconds .

2.Crop insurance . One can easily create a derivative contract with weather conditions as input instead of any price index. If a farmer in Iowa buys a derivative that pays out based on the inverse of rainfall in Iowa, then if there

is a drought, the farmer will automatically receive a payout and if there is enough rain he will be happy because his crops will do well.

3.A decentralized data publisher . For difference-based financial contracts, it is actually possible to decentralize the data publisher through the "Schelling Point" protocol. Schelling Point works as follows: N parties provide input values to the system for a specified data (such as the ETH/USD price), all values are sorted, and each node that provides between 5% and 10% of the values will be rewarded. Everyone has an incentive to provide the answer that others will provide. The answer that a large number of players can really agree on is obviously the correct answer by default. This constructs a decentralized protocol that can theoretically provide many values, including the ETH/USD price, the temperature in Berlin, and even the result of a particularly difficult calculation.

4.Cloud computing . EVM technology can also be used to create a verifiable computing environment that allows users to invite others to perform computations and then selectively request proof that the computations were completed correctly at certain randomly selected checkpoints. This makes it possible to create a cloud computing market that any user can participate in with their desktop, laptop, or dedicated server, and on-site inspections and security deposits can be used to ensure that the system is trustworthy (i.e., no node can profit from cheating). Although such a system may not be suitable for all tasks; for example, tasks that require advanced inter-process communication are not easy to complete on a large cloud of nodes. However, some other tasks are easy to implement in parallel; projects such as SETI@home, folding@home, and genetic algorithms are easy to carry out on such a platform.

5. Peer-to-peer gambling . Any number of peer-to-peer gambling protocols can be moved to the Aurei blockchain, such as Frank Stajano and Richard Clayton's Cyberdice. The simplest gambling protocol is actually a simple contract that bets on the difference between the hash value and the guess value of the next block. More complex gambling protocols can be created based on this to achieve nearly zero-fee and fraud-free gambling services.

6. Prediction markets . Prediction markets will be easy to implement with either oracles or Schellingcoins, and prediction markets with Schellingcoins may prove to be the first mainstream “futarchy” application as a decentralized organization management protocol.

7. On-chain decentralized market, based on identity and reputation system. Market development in the emerging field of digital encrypted payment.

Miscellaneous and Concerns

Implementation of the Improved Ghost Protocol

The "Greedy Heaviest Observed Subtree" (GHOST) protocol is an innovation introduced by Yonatan Sompolinsky and Aviv Zohar in December 2013. The motivation for GHOST is that current fast-confirming blockchains suffer from low security due to a high rate of block obsolescence; since blocks take some time (let's say t) to propagate to the network, if miner A mines a block and then miner B happens to mine another block before A's block propagates to B, miner B's block will be obsolete and not contribute to the security of the network. In addition, there is a centralization problem here: if A is a mining pool with 30% of the network's hashrate and B has 10%, A will be at risk of producing obsolete blocks 70% of the time and B will be producing obsolete blocks 90% of the time. Therefore, if the scrap rate is high, A will simply be more efficient due to a higher share of hashrate, and combining these two factors, a blockchain with a fast block generation rate is likely to result in a mining pool with a share of hashrate that actually controls the mining process.

As described by Sompolinsky and Zohar, Ghost solves the first problem that reduces the security of the network by including obsolete blocks in the calculation of which chain is "longest"; that is, not only a block's parents and earlier ancestors, but also its obsolete descendants (called "uncles" in Aurei terminology) are included in the calculation of which block has the most proof-of-work backing it. We go beyond the protocol described by Sompolinsky and Zohar to solve the second problem - centralization

tendency. Aurei pays 10% of the reward to obsolete blocks that contribute to the confirmation of new blocks as "uncles". "Nephews" who include them in the calculation will receive 5 % of the reward. However, uncles are not rewarded with transaction fees. Aurei implements a simplified version of Ghost that only goes down to layer 5. Its characteristic is that the obsolete block can only be included in the calculation as an uncle block by the second to fifth generation descendants of its parent, but not by more distant descendants (such as the sixth generation descendants of the parent block, or the third generation descendants of the grandfather block). There are several reasons for this. First, the unconditional ghost protocol will bring too much complexity to the calculation of which uncle block of a given block is legitimate. Second, the unconditional ghost protocol with compensation used by Aurei deprives miners of the incentive to mine on the main chain instead of a public attacker's chain. Finally, calculations show that the five-layer ghost protocol with incentives achieves more than 95% efficiency even when the block time is 15s, and the miners with 25% of the computing power benefit less than 3% from centralization.

