BY₵OI.io

MultiChain Blockchain

Private, is centralised and publicly visible — this is the White Paper. This paper is based on the format of the original version by Dr Gideon Greenspan, Founder and CEO, Coin Sciences Ltd.
It can be sourced here.

Bycoi.io is owned, operated and managed by BCB Coinstrike Limited, 71-75 Shelton Street, London, Greater London, WC2H 9JQ, United Kingdom with registration number 11982321.

This version is dated 5 June 2019.

This paper is issued with a single purpose of giving our users the insight to what Bycoi (BYC) is, and what differentiates us from other blockchain technologies. We hope that you also will gain a better understanding, in general, about crypto-currency, mining, smart contracts and blockchain technology in general.

This paper is not issued for the purposes of either an ICO or crowdfunding.

MultiChain and Bycoi.io

General

Bycoi have chosen MultiChain Open Source platform to build, and develop its centralised private, but publicly viewed blockchain and virtual crypto-currency Bycoi (BYC). It is operating with consensus of users sending and receiving the virtual crypto-currency. 120 billion Bycoi's was issued as assets when the blockchain was created. Transactions times to validate a block is between 1-2 seconds. There are 10 global administrators combined consensus in order for any smart filters with build in rules to be adjusted. The blockchain is immutable and transactions irreversible. It is publicly viewable using a public explorer.

Mining

Mining (or more generally, block validation) is performed automatically in MultiChain, on the nodes which have mine permissions. It is hence very environmentally friendly and does not require a massive number of servers. By restricting mining to a set of identifiable entities, MultiChain resolves the dilemma posed by private blockchains, in which one participant can monopolize the mining process. The solution lies in a constraint on the number of blocks which may be created by the same miner within a given window. MultiChain implements this scheme using a parameter called mining diversity, which is constrained by 0 <= mining diversity <= 1. The validity of a block is verified in the following way:

1. Apply all the permissions changes defined by transactions in the block in order.
2. Count the number of permitted miners who are defined after applying those changes.
3. Multiply miners by mining diversity, rounding up to get spacing.
4. If the miner of this block mined one off the previous spacing blocks, the block is invalid.

This enforces a round-robin schedule, in which the permitted miners must create blocks in rotation in order to generate a valid blockchain. The mining diversity parameter defines the strictness of the scheme, i.e. the proportion of permitted miners who would need to collude in order to undermine the network. A value of 1 ensures that every permitted miner is included in the rotation, whereas 0 represents no restriction at all. In general, higher values are safer, but a value too close to 1 can cause the blockchain to freeze up if some miners become inactive. We suggest a value of 0.75 as a reasonable compromise. To conserve resources, nodes will not attempt to mine on a chain in which they already mined one of the previous spacing blocks.

Security

The creating of wallet in bycoi.io is made using public cryptography with the highest level of security including 2 step authentications using email and mobile phone verification accessing wallets. We use PBKDF2 algorithm with 1000 iterations of hashing password. No private keys are stored in the blockchain. The blockchain only allows user wallets to connect that is fully KYC verified and integrated to their wallets, where IDs are verified using 3D face recognition technology provided by Jumio.

Background crypto-currency

In December 2018 there were well over 2500 recognized crypto currencies created by companies, organizations, individuals as well as governments. Compare this to the 180 global currencies that is issued solely by federal banks in respective jurisdictions and recognized by the United Nations. Although some crypto-currencies are fake and scams, using simple scripts to convince the public there is a real blockchain with real technology behind them, the simplest way to check if a blockchain is real is to see what blockchain technology the crypto-currency is using. All issued crypto-currency that is listed in any type of legal and accredited exchange is using open source blockchain technology that is publicly accessible.

Bitcoin

Bitcoin is now recognized as a cheap, rapid and reliable method for moving economic value across the Internet in a peer-to-peer manner. Aside from a brief fork between incompatible versions in March 2013, the Bitcoin network has been operating continuously and smoothly for 10 years. Although there have been losses and thefts of Bitcoins belonging to individual holders, the network itself has never been successfully attacked or impeded.

Despite Bitcoin's many technical achievements, it is far from reaching mainstream consumer or business adoption. Judging by the current trend in transaction volumes, the sluggish growth of 1 Bitcoin usage shows no sign of changing in the foreseeable future. This is despite the availability of many easy to use Bitcoin wallets, and the fact that Bitcoin can now be spent online at many mainstream businesses.

There are many possible causes of Bitcoin's slow adoption, including: (a) end-user satisfaction with existing payment systems, (b) the practical difficulty of purchasing Bitcoins, (c) the volatility of Bitcoin's value relative to government issued currencies, (d) the perception that Bitcoin is insecure, (e) questions over Bitcoin's legal status, (f) the irreversible and unforgiving nature of Bitcoin transactions, and (g) a lack of support for Bitcoin in the mainstream financial sector.

In the absence of end-user adoption, many have suggested that Bitcoin could help improve internal processes within the traditional financial sector, by lowering costs, reducing settlement times and eliminating intermediaries. One immediate theoretical possibility is using Bitcoin as a currency and conduit for rapid interbank settlement.

However, the volatility of Bitcoin's value relative to government issued currencies renders this unworkable in practice. A more promising direction is to use Bitcoin's infrastructure to transact in assets other than Bitcoin itself.

Ownership

Simplified chain of ownership is illustrated in the Bitcoin whitepaper. In practice, a transaction can have more than one input and more than one output. In the blockchain, Bitcoins are registered to Bitcoin addresses. Creating a Bitcoin address requires nothing more than picking a random valid private key and computing the corresponding Bitcoin address. This computation can be done in a split second. But the reverse, computing the private key of a given Bitcoin address, is mathematically unfeasible. Users can tell others or make public a Bitcoin address without compromising its corresponding private key. Moreover, the number of valid private keys is so vast that it is extremely unlikely someone will compute a key-pair that is already in use and has funds. The vast number of valid private keys makes it unfeasible that brute force could be used to compromise a private key. To be able to spend their Bitcoins, the owner must know the corresponding private key and digitally sign the transaction. The network verifies the signature using the public key.

