

TABLE OF WhitePapers

001 Bitcoin

224 IOTA

010 Ethereum

252 Dash

042 Ripple

265 NEO

050 EOS

274 Maker

058 Tether

295 NEM

078 Tron

353 VeChain

118 Stellar

150 Cardano

204 Monero

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

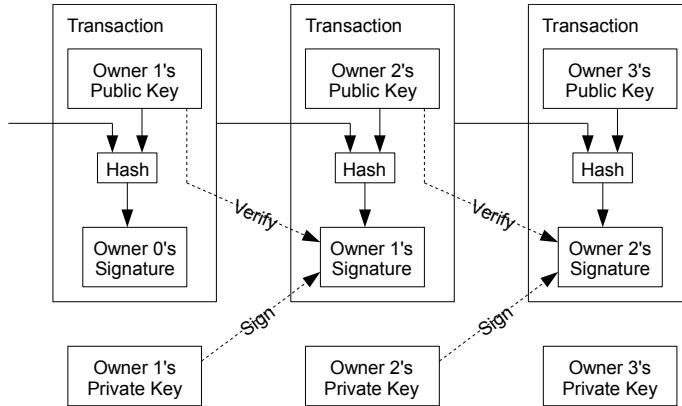
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

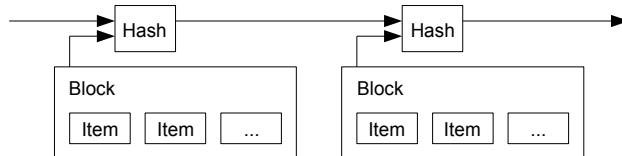


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

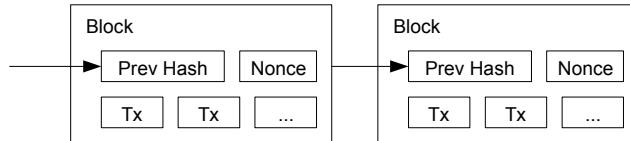
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

- 1) New transactions are broadcast to all nodes.
- 2) Each node collects new transactions into a block.
- 3) Each node works on finding a difficult proof-of-work for its block.
- 4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
- 5) Nodes accept the block only if all transactions in it are valid and not already spent.
- 6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

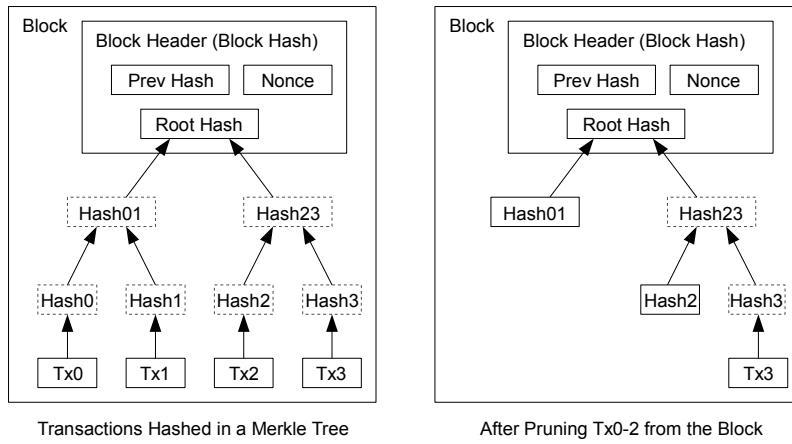
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

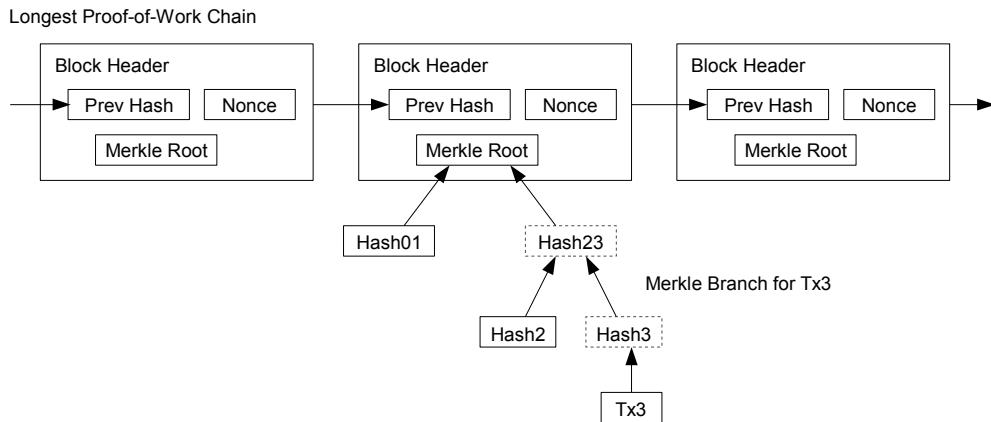
Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, $80 \text{ bytes} * 6 * 24 * 365 = 4.2\text{MB}$ per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