cost

Because every transaction published to the blockchain incurs a cost to download and verify, a regulated mechanism that includes transaction fees is needed to prevent spam. The default approach used by Bitcoin is purely voluntary transaction fees, relying on miners to act as gatekeepers and set a dynamic minimum fee. Because this approach is "market-based," allowing miners and transaction senders to determine prices based on supply and demand, it has been well accepted in the Bitcoin community. However, the

problem with this logic is that transaction processing is not a market; while it is intuitively attractive to interpret transaction processing as a service provided by miners to senders, the fact is that a miner's inclusion of a transaction is required to be processed by every node in the network, so the largest portion of the cost of transaction processing is borne by third parties rather than the miners who decide whether to include the transaction. As a result, the tragedy of the commons is very likely to occur.

However, this market-based mechanism loophole magically eliminates its own impact when a particular imprecise simplifying assumption is made. The argument is as follows. Assumptions:

1.A transaction takes k steps, providing a reward kR to any miner that includes the transaction, where R is set by the transaction publisher, and both k and R are (roughly) visible to the miner in advance.

2.The cost of each node processing each operation is C (that is, the efficiency of all nodes is the same).

3.There are N mining nodes, each with the same computing power (i.e. 1/N of the total network computing power).

4.There is no full node that does not mine.

Miners are willing to mine when the expected reward is greater than the cost. Thus, since miners have a 1/N chance of processing the next block, the expected benefit is kR/N, and the processing cost of the miner is simply kC. Thus, when kR/N > kC, that is, R > NC. Miners are willing to include transactions. Note that R is the per-step fee provided by the sender of the transaction, which is the lower limit of the benefit that miners can gain from

processing transactions. NC is the cost of processing an operation for the entire network. Therefore, miners are only motivated to include transactions whose benefits are greater than the costs. However, these assumptions have several important deviations from reality:

1.Because the extra verification time delays the block's propagation and thus increases the chance that the block will become invalid, the miner processing the transaction pays a higher cost than other validating nodes.

2.Full nodes that do not mine do exist.

3.In practice, the distribution of computing power may end up being extremely uneven.

4.Speculators, political enemies, and lunatics who are committed to destroying the network do exist, and they can cleverly set up contracts so that their costs are much lower than other validators. Point 1 above drives miners to include fewer transactions, and point 2 increases NC; therefore the effects of these two points at least partially offset each other. Points 3 and 4 are the main problems; as a solution we simply establish a floating upper limit: no block can contain more operations than BLK_LIMIT_FACTOR times the long-term exponential moving average. Specifically:

blk.oplimit = floor((blk.parent.oplimit * (EMAFACTOR - 1) + floor (parent.opcount * BLK_LIMIT_FACTOR)) /EMA_FACTOR)

BLK_LIMIT_FACTOR and EMA_FACTOR are constants that are temporarily set to 65536 and 1.5, but may be adjusted after further analysis.

Computation and Turing completeness

It is important to emphasize that the Aurei virtual machine is Turing complete; this means that EVM code can implement any conceivable computation, including infinite loops. EVM code can implement loops in two ways. First, the JUMP instruction allows the program to jump back to a previous point in the code, and there is also the JUMPI instruction that allows conditional jumps such as while x < 27: x = x * 2. Second, contracts can call other contracts, with the potential to implement loops through recursion. This naturally leads to the question: Can malicious users force miners and full nodes to shut down by forcing them into infinite loops? This question arises because of a problem in computer science called the halting problem: in general, there is no way to know whether a given program will finish running in a finite amount of time.

As mentioned in the state transition section, our solution solves the problem by setting a maximum number of computation steps to be executed for each transaction, if exceeded the computation is reverted but the fee is still paid. Messages work in the same way. To show the motivation behind this solution, consider the following example:

An attacker creates a contract that runs an infinite loop, then sends a transaction to a miner that activates the loop. The miner will process the transaction, running the infinite loop until the gas runs out. Even if the transaction is stopped halfway due to gas exhaustion, the transaction is still correct (going back to the original point) and the miner still earns the fees for each step of the calculation from the attacker.

An attacker creates a very long infinite loop with the intention of forcing miners to keep calculating for so long that several blocks are generated before the calculation is completed and the miners cannot include transactions to earn fees. However, the attacker needs to publish a STARTGAS value to limit the number of executable steps, so the miners will know in advance that the calculation will take too many steps.

☒An attacker sees a contract with a format like send(A, self.storage); self.storage = 0 and sends a transaction with just enough fees to execute the first step but not the second step (i.e. withdraw the funds but not reduce the account balance). Contract authors don't need to worry about defending against such attacks, because if execution is stopped midway all changes are reverted.