If the private key is lost, the Bitcoin network will not recognize any other evidence of ownership; the coins are then unusable, and effectively lost. For example, in 2013 one user claimed to have lost 7,500 Bitcoins, worth $7.5 million at the time, when he accidentally discarded a hard drive containing his private key.

About 20% of all Bitcoins are believed to be lost. They would have a market value of about $20 billion at July 2018 prices. Approximately one million Bitcoins, valued at $7 billion in July 2018, have been stolen.

The Binance exchange, which stores Bitcoin and other crypto-currencies for members, said hackers took 7,000 Bitcoins in one go on in May 2019.

Bitcoin's shortcomings

Notwithstanding the promise of asset tokenization protocols, there are several reasons why the Bitcoin blockchain is not yet suitable for institutional financial transactions.

The problems can be divided into two groups, the first of which relates to scalability and cost:

● Limited capacity. The Bitcoin blockchain currently supports around 300,000 transactions per day, as determined by its maximum block size of 1MB. This capacity must be shared between all network users and is clearly insufficient for many financial applications. For example, the Visa network currently handles 150 million transactions per day in the USA alone.

While the maximum block size will likely increase in future, there is an ongoing debate over how quickly this can happen without driving out regular users and increasing the frequency of forks in network consensus. In any event, institutional users cannot control the pace of this change, which will ultimately be determined by miner adoption.

● Transaction costs. The fee charts and tables are in US dollars per transaction and in Satoshis per byte. Currently (May 2019) the fee is 92 Satoshi per byte. In November and December 2017 there was a peak in the fees peaking at almost 1000 satoshi per byte. This would equal around 20 USD and is collected by the miner of the block in which that transaction is confirmed. While this fee is optional, transactions with lower fees can encounter significant delays in confirmation. This sum is already a nontrivial tax on transactions of small monetary value. Furthermore, when the demand for Bitcoin transactions outgrows the supply of available block space (see previous point), this fee may increase substantially, as transactions are forced to bid with each other to compete for inclusion in a block.

● Irrelevant data. Institutions deploying over the Bitcoin network need to process and store a large quantity of information that is of no interest to them. When a new Bitcoin node is launched, it first downloads, verifies and stores the entire history of all Bitcoin transactions. Going forward, it must also verify all new transactions and blocks created, even though most are of no relevance to the user of that node. This problem is avoided by lightweight nodes, which can transact over the blockchain without storing it. However, their weaker security renders them unsuitable for serving as the backbone of an institutional system. Currently (May 2019) there are 9468 nodes in the Bitcoin network.

The second group of problems relates to privacy and security:

● Mining risks. Bitcoin's proof of work mining is an open global race to solve the difficult mathematical problem required to create a new block. While this process is well suited for a general-purpose decentralised network, it entails several risks for institutional users:

(a) the unpredictable delay for transaction confirmations, with block creation times defined by a Poisson distribution with average 10 minutes, (b) the risk of some miners refusing to confirm institutional transactions for ideological or economic reasons, (c) the potential for a 51% attack, where a group of miners controlling over half of the network's computational power collude to rewrite a significant period of the blockchain's recent history . Although such an attack is unlikely to occur, it clashes with the institutional need for transactions to be absolutely irreversible once settlement has taken place. China today controls 81% of the mining pools and Czech Republic 10%. The 3 largest mining pools controls over 50% of the mining servers today.

● Lack of privacy. By design, all Bitcoin transactions are visible to all network nodes and therefore to the entire world via blockchain explorers such as blockchain.info. This public aspect is mitigated by the fact that Bitcoin addresses cannot easily be connected to their real-world owners. Nonetheless the existence and rate of transactions cannot be hidden, and participants run the risk of their identities being revealed at some point in future, at which point their entire transaction history could be retroactively inferred.

● Openness. Anybody with an Internet connection is able to connect to the Bitcoin network and transact with other participants. This makes Bitcoin an attractive conduit for illegal transactions, since Know Your Customer (KYC) checks cannot be enforced at the network level. While it is certainly possible for regulated institutions to ensure that they (or their customers) only transact with known counterparties, this requires every transaction to be individually vetted, creating a significant burden in terms of architecture and workflow.

Some myths about Bitcoin.

Myth 1: Data on the Blockchain is secure!

This is the most common misperception. The mistaken perception is that the data within the blockchain is somehow cryptographically stored and therefore "secure", i.e. no one without proper authorization can view or access the data once it is on a blockchain. Thus, you can securely store your bank account, password and social security number etc. on the blockchain without fear of it being hacked. Nothing could be further from the truth!

In the case of a public blockchain the data that is stored on a blockchain is actually visible to everyone that is part of the blockchain network, i.e. every node in the public blockchain network has a local copy of the entire blockchain on their node and can view the data contents of the blocks.

Data stored within the Bitcoin blockchain can be viewed by anyone! Every public blockchain has the same visibility.

Thus, public blockchains are not great for storing sensitive or private information (like your password, social security number or bank account number), since everyone can view the contents of the blockchain.

When people say that data on a blockchain is "secure", they simply mean that it is "immutable". i.e. no one can alter the data in the blockchain without someone being aware that the data was altered.

Data on a blockchain is not secure. It is simply immutable.

The word "secure" means different things to different people. In the context of blockchain the word "secure" simply means "immutable" and does not imply that the data is secure from someone trying to read or access it.

Side note: You can browse through the entire Bitcoin blockchain on https://www.blockchain.com/explorer. The site is connected to the Bitcoin blockchain and allows you to view any block on the Bitcoin blockchain, and also see the individual Bitcoin transactions within each block.

Myth 2: Blockchain is great for storing data!

Blockchain is actually not great for storing large amounts of data.

The distributed nature of the blockchain means that every node that is part of the blockchain network has a full copy of the blockchain. If the blockchain is used to store large files (e.g. images, video etc.) the size of the blockchain would be huge and each node would have to replicate the entire blockchain data on itself, making it inefficient.

In reality the blockchain is great for recording transactional data. Typically, large data files are stored externally to the blockchain, using some context-based location distributed file system (e.g. IPFS, Swarm, SAFE Network, perkeep etc.) and the hashed address of the data file is stored on the blockchain.

Myth 3: Smart Contracts are regular real-world contracts stored on the blockchain!