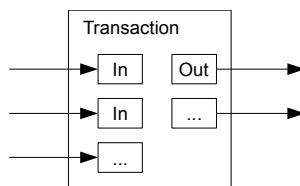
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

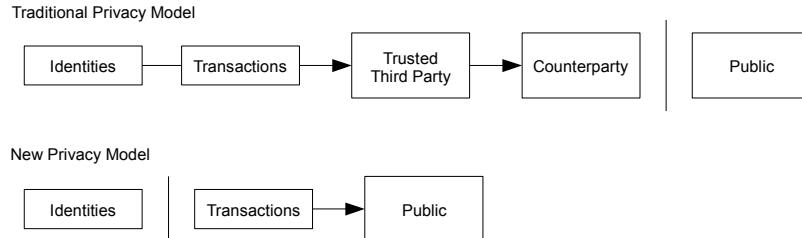
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block

q = probability the attacker finds the next block

q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z \frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \begin{cases} (q/p)^{(z-k)} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^z \frac{\lambda^k e^{-\lambda}}{k!} (1 - (q/p)^{(z-k)})$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0    P=1.0000000
z=1    P=0.2045873
z=2    P=0.0509779
z=3    P=0.0131722
z=4    P=0.0034552
z=5    P=0.0009137
z=6    P=0.0002428
z=7    P=0.0000647
z=8    P=0.0000173
z=9    P=0.0000046
z=10   P=0.0000012

q=0.3
z=0    P=1.0000000
z=5    P=0.1773523
z=10   P=0.0416605
z=15   P=0.0101008
z=20   P=0.0024804
z=25   P=0.0006132
z=30   P=0.0001522
z=35   P=0.0000379
z=40   P=0.0000095
z=45   P=0.0000024
z=50   P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10  z=5
q=0.15  z=8
q=0.20  z=11
q=0.25  z=15
q=0.30  z=24
q=0.35  z=41
q=0.40  z=89
q=0.45  z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

- [1] W. Dai, "b-money," <http://www.weidai.com/bmoney.txt>, 1998.
- [2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.
- [3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.
- [4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.
- [5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.
- [6] A. Back, "Hashcash - a denial of service counter-measure," <http://www.hashcash.org/papers/hashcash.pdf>, 2002.
- [7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.
- [8] W. Feller, "An introduction to probability theory and its applications," 1957.

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated, through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make a decentralised value-transfer system, shared across the world and virtually free to use. This system can be said to be a very specialised version of a cryptographically secure, transaction-based state machine. Follow-up systems such as Namecoin adapted this original “currency application” of the technology into other applications albeit rather simplistic ones.

Ethereum is a project which attempts to build the generalised technology; technology on which all transaction-based state machine concepts may be built. Moreover it aims to provide to the end-developer a tightly integrated end-to-end system for building software on a hitherto unexplored compute paradigm in the mainstream: a trustful object messaging compute framework.

1.1. Driving Factors. There are many goals of this project; one key goal is to facilitate transactions between consenting individuals who would otherwise have no means to trust one another. This may be due to geographical separation, interfacing difficulty, or perhaps the incompatibility, incompetence, unwillingness, expense, uncertainty, inconvenience or corruption of existing legal systems. By specifying a state-change system through a rich and unambiguous language, and furthermore architecting a system such that we can reasonably expect that an agreement will be thus enforced autonomously, we can provide a means to this end.

Dealings in this proposed system would have several attributes not often found in the real world. The incorruptibility of judgement, often difficult to find, comes naturally from a disinterested algorithmic interpreter. Transparency, or being able to see exactly how a state or judgement came about through the transaction log and rules or instructional codes, never happens perfectly in human-based systems since natural language is necessarily vague,

information is often lacking, and plain old prejudices are difficult to shake.

Overall, I wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

1.2. Previous Work. Buterin [2013a] first proposed the kernel of this work in late November, 2013. Though now evolved in many ways, the key functionality of a blockchain with a Turing-complete language and an effectively unlimited inter-transaction storage capability remains unchanged.

Dwork and Naor [1992] provided the first work into the usage of a cryptographic proof of computational expenditure (“proof-of-work”) as a means of transmitting a value signal over the Internet. The value-signal was utilised here as a spam deterrence mechanism rather than any kind of currency, but critically demonstrated the potential for a basic data channel to carry a *strong economic signal*, allowing a receiver to make a physical assertion without having to rely upon *trust*. Back [2002] later produced a system in a similar vein.