A financial contract works by extracting the median of nine dedicated data publishers to minimize the risk that an attacker takes over one of the data providers and then turns the data provider, which is designed to be changeable as described in the DAO chapter, into running an infinite loop in an attempt to force any attempt to claim funds from the financial contract to abort due to gas exhaustion. However, the financial contract can set gas limits in messages to prevent this problem. The alternative to Turing completeness is Turing incompleteness, where JUMP and JUMPI instructions do not exist and only one copy of each contract is allowed to exist in the call stack at a given time. In such a system, the fee system described above and the uncertainty surrounding the efficiency of our solution may not be needed, because the cost of executing a contract will be determined by its size. Moreover, Turing incompleteness is not even a big

limitation, of all the example contracts we have envisioned internally, only one so far requires a loop, and even this loop can be replaced by 26 repetitions of a single line of code. Given the serious troubles and limited benefits of Turing completeness, why not simply use a Turing incomplete language? In fact, Turing incompleteness is far from a neat solution. Why? Consider the following contract:

C0 : call (C1); call(C1);
C1 : call (C2); call (C2);
C2 : call (C3); call (C3);
...
C49 : call (C50); call(C50);

C50 : (Do a Turing machine step calculation and record the result in the long-term storage of the contract)

Now, send such a transaction to A. Thus, in 51 transactions, we have a contract that takes 2^50 steps to compute. Miners may try to detect such logic bombs in advance by maintaining a maximum number of executable steps for each contract and calculating the number of possible execution steps for contracts that recursively call other contracts, but this will prohibit miners from creating contracts that create other contracts (because the creation and execution of the above 26 contracts can easily be put into a single contract). Another problem is that the address field of a message is a variable, so in general it may not even be possible to know in advance which other contract a contract will call. So, in the end we have a surprising conclusion: Turing completeness is surprisingly easy to manage, and Turing incompleteness is surprisingly difficult to manage in the absence of the same control - so why not make the protocol Turing complete?

Currency and issuance

The Aurei network includes its own built-in currency, ARI coin, which plays a dual role, providing the main liquidity for various digital asset transactions, and more importantly, providing a mechanism for paying transaction fees. For convenience and to avoid future disputes, the names of different denominations will be set in advance:

This should be thought of as an extension of the concepts of “yuan” and “cent” or “bitcoin” and “satoshi”. In the near future, we expect “ARI coins” to be used for normal transactions, “finneys” to be used for microtransactions, and “saabs” and “weis” to be used for discussions about fees and protocol implementation.

The distribution mode is as follows:

Through the sale, ARI coins will be sold at a price of 1337-2000 ARI coins per BTC. A mechanism designed to raise funds for the Aurei organization and pay developers has been successfully used on some other cryptocurrency platforms. Early buyers will enjoy a large discount, and the BTC raised from the sale will be used entirely to pay salaries and rewards for developers and researchers, as well as projects invested in the cryptocurrency ecosystem.

0.14285714 $ per coin (100 million is the total number of coins issued) will be allocated to early contributors who participated in the development before BTC financing or other deterministic financing is successful, and another 0.14285714 $ per coin will be allocated to the value before the issuance is completed .

100 million ARI will be issued in a fixed amount from the time of launch to facilitate subsequent value maintenance and improvement.

Release Breakdown

The perpetual linear growth model reduces the risk of excessive concentration of wealth in Bitcoin and gives fair opportunities to people living in the present and future to acquire currency, while maintaining incentives to acquire and hold ARI coins, because the "money supply growth rate" tends to zero in the long run. We also infer that as time goes by, there will always be coins lost due to carelessness and death. Assuming that coins are lost as a fixed proportion of the annual money supply, the total money supply in circulation will eventually stabilize at a value equal to the annual money supply divided by the loss rate.

In addition to the linear issuance method, like Bitcoin, the supply growth rate of ARI coins also tends to increase by 5.76 % -867 % per year in the long run .

Scalability

Scalability is a common concern for Aurei, and like Bitcoin, Aurei suffers from the dilemma that every transaction needs to be processed by every node in the network. Bitcoin's current blockchain size is about 20GB, growing at a rate of 1MB per hour. If the Bitcoin network were to handle Visa-level transactions at 2,000 tps, it would grow at a rate of 1MB every three seconds (1GB per hour, 8TB per year). Aurei may also experience a similar or even worse growth pattern, because there are many applications on top of the Aurei blockchain, rather than just a simple currency like Bitcoin, but the fact that Aurei full nodes only need to store state rather than the complete blockchain history makes the situation better.