Smart Contracts have absolutely no relation to real world contracts. Smart contracts are simply, computer programs that are stored on the blockchain and can be executed on the blockchain.

Smart contracts are written in a programming language like Solidity or Serpent in Ethereum blockchain and Go or Java in Hyperledger Fabric blockchain. Smart contracts are executed on the Ethereum blockchain via the EVM (Ethereum Virtual Machine). On the Hyperledger blockchain the chain code is executed within Docker containers.

The concept of smart contracts was introduced as part of the Ethereum blockchain (Ethereum is considered the second generation of cryptocurrency). Adding the EVM to the Ethereum blockchain platform allowed one to expand the capabilities and uses cases of a Blockchain by enabling computer programs to be stored and run on the blockchain.

Bitcoin blockchain (considered the first generation of cryptocurrency) does not have the concept of smart contracts i.e. You cannot create smart contracts on the Bitcoin blockchain.

Side note: While Smart Contracts in Ethereum blockchain and Chain code in Hyperledger Fabric blockchain are similar; in that they provide the ability to execute computer programs on the blockchain, they do this using very different mechanisms.

Smart contracts actually reside on the Ethereum blockchain as byte code i.e. the smart contract program written in Solidity programming language, is compiled by the Solidity compiler into byte code, and this byte code is stored on the Ethereum blockchain. The smart contract runs on the Ethereum blockchain itself and is executed at each node of the Ethereum blockchain by the EVM (Ethereum Virtual Machine).

By contrast in Hyperledger Fabric Chain code programs, by design, are kept insulated from the blockchain. Chain code programs are written in Go or Java and are executed within separate Docker containers that run on each node, and not on the Blockchain. This enables Hyperledger Fabric's modular architecture which allows consensus algorithms to be plug and play features.

Myth 4: Bitcoin is a collection of digital coins

Bitcoin is not a collection of digital coins. In fact, a Bitcoin does not actually exist at all!

A Bitcoin only exists as part of a transactional record.

While this may sound confusing, there is no such thing as “newly minted” Bitcoins. A miner who spends computational energy to mine the next block in the Bitcoin blockchain is rewarded with new Bitcoins. But the way this happens is that there is simply a transaction record that says “12.5 Bitcoins are transferred to miner’s Bitcoin wallet”, and such a transaction is considered a valid transaction by the blockchain.

It is important to understand that these “12.5 Bitcoins” that were transferred to the miner’s Bitcoin wallet address do not come from anywhere i.e. it’s not like there is a “treasury” of “digital coins” from which the miner is rewarded. The 12.5 Bitcoins never existed before and will never exist. The only thing that exists is a (valid) transactional record that 12.5 Bitcoins are transferred to the miner’s wallet i.e. Bitcoins simply exists as a transactional record, and not as a real digital coin.

Side note: To connect to the Bitcoin blockchain you do not have to download the entire blockchain (the current size of the Bitcoin blockchain is upwards of 210 GB as of April 2019).

If you simply wish to interact with the Bitcoin blockchain i.e. to buy, send and receive Bitcoins, all you need is a Bitcoin wallet. You can download a Bitcoin wallet from Coinbase (https://www.coinbase.com), one of the most popular and trusted sites to buy, sell and manage Bitcoins.

However, it is important to understand that your Bitcoin wallet doesn’t actually hold any digital coins.

A Bitcoin wallet is simply a cryptographic key (address) and not a real store of Bitcoins, since as we mentioned earlier Bitcoins do not actually exist as digital coins (Bitcoins are simply a transactional record). So, your Bitcoin wallet will not contain digital coins, instead your Bitcoin wallet will simply contain a cryptographic key that will allow nodes to validate if you have the ability to spend Bitcoins as part of a valid transaction record.

If you do want to join the Bitcoin blockchain as a node, it would mean that you would need to download a copy of the entire Bitcoin blockchain on your computer (which can take several hours to first download). The only reason you would join a Bitcoin blockchain as a node is if you wanted to “mine” new Bitcoins. However, owing to the growing computational power required to “mine” new Bitcoin blocks, individual personal computers are inadequate; and typically you would connect your computer to a “mining pool” (which is a collection of computers that mine collectively for the next block, and divide the Bitcoin rewards within the mining pool).

Myth 5: Bitcoin is not used as a mainstream currency because governments see them as a threat!

The only thing standing in the way of Bitcoin being used as a mainstream currency is Bitcoin itself!

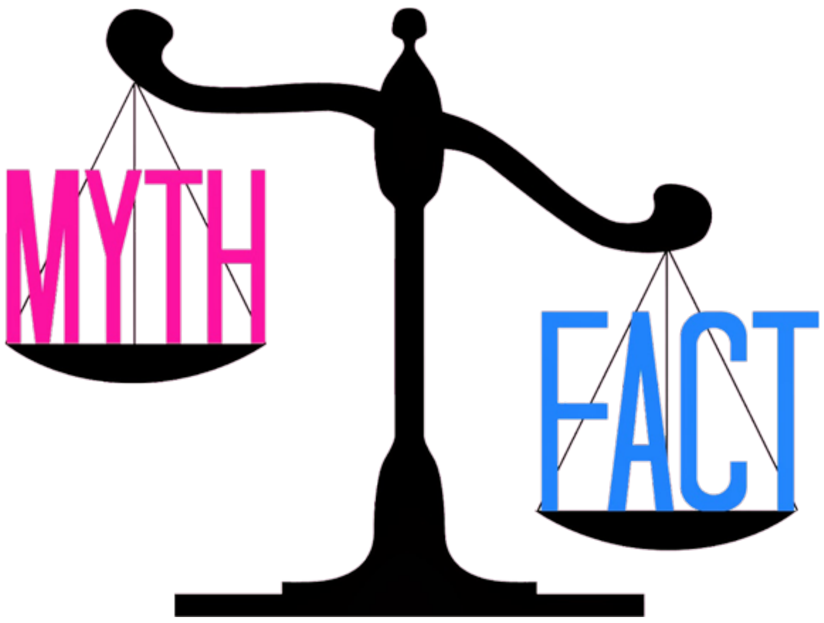
Bitcoin in its current form, has an inherent scalability issue, and can only process 7 transactions per second! Ethereum, the second largest cryptocurrency platform, is currently only able to manage 20 transactions per second.

By contrast Visa and PayPal can process 1667 and 193 transactions per second, respectively. MultiChain can process up to 1000 transactions per second.

Bitcoin's inherent limitation is because, by design it takes ten minutes to generate a new Block on the Bitcoin Blockchain, and every block has a size limitation of ~1MB. Also, if the Bitcoin blockchain were to process a thousand transactions per second, it would require all the nodes on the Blockchain to have high network bandwidth to be able to keep up with all the new records in the local copy of their blockchain.

This makes the Bitcoin blockchain great for use cases like money transfers, that do not need immediate transaction confirmations, and can wait for an hour or so before the transaction is confirmed. However, it is not suited for rapid and immediate transaction processing.

So, it is not government, legislative or regulatory hurdles, but the inherent scalability issue, which is the major reason why Bitcoin cannot, in its current implementation, be used as mainstream currency exchange.



Centralised or Decentralised blockchain?

Blockchains have garnered interest from investors from all over the world due to their incredible promise of being an incorruptible ledger. When most people think of blockchains, they are referring to the decentralised or public blockchains like Bitcoin which anyone can access and participate in. However, blockchain technology is not limited to being just decentralised as the centralised or private blockchains also have some advantages for corporations over the public ones. Private blockchains are useful for corporations who want to use the power of decentralised ledgers to improve the ongoing function. Let's take a look at an in-depth comparison of public and private blockchains.

Similarities Between centralised and Decentralised Blockchains

From a technological standpoint, both centralised and decentralised blockchains are very similar as both are distributed peer to peer networks where every node is responsible for storing and securing the shared ledger. Both public and private blockchains require a consensus mechanism (like proof-of-work, consensus or proof-of-stake) among nodes to establish a single ledger. Both of these types of blockchains also have to provide upper and lower bounds on the security and efficiency of the network.

Differences Between centralised and Decentralised Blockchains

The biggest factor that differentiates public blockchains from private ones is the pool of nodes that can participate in the network and make administrative changes to the network. So, for example, Bitcoin which is the largest public blockchain in the world has no barrier to entry when it comes to accessing the ledger and sharing computer power to execute its proof of work algorithm. By contrast, IBM's Hyperledger Fabric is more customizable in the sense that the organization that is deploying the blockchain has a say in every aspect of blockchain participation. Private blockchains are typically more restrictive in who they allow making changes to the ledger as they use the blockchain for the internal records. MultiChain uses global consensus of all administrators.

Advantages of Decentralised Blockchains

Decentralised blockchains like Bitcoin, which is the most popular blockchain in the world, have very high security because of the enormous amount of mining resources that go in to secure the network. That means to coordinate a successful attack on the Bitcoin network; a malicious actor would have to acquire a massive amount of resources which is economically unviable. Another advantage is that anyone can use the network to send funds to any part of the world without going through an intermediary.

Disadvantages of Decentralised Blockchains

Due to their public nature, public blockchains like Bitcoin are susceptible to all kinds of analyses that can reveal more information about the network participants making the blockchain less private. The massive amount of miners mining on the network means that difficulty needs to keep increasing thereby leading to mostly useless computations done by miners to outcompete each other. It is estimated that every Bitcoin transaction cost about as much electricity that is required to power an average home for eight days. Therefore, public ledgers are not very environment-friendly.

Advantages of centralised Blockchains

Centralised blockchains offer much more customizability and control over the network to the organization deploying it as they can decide who gets to participate in the network. That means that not as much resources have to be invested in competing to secure the network which makes centralised Blockchains more environmentally friendly compared to their Decentralised counterparts. This also means that they have higher overall throughput because they get to decide the hardware that the network runs on. In practice, this means that corporations could use private blockchains to store sensitive information among nodes that they trust. This allows them to use the incredible power of blockchains without having to make the sensitive information public.

Disadvantages of centralised Blockchains

Since there is not as much computing power securing the network as in the case of decentralised blockchains, centralised blockchains are less secure. It only requires a few of the nodes hosting the network to collude by amassing enough resources to hack the network. Also, since transactions are not publicly viewable, it is harder to verify the authenticity of the transactions for an outside party. Also, since private ledgers are normally not available for public use, they are of little use to anyone besides the corporations that deploy them. The biggest difference with MultiChain and Bycoi is that the ledgers are publicly viewable using standard explorer but still being a private blockchain. This gives full transparency in terms of following the AML and KYC regulations being enforced by governments around the world.

What are smart contracts?

In most discussions about blockchains, it doesn't take long for the notion of "smart contracts" to come up. In the popular imagination, smart contracts automate the execution of interparty interactions, without requiring a trusted intermediary. By expressing legal relationships in code rather than words, they promise to enable transactions to take place directly and without error, whether deliberate or not.

From a technical viewpoint, a smart contract is something more specific: computer code that lives on a blockchain and defines the rules for that chain's transactions. This description sounds simple enough, but behind it lies a great deal of variation in how these rules are expressed, executed and validated. When choosing an open source blockchain platform for a new application, the question "Does this platform support smart contracts?" isn't the right one to ask. Instead, we need to be asking: "What type of smart contracts does this platform support?"

Let's look at some of the major differences between smart contract approaches and the trade-offs they represent. We'll do this by looking at four popular enterprise blockchain platforms which support some form of customized on-chain code including our own MultiChain used by Bycoi.

First, IBM's Hyperledger Fabric, which calls its contracts "chain code". Second, our MultiChain platform, which introduced smart filters in version 2.0 in March of 2019. Third, Ethereum (and its permissioned Quorum and Burrow spin-offs), which popularized the "smart contract" name. And finally, R3 Corda, which references "contracts" in its transactions. Despite all of the different terminology, ultimately all of these refer to the same thing – application-specific code that defines the rules of a chain.

Before going any further, we should warn any reader that much of the following content is technical in nature and assumes some familiarity with general programming and database concepts. For good or bad, this cannot be avoided – without getting into the details it's impossible to make an informed decision about whether to use a blockchain for a particular project, and (if so) the right type of blockchain to use. Bycoi choose to use MultiChain due to its flexibility, security, enhanced smart filer system and the ease to build applications based on the technology including all new planned releases and upgrades as presented here.

Blockchain basics

Let's begin with some context. Imagine an application that is shared by multiple organizations, which is based on an underlying database. In a traditional centralised architecture, this database is hosted and administered by one single party which all of the participants trust, even if they do not trust each other. Transactions which modify the database are initiated only by applications on this central party's systems, often in response to messages received from the participants. The database simply does what it's told because the application is implicitly trusted to only send it transactions that make sense.

Blockchains provide an alternative way of managing a shared database, without a trusted intermediary. In a blockchain, each participant runs a "node" that holds a copy of the database and independently processes the transactions which modify it. Participants are identified using public keys or "addresses", each of which has a corresponding private key known only to the identity owner. While transactions can be created by any node, they are "digitally signed" by their initiator's private key in order to prove their origin.

Nodes connect to each other in a peer-to-peer fashion, rapidly propagating transactions and the "blocks" in which they are timestamped and confirmed across the network. The blockchain itself is literally a chain of these blocks, which forms an ordered log of every historical transaction. A "consensus algorithm" is used to ensure that all nodes reach agreement on the content of the blockchain, without requiring centralised control. (Note that some of this description does not apply to Corda, in which each node has only a partial copy of the database and there is no global blockchain. We'll talk more about that later on.)

In principle, any shared database application can be architected by using a blockchain at its core. But doing so creates a number of technical challenges which do not exist in a centralised scenario:

- Transaction rules. If any participant can directly change the database, how do we ensure that they follow the application's rules? What stops one user from corrupting the database's contents in a self-serving way?
- Determinism. Once these rules are defined, they will be applied multiple times by multiple nodes when processing transactions for their own copy of the database. How do we ensure that every node obtains exactly the same result?
- Conflict prevention. With no central coordination, how do we deal with two transactions that each follow the application's rules, but nonetheless conflict

with each other? Conflicts can stem from a deliberate attempt to game the system or be the innocent result of bad luck and timing.

So where do smart contracts, smart filters and chain code come in? Their core purpose is to work with a blockchain's underlying infrastructure in order to solve these challenges. Smart contracts are the decentralised equivalent of application code – instead of running in one central place, they run on multiple nodes in the blockchain, creating or validating the transactions which modify that database's contents.

Let's begin with transaction rules, the first of these challenges, and see how they are expressed in Fabric, MultiChain, Ethereum and Corda respectively.

Transaction rules

Transaction rules perform a specific function in blockchain-powered databases – restricting the transformation that can be performed on that database's state. This is necessary because a blockchain's transactions can be initiated by any of its participants, and these participants do not trust each other sufficiently to allow them to modify the database at will.

Let's see two examples of why transaction rules are needed. First, imagine a blockchain designed to aggregate and timestamp PDF documents that are published by its participants. In this case, nobody should have the right to remove or change documents, since doing so would undermine the entire purpose of the system – document persistence. Second, consider a blockchain representing a shared financial ledger, which keeps track of the balances of its users. We cannot allow a participant to arbitrarily inflate their own balance or take others' money away.

Inputs and outputs

Blockchain platforms rely on two broad approaches for expressing transaction rules. The first, which we call the "input–output model", is used in MultiChain and Corda. Here, transactions explicitly list the database rows or "states" which they delete and create, forming a set of "inputs" and "outputs" respectively. Modifying a row is expressed as the equivalent operation of deleting that row and creating a new one in its place.

Since database rows are only deleted in inputs and only created in outputs, every input must "spend" a previous transaction's output. The current state of the database is defined as the set of "unspent transaction outputs" or "UTXOs", i.e. outputs from previous transactions which have not yet been used. Transactions may also contain additional information, called "metadata", "commands" or "attachments", which don't become part of the database but help to define their meaning or purpose.

Given these three sets of inputs, outputs and metadata, the validity of a transaction in MultiChain or Corda is defined by some code which can perform arbitrary computations on those sets. This code can validate the transaction, or else return an error with a corresponding explanation. You can think of the input–output model as an automated "inspector" holding a checklist which ensures that transactions follow each and every rule. If the transaction fails any one of those checks, it will automatically be rejected by all of the nodes in the network.

It should be noted that, despite sharing the input–output model, MultiChain and Corda implement it very differently. In MultiChain, outputs can contain assets and/or data in JSON, text or binary format. The rules are defined in "transaction filters" or "stream filters", which can be set to check all transactions, or only those involving particular assets or groupings of data.

By contrast, a Corda output "state" is represented by an object in the Java or Kotlin programming language, with defined data fields. Corda's rules are defined in "contracts" which are attached to specific states, and a state's contract is only applied to transactions which contain that state in its inputs or outputs. This relates to Corda's unusual visibility model, in which transactions can only be seen by their counterparties or those whose subsequent transactions they affect.

Contracts and messages

The second approach, which we call the "contract–message model", is used in Hyperledger Fabric and Ethereum. Here, multiple "smart contracts" or "chain codes" can be created on the blockchain, and each has its own database and associated code. A contract's database can only be modified by its code, rather than directly by blockchain transactions. This design pattern is similar to the "encapsulation" of code and data in object-oriented programming.

With this model, a blockchain transaction begins as a message sent to a contract, with some optional parameters or data. The contract's code is executed in reaction to the message and parameters and is free to read and write its own database as part of that reaction. Contracts can also send messages to other contracts but cannot access each other's databases directly. In the language of relational databases, contracts act as enforced "stored procedures", where all access to the database goes via some predefined code.

Both Fabric and Quorum, a variation on Ethereum, complicate this picture by allowing a network to define multiple "channels" or "private states". The aim is to mitigate the problem of blockchain confidentiality by creating separate environments, each of which is only visible to a particular sub-group of participants. While this sounds promising in theory, in reality the contracts and data in each channel or private state are isolated from those in the others. As a result, in terms of smart contracts, these environments are equivalent to separate blockchains.

Example rules

Let's see how to implement the transaction rules for a single-asset financial ledger with these two models. Each row in MultiChain ledger's database has two columns, containing the owner's address and the quantity of the asset owned. In the input–output model, transactions must satisfy two conditions:

1. The total quantity of assets in a transaction's outputs has to match the total in its inputs. This prevents users from creating or deleting money arbitrarily.

2. Every transaction has to be signed by the owner of each of its inputs. This stops users from spending each other's money without permission.

Taken together, these two conditions are all that is needed to create a simple but very viable financial system.

In the contract–message model, the asset's contract supports a "send payment" message, which takes three parameters: the sender's address, recipient's address, and quantity to be sent. In response, the contract executes the following four steps:

1. Verify that the transaction was signed by the sender.
2. Check that the sender has sufficient funds.
3. Deduct the requested quantity from the sender's row.
4. Add that quantity to the recipient's row.

If either of the checks in the first two steps fails, the contract will abort, and no payment will be made.

Built-in rules

When it comes to transaction rules, there is one way in which MultiChain specifically differs from Fabric, Ethereum and Corda. Unlike these other platforms, MultiChain has several built-in abstractions that provide some basic building blocks for blockchain-driven applications, without requiring developers to write their own code. These abstractions cover three areas that are commonly needed: (a) dynamic permissions, (b) transferrable assets, and (c) data storage.

For example, MultiChain manages permissions for connecting to the network, sending and receiving transactions, creating assets or streams, or controlling the permissions of other users. Multiple fungible assets can be issued, transferred, retired or exchanged safely and atomically. Any number of "streams" can be created on a chain, for publishing, indexing and retrieving on-chain or off-chain data in JSON, text or binary formats.

When developing an application on MultiChain, it's possible to ignore this built-in functionality, and express transaction rules using smart filters only. However, smart filters are designed to work together with its built-in abstractions, by enabling their default behaviour to be restricted in customized ways. For example, the permission for certain activities might be controlled by specific global administrators, rather than the default behaviour where any administrator will do. The transfer of certain assets can be limited by time or require additional approval above a certain amount. The data in a particular stream can be validated to ensure that it consists only of JSON structures with required fields and values.

In all of these cases, smart filters create additional requirements for transactions to be validated, but do not remove the simple rules that are built in. This can help address one of the key challenges in blockchain applications: the fact that a bug in some on-chain code can lead to disastrous consequences. We've seen endless examples of this problem in the public Ethereum blockchain, most famously in the Demise of The DAO and the Parity multisignature bugs. Broader surveys have found a large number of common vulnerabilities in Ethereum smart contracts that enable attackers to steal or freeze other peoples' funds.

Of course, our MultiChain smart filters may contain bugs too, but their consequences are more limited in scope. For example, the built-in asset rules prevent one user from spending another's money, or accidentally making their own money disappear, no matter what other logic a smart filter contains. If a bug is found in a smart filter, it can be deactivated and replaced with a corrected version, while the ledger's basic integrity is protected.

Transaction rules

	Fabric	MultiChain	Ethereum	Corda
Model	Contract-message	Input-output	Contract-message	Input-output
Built-ins	None	Permission + assets + streams	None	None

Determinism

Let's move on to the next part of this showdown. No matter which approach we choose, the custom transaction rules of a blockchain application are expressed as computer code written by application developers. And unlike centralised applications, this code is going to be executed more than one time and in more than one place for each transaction. This is because multiple blockchain nodes belonging to different participants have to each verify and/or execute that transaction for themselves.

This repeated and redundant code execution introduces a new requirement that is rarely found in centralised applications: determinism. In the context of computation, determinism means that a piece of code will always give the same answer for the same parameters, no matter where and when it is run. This is absolutely crucial for code that interacts with a blockchain because, without determinism, the consensus between the nodes on that chain can catastrophically break down.

Let's see how this looks in practice, first in the input–output model. If two nodes have a different opinion about whether a transaction is valid, then one will accept a block containing that transaction and the other will not. Since every block explicitly links back to a previous block, this will create a permanent "fork" in the network, with one or more nodes not accepting the majority opinion about the entire blockchain's contents from that point on. The nodes in the minority will be cut off from the database's evolving state and will no longer be able to effectively use the application.

Now let's see what happens if consensus breaks down in the contract–message model. If two nodes have a different opinion about how a contract should respond to a particular message, this can lead to a difference in their databases' contents. This in turn can affect the contract's response to future messages, including messages it sends to other contracts. The end result is an increasing divergence between different nodes' view of the database's state. (The "state root" field in Ethereum blocks ensures that any difference in contracts' responses leads immediately to a fully catastrophic blockchain fork, rather than risking staying hidden for a period of time.)

Sources of non-determinism

So non-determinism in blockchain code is clearly a problem. But if the basic building blocks of computation, such as arithmetic, are deterministic, what do we have to worry about? Well, it turns out, quite a few things:

- Most obviously, random number generators, since by definition these are designed to produce a different result every time.
- Checking the current time, since nodes won't be processing transactions at exactly the same time, and in any event their clocks may be out of sync. (It's still possible to implement time-dependent rules by making reference to timestamps within the blockchain itself.)
- Querying external resources such as the Internet, disk files, or other programs running on a computer. These resources cannot be guaranteed to always give the same response and may become unavailable.
- Running multiple pieces of code in parallel "threads", since this leads to a "race condition" where the order in which these processes finish cannot be predicted.
- Performing any floating-point calculations which can give even minutely different answers on different computer processor architectures.

These four blockchain platforms employ several different approaches to avoiding these pitfalls.

Deterministic execution

Let's start with Ethereum, since its approach is the "purest". Ethereum contracts are expressed in a special-purpose format called "Ethereum bytecode", which is executed by the Ethereum Virtual Machine ("EVM"). Programmers do not write bytecode directly, but rather generate or "compile" it from a JavaScript-like programming language called Solidity. (Other languages used to be available but have since been deprecated.) Determinism is guaranteed by the fact that Solidity and Ethereum bytecode cannot encode any non-deterministic operations – it's that simple.

MultiChain filters and Corda contracts choose a different approach, by adapting existing programming languages and runtime environments. MultiChain uses JavaScript running in Google's V8 engine, which also forms the core of the Chrome browser and the Node.js platform, but with sources of non-determinism disabled. Similarly, Corda uses Java or Kotlin, both of which are compiled to "Java bytecode" which executes within a Java Virtual Machine ("JVM"). For now, Corda uses Oracle's standard non-deterministic JVM, but work is under way to integrate a deterministic version. In the meantime, Corda contract developers must take care not to allow non-determinism in their code.

How does Ethereum's purism compare with the evolutionary approach taken by MultiChain and Corda? The main advantage for Ethereum is risk minimization – a built-for-purpose virtual machine is less likely to contain an inadvertent source of non-determinism. While any such oversight could be fixed by a software update, it would be disruptive to any chain that was unlucky enough to encounter it. Ethereum's problem, however, is that Solidity and the EVM constitute a tiny and nascent ecosystem in the wider context of programming languages and runtime environments.

By comparison, JavaScript, and Java are the common two languages on GitHub, run on billions of digital devices, and have runtimes that have been optimized over decades. Presumably this is why the public Ethereum blockchain is considering a transition to eWASM, a deterministic fork of the emerging Web Assembly standard.

Determinism by endorsement

When it comes to determinism, Hyperledger Fabric adopts a completely different approach. In Fabric, when a "client" node wants to send a message to some chain code, it first sends that message to some "endorser" nodes. Each of these nodes executes the chain code independently, forming an opinion of the message's effect on that chain code's database. These opinions are sent back to the client together with a digital signature which constitutes a formal "endorsement". If the client receives enough endorsements of the intended outcome, it creates a transaction containing those endorsements, and broadcasts it for inclusion in the chain.

In order to guarantee determinism, each piece of chain code has an "endorsement policy" which defines exactly what level of approval is required in order to render its transactions valid. For example, one chain code's policy might state that endorsements are required from at least half of the blockchain's nodes. Another might require an endorsement from any one of three trusted parties. Either way, every node can independently check if the necessary endorsements were received.

To clarify the difference, determinism in most blockchain platforms is based on the question: "What is the result of running this code on this data?" – and we need to be absolutely sure that every node will answer this question identically. By contrast, determinism in Fabric is based on a different question: "Do enough endorsers agree on the result of running this code on this data?" Answering that is a rather simple matter of counting, and there's no room for non-determinism to creep in.

Fabric's determinism-by-endorsement has a number of interesting consequences. First, chain code can be written in many different programming languages, since these don't need to be adapted for determinism (Go, Java and JavaScript are currently supported). Second, chain code can be hidden from some of a blockchain's participants, since it only needs to be executed by clients and endorsers (the database itself is globally visible). Finally, and most notably, Fabric chain code can do things that are forbidden in other blockchain environments, such as checking the weather using an online web API. In the worst case, where every endorser gets a different answer from this API, the client will fail to obtain enough endorsements for any particular outcome, and no transaction will take place. (It should be noted that Fabric team members still recommend using deterministic logic inside chain code, in order to avoid surprises.)

What price does Fabric pay for this flexibility? If the purpose of a blockchain is to remove intermediaries from a shared database-driven application, then Fabric's reliance on endorsers takes a big step away from that goal. For the participants in the chain, it is no longer enough to follow the chain code's rules – they also need certain other nodes to agree that they have done so. Even worse, a malicious subset of endorsers could approve database changes that do not follow chain code at all. This gives endorsers much more power than the validators in regular blockchains, who can censor transactions but cannot violate the blockchain's rules. Blockchain application developers must decide whether this trade-off makes sense in their particular case.

Determinism

	Fabric	MultiChain	Ethereum	Corda
Model	Endorsements	Adapted runtime	Purpose-built VM	Adapted runtime
Languages	Go + Java + JavaScript	JavaScript	Solidity	Java + Kotlin
Code visibility	Counterparties + endorsers	Blockchain	Blockchain	Counterparties + dependents
Enforced	No	Yes	Yes	No (for now)

Conflict prevention

So far, we've discussed how different blockchain platforms express transaction rules in code, and how they deterministically ensure that every node applies those rules identically. Now it's time to talk about a third aspect of our showdown: How does each platform deal with the possibility that two transactions, which are valid in and of themselves, conflict with each other? In the simplest example, imagine that Alice has €10 in a financial ledger and broadcasts two transactions – one sending €8 to Bob, and the other sending €7 to Charlie. Clearly, only one of these transactions can be allowed to succeed.

Two models

We can begin by grouping MultiChain's and Corda's approach to this problem together. As described earlier, both of these use an input–output model for representing transactions and their rules, in which each transaction input spends a previous transaction output. This leads to a simple principle for preventing conflicts: Every output can only be spent once. MultiChain filters and Corda contracts can rely on their respective platforms to enforce this restriction absolutely. Since Alice's €10 is represented by a previous transaction output, this single-spend rule automatically stops her sending it to both Bob and Charlie.

Despite this similarity, it is important to point out a key difference in how MultiChain and Corda prevent conflicts. In MultiChain, every node sees every transaction and so can independently verify that each output is only spent once. Any transaction which performs a double spend against a previously confirmed transaction will be instantly and automatically rejected. By contrast, in Corda there is no global blockchain, so “notaries” are required to prevent these doubles spends. Every Corda output state is assigned to a notary, who has to sign any transaction spending that output, confirming it has not been spent before. A blockchain’s participants must trust notaries to follow this rule honestly, and malicious notaries can cause havoc at will. As with endorsements in Fabric, this “single-spend as a service” design has advantages in terms of confidentiality but reintroduces intermediaries, going against the blockchain grain. (It’s important to clarify that Corda notaries can be run by groups of participants using a consensus algorithm, so the integrity of the ledger can still be protected against individual bad actors).

Let’s move on to Ethereum. To recall, Ethereum uses contracts and messages rather than inputs and outputs. As a result, transaction conflicts such as Alice’s two payments are not immediately visible to the blockchain engine. Instead, they are detected and blocked by the contract which processes the transactions, after their order is confirmed on the chain. When processing each of Alice’s payments, the contract verifies whether her balance is sufficient. If the transaction paying €8 to Bob comes first, it will be processed as usual, leaving Alice with €2 in her account. As a result, when the contract processes the second transaction paying €7 to Charlie, it sees that Alice lacks the necessary funds and the transaction aborts.

Outputs vs contracts

So far, we’ve seen two different techniques for preventing conflicting transactions – single-spend outputs in MultiChain and Corda, and contract-based verification in Ethereum. So, which is better?

In order to help answer this question, let’s consider an example “1-of-2 multi-signature” account which holds €100 on behalf of Gavin and Helen, and allows either of them to spend that money independently. Gavin instructs his application to pay €80 to Donna, and a few seconds later, Helen wants to send €40 to Edward. Since there are insufficient funds for both payments, these transactions would inevitably conflict. In the event that both transactions are broadcast, the outcome will be determined by whichever makes it first into the chain. Note that unlike Alice’s example, this conflict is accidental, since no one is trying to break the application’s rules – they simply had unlucky timing.

In considering the likelihood of this conflict occurring, the key question is this: after Gavin sends his transaction, how long will it take Helen’s node to know that her payment might fail? The shorter this period is, the more likely Helen is to be stopped from attempting that payment, saving her and her application from a subsequent surprise.

With the input–output model, any conflict between transactions is directly visible to the blockchain platform, since the two transactions will be explicitly attempting to spend the same previous output. In MultiChain, this happens as soon as Gavin’s transaction has propagated to Helen’s node, usually in a second or less. In Corda, the output’s notary will refuse the request to sign Helen’s transaction, since it has already signed Gavin’s, so Helen will instantly know that her payment will fail. (Although if the

Corda notary is itself distributed, she may have to wait a few seconds for a reply.) Either way, there is no need to wait for a transaction to be confirmed and ordered in the blockchain.

What about the Ethereum's model? In this case, there is no immediate way for the blockchain platform to know that a conflict will occur. While Helen's node may see Gavin's transaction on the network, it cannot know how this will affect Helen's own transaction, since from its perspective these are simply two messages being sent to the same contract. Perhaps ten seconds later, once the final ordering of the conflicting transactions is confirmed on the blockchain, Helen's node will recalculate the actual instead of the expected outcome, and her application will update its display accordingly. In the meantime, both Gavin and Helen will be left in the dark.

But we shouldn't conclude from this that the input–output model always works best. Consider a variation on our example scenario, where both Gavin and Helen request smaller €40 payments from the original balance of €100, at exactly the same time. In the input–output model these transactions would conflict, since they are both spending the same database row containing that €100, and only one of the payments would succeed. But in Ethereum, both transactions would be successfully processed, irrespective of their final order, since the account contains sufficient funds for both. In this case, Ethereum more faithfully fulfils Gavin's and Helen's intentions.

Read-write sets

Finally, Fabric… Fabric's endorsement-based approach is a hybrid of these two above mentioned techniques. As explained earlier, when a Fabric "client" node wants to send a message to a contract, it first asks some endorsing nodes to execute that message on its behalf. The endorsing nodes do so in a similar way to Ethereum – running the contract against their local database – but this process is observed rather than immediately applied. Each endorser records the set of rows that would be read and written, noting also the exact version of those rows at that point in time. This "read-write set" of versioned rows is explicitly referenced in the endorsement and included in the transaction which the client broadcasts.

Conflicts between Fabric transactions are resolved once their order is finalized in the chain. Every node processes each transaction independently, checking endorsement policies and applying the database changes specified. However, if a transaction reads or writes a database row version that has already been modified by a previous transaction, then that second transaction is ignored. To go back to Alice's conflicting payments to Bob and Charlie, both of these transactions will read and modify the same row version, containing the €10 with which Alice started. So, the second transaction will be safely and automatically voided.

Fabric's approach to conflict resolution works, but in terms of performance and flexibility it combines the worst of the previous two models. Because endorsements convert transactions into specific read-write sets, Gavin and Helen's simultaneous but compatible €40 payments would lead to a conflict that Ethereum avoids. However, Fabric does not gain the speed advantage of the input–output model, since endorsers execute contracts against the most recent version of the database confirmed by the blockchain, ignoring unconfirmed transactions. So, if Helen initiates her payment a few seconds after Gavin, but before Gavin's has been confirmed on the blockchain, Fabric will create conflicting transactions that a pure input–output model avoids.

Conflict prevention

	Fabric	MultiChain	Ethereum	Corda
Model	Read-write sets	Single spend	Contract checks	Single spend
Verification	Independent	Independent	Independent	Trusted notaries
Speed	~10s (confirmation)	~1s (propagation)	~10s (confirmation)	0~5s (notary)

Legal Considerations and Risks

Market volatility risk – cryptocurrency markets, since their appearance, have been characterized by high degree of uncertainty and volatility. In addition, having in mind the relatively recent rise of this market, the unpredictability of the future economic conditions may have significant impact on the success of any such offerings in general;

Cyber security risk – the company may be a target of malicious attacks aimed to find and exploit weaknesses in the software, which may potentially result in loss and/or theft of tokens. In addition, there is an ongoing risk of system failures, that may result is service interruptions and/or other negative consequences;

Industry/competition risk – companies engaged in the high- technology business sphere face significant competition, due to the high-speed development of the technologies and amortization of the currently exploited systems, software etc.; Risks can be related to larger client base, name recognition, bad commercial practices and implementation of practices in violation of the common competition rules;

Media risk – as usual, development of technologies is related to increasing media interest, where the sector is tide by high level of confidentiality, commercial and trade secrets. In this case excessive media interest may seriously harm the business, as for example activating other risks as the above described "competition risk" or via negative publications to affect client base etc.;

Regulatory & compliance risk – because of the "cutting edge" status of the sector, regulations are following the practical implementation of cryptocurrency and blockchain technologies. The rapidly developing sector may suddenly become a subject to statutory and regulatory requirements, which may potentially alter the business;

Summary

As you have read each platform represents a complex multi-way trade-off between flexibility, simplicity, performance, disintermediation, safety and confidentiality.

BCB Coinstrike Limited chose to build our technology and all future applications based on Open Source block chain from MultiChain due to its flexibility, security, enhanced smart filer system and the ease to build applications based on the technology including all new planned releases and upgrades as presented here. MultiChain has been in development since the end of 2014, so almost 5 years. The first alpha release of MultiChain 1.0 was in June 2015, the second version 2.0 was released in March 2019.

Bycoi launched its MultiChain Blockchain in April 2019. Full open source code on GitHub you can find here.

Welcome to the world of Bycoi.

Duncan Malcolm Arthur
BCB Coinstrike Limited