The first example of utilising the proof-of-work as a strong economic signal to secure a currency was by Vishnumurthy et al. [2003]. In this instance, the token was used to keep peer-to-peer file trading in check, ensuring “consumers” be able to make micro-payments to “suppliers” for their services. The security model afforded by the proof-of-work was augmented with digital signatures and a ledger in order to ensure that the historical record couldn’t be corrupted and that malicious actors could not spoof payment or unjustly complain about service delivery. Five years later, Nakamoto [2008] introduced another such proof-of-work-secured value token, somewhat wider in scope. The fruits of this project, Bitcoin, became the first widely adopted global decentralised transaction ledger.

Other projects built on Bitcoin’s success; the alt-coins introduced numerous other currencies through alteration to the protocol. Some of the best known are Litecoin and Primecoin, discussed by Sprankel [2013]. Other projects sought to take the core value content mechanism of the

protocol and repurpose it; Aron [2012] discusses, for example, the Namecoin project which aims to provide a decentralised name-resolution system.

Other projects still aim to build upon the Bitcoin network itself, leveraging the large amount of value placed in the system and the vast amount of computation that goes into the consensus mechanism. The Mastercoin project, first proposed by Willett [2013], aims to build a richer protocol involving many additional high-level features on top of the Bitcoin protocol through utilisation of a number of auxiliary parts to the core protocol. The Coloured Coins project, proposed by Rosenfeld [2012], takes a similar but more simplified strategy, embellishing the rules of a transaction in order to break the fungibility of Bitcoin’s base currency and allow the creation and tracking of tokens through a special “chroma-wallet”-protocol-aware piece of software.

Additional work has been done in the area with discarding the decentralisation foundation; Ripple, discussed by Boutellier and Heinzen [2014], has sought to create a “federated” system for currency exchange, effectively creating a new financial clearing system. It has demonstrated that high efficiency gains can be made if the decentralisation premise is discarded.

Early work on smart contracts has been done by Szabo [1997] and Miller [1997]. Around the 1990s it became clear that algorithmic enforcement of agreements could become a significant force in human cooperation. Though no specific system was proposed to implement such a system, it was proposed that the future of law would be heavily affected by such systems. In this light, Ethereum may be seen as a general implementation of such a *crypto-law* system.

2. THE BLOCKCHAIN PARADIGM

Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some final state. It is this final state which we accept as the canonical “version” of the world of Ethereum. The state can include such information as account balances, reputations, trust arrangements, data pertaining to information of the physical world; in short, anything that can currently be represented by a computer is admissible. Transactions thus represent a valid arc between two states; the ‘valid’ part is important—there exist far more invalid state changes than valid state changes. Invalid state changes might, e.g. be things such as reducing an account balance without an equal and opposite increase elsewhere. A valid state transition is one which comes about through a transaction. Formally:

$$(1) \quad \sigma_{t+1} \equiv \Upsilon(\sigma_t, T)$$

where Υ is the Ethereum state transition function. In Ethereum, Υ , together with σ are considerably more powerful than any existing comparable system; Υ allows components to carry out arbitrary computation, while σ allows components to store arbitrary state between transactions.

Transactions are collated into blocks; blocks are chained together using a cryptographic hash as a means of reference. Blocks function as a journal, recording a series of transactions together with the previous block and an

identifier for the final state (though do not store the final state itself—that would be far too big). They also punctuate the transaction series with incentives for nodes to *mine*. This incentivisation takes place as a state-transition function, adding value to a nominated account.

Mining is the process of dedicating effort (working) to bolster one series of transactions (a block) over any other potential competitor block. It is achieved thanks to a cryptographically secure proof. This scheme is known as a proof-of-work and is discussed in detail in section 11.5.

Formally, we expand to:

$$\begin{aligned} (2) \quad \sigma_{t+1} &\equiv \Pi(\sigma_t, B) \\ (3) \quad B &\equiv (\dots, (T_0, T_1, \dots)) \\ (4) \quad \Pi(\sigma, B) &\equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1) \dots) \end{aligned}$$

Where Ω is the block-finalisation state transition function (a function that rewards a nominated party); B is this block, which includes a series of transactions amongst some other components; and Π is the block-level state-transition function.

This is the basis of the blockchain paradigm, a model that forms the backbone of not only Ethereum, but all decentralised consensus-based transaction systems to date.

2.1. Value. In order to incentivise computation within the network, there needs to be an agreed method for transmitting value. To address this issue, Ethereum has an intrinsic currency, Ether, known also as ETH and sometimes referred to by the Old English *D*. The smallest subdenomination of Ether, and thus the one in which all integer values of the currency are counted, is the Wei. One Ether is defined as being 10^{18} Wei. There exist other subdenominations of Ether:

Multiplier	Name
10^0	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Throughout the present work, any reference to value, in the context of Ether, currency, a balance or a payment, should be assumed to be counted in Wei.

2.2. Which History? Since the system is decentralised and all parties have an opportunity to create a new block on some older pre-existing block, the resultant structure is necessarily a tree of blocks. In order to form a consensus as to which path, from root (the genesis block) to leaf (the block containing the most recent transactions) through this tree structure, known as the blockchain, there must be an agreed-upon scheme. If there is ever a disagreement between nodes as to which root-to-leaf path down the block tree is the ‘best’ blockchain, then a *fork* occurs.

This would mean that past a given point in time (block), multiple states of the system may coexist: some nodes believing one block to contain the canonical transactions, other nodes believing some other block to be canonical, potentially containing radically different or incompatible transactions. This is to be avoided at all costs as the uncertainty that would ensue would likely kill all confidence in the entire system.

The scheme we use in order to generate consensus is a simplified version of the GHOST protocol introduced by

Sompsonsky and Zohar [2013]. This process is described in detail in section 10.

3. CONVENTIONS

I use a number of typographical conventions for the formal notation, some of which are quite particular to the present work:

The two sets of highly structured, ‘top-level’, state values, are denoted with bold lowercase Greek letters. They fall into those of world-state, which are denoted σ (or a variant thereupon) and those of machine-state, μ .

Functions operating on highly structured values are denoted with an upper-case greek letter, e.g. Υ , the Ethereum state transition function.

For most functions, an uppercase letter is used, e.g. C , the general cost function. These may be subscripted to denote specialised variants, e.g. C_{STOKE} , the cost function for the `STOKE` operation. For specialised and possibly externally defined functions, I may format as typewriter text, e.g. the Keccak-256 hash function (as per the winning entry to the SHA-3 contest) is denoted `KEC` (and generally referred to as plain Keccak). Also `KEC512` is referring to the Keccak 512 hash function.

Tuples are typically denoted with an upper-case letter, e.g. T , is used to denote an Ethereum transaction. This symbol may, if accordingly defined, be subscripted to refer to an individual component, e.g. T_n , denotes the nonce of said transaction. The form of the subscript is used to denote its type; e.g. uppercase subscripts refer to tuples with subscriptable components.

Scalars and fixed-size byte sequences (or, synonymously, arrays) are denoted with a normal lower-case letter, e.g. n is used in the document to denote a transaction nonce. Those with a particularly special meaning may be greek, e.g. δ , the number of items required on the stack for a given operation.

Arbitrary-length sequences are typically denoted as a bold lower-case letter, e.g. \mathbf{o} is used to denote the byte-sequence given as the output data of a message call. For particularly important values, a bold uppercase letter may be used.

Throughout, we assume scalars are positive integers and thus belong to the set \mathbb{P} . The set of all byte sequences is \mathbb{B} , formally defined in Appendix B. If such a set of sequences is restricted to those of a particular length, it is denoted with a subscript, thus the set of all byte sequences of length 32 is named \mathbb{B}_{32} and the set of all positive integers smaller than 2^{256} is named \mathbb{P}_{256} . This is formally defined in section 4.4.

Square brackets are used to index into and reference individual components or subsequences of sequences, e.g. $\mu_s[0]$ denotes the first item on the machine’s stack. For subsequences, ellipses are used to specify the intended range, to include elements at both limits, e.g. $\mu_m[0..31]$ denotes the first 32 items of the machine’s memory.

In the case of the global state σ , which is a sequence of accounts, themselves tuples, the square brackets are used to reference an individual account.

When considering variants of existing values, I follow the rule that within a given scope for definition, if we assume that the unmodified ‘input’ value be denoted by the placeholder \square then the modified and utilisable value is denoted as \square' , and intermediate values would be \square^* ,

\square^{**} &c. On very particular occasions, in order to maximise readability and only if unambiguous in meaning, I may use alpha-numeric subscripts to denote intermediate values, especially those of particular note.

When considering the use of existing functions, given a function f , the function f^* denotes a similar, element-wise version of the function mapping instead between sequences. It is formally defined in section 4.4.

I define a number of useful functions throughout. One of the more common is ℓ , which evaluates to the last item in the given sequence:

$$(5) \quad \ell(\mathbf{x}) \equiv \mathbf{x}[\|\mathbf{x}\| - 1]$$

4. BLOCKS, STATE AND TRANSACTIONS

Having introduced the basic concepts behind Ethereum, we will