The problem with large blockchains is the risk of centralization. If the blockchain size increases to, say, 100TB, the likely scenario is that only a very small number of large merchants will run full nodes, while regular users will use light SPV nodes. This raises concerns about the risk of full nodes colluding to commit fraud for profit (e.g. changing the block reward to give themselves BTC). Light nodes will have no way to detect such fraud immediately. Of course, there might be at least one honest full node, and after a few hours the news about the fraud would leak through channels like Reddit, but by then it would be too late: no matter how hard regular users try to invalidate blocks that have already been produced, they will run into a huge, infeasible coordination problem of the same scale as launching a successful 51% attack. In Bitcoin, this is a problem right now, but a change suggested by Peter Todd could alleviate this problem.

In the near term, Aurei will use two additional strategies to address this problem. First, because of the blockchain-based mining algorithm, at least every miner is forced to be a full node, which guarantees a certain number of full nodes. Second, and more importantly, after processing each transaction, we include the root of an intermediate state tree in the blockchain. Even if block verification is centralized, as long as there is an honest validator, the centralization problem can be avoided through a verification protocol. If a miner publishes an incorrect block, it is either malformed or the state S[n] is incorrect. Since S[0] is correct, there must be a first incorrect state S[i] but S[i-1] is correct. The validator will provide the index i, along with the subset of Patricia tree nodes required to process APPLY(S[i-1],TX[i]) -> S[i]. These nodes will be instructed to perform this part of the calculation and see if the resulting S[i] is consistent with the previously provided value.

In addition, what is more complicated is that malicious miners publish incomplete blocks to attack, resulting in insufficient information to determine whether the block is correct. The solution is a challenge-response protocol: the verification node challenges the target transaction index, and the light node that receives the challenge information will cancel the trust in the corresponding block until another miner or validator provides a Patricia node subset as correct evidence.

Overview: Decentralized Applications

The contract mechanism described above enables anyone to build a command-line application running on a virtual machine with network-wide consensus (essentially), which can modify a network-wide accessible state as its "hard disk". However, for most people, the command-line interface used as a transaction sending mechanism is not user-friendly enough to make decentralization an attractive alternative. Ultimately, a complete "decentralized application" should include both the underlying business logic components [whether fully implemented in Aurei, using Aurei in combination with other systems (such as a P2P messaging layer, one of which is being planned to be built into the Aurei client), or just other systems] and the upper-level graphical user interface components. The Aurei client is designed to be a web browser, but includes support for the "eth" Javascript API object, which can be used by special web pages viewed in the client to interact with the Aurei blockchain. From the perspective of "traditional" web pages, these pages are completely static content, because blockchains and other decentralized protocols will completely replace servers in handling user-initiated requests. Finally, it is hoped that decentralized protocols will use Aurei to store web pages themselves in some way.

in conclusion

The Aurei protocol was originally conceived as an upgraded cryptocurrency that provides advanced features such as on-chain contracts, withdrawal limits and financial contracts, gambling markets, etc. through a highly general language. The Aurei protocol will not directly "support" any application, but the existence of a Turing-complete programming language means that in theory arbitrary contracts can be created for any transaction type and application. However, what is more interesting about Aurei is that the Aurei protocol goes further than just currency. Protocols and decentralized applications built around decentralized storage, decentralized computing and decentralized prediction markets, as well as dozens of similar concepts, have the potential to fundamentally improve the efficiency of the computing industry and provide strong support for other P2P protocols by adding an economic layer for the first time. In the end, there will also be a large number of applications that have nothing to do with money.

The concept of arbitrary state transitions implemented by the Aurei protocol provides a platform with unique potential; unlike closed protocols designed for a single purpose such as data storage, gambling or finance, Aurei is open by design, and we believe it is extremely suitable as a base layer for the vast number of financial and non-financial protocols that will emerge in the coming years.

Notes and Further Reading

annotation

1. An experienced reader will note that a Bitcoin address is in fact the hash of an elliptic curve public key, not the public key itself, however in fact it is perfectly reasonable to call a public key hash a public key from a cryptographic point of view. This is because Bitcoin cryptography can be thought of as a custom digital signature algorithm, where the public key consists of the hash of the elliptic curve public key, the signature consists of the elliptic curve public key concatenated with the elliptic curve signature, and the verification algorithm consists of checking the elliptic curve public key with the elliptic curve public key hash provided as the public key, and then verifying the elliptic curve signature with the elliptic curve public key.

2. Technically, the median of the first 11 blocks.

3. Internally, both 2 and "CHARLIE" are numbers, the latter one has a huge base256 encoding format, and the number can range from 0 to 2^256-1.

ARI Aurei Digital asset exchange center welcomes you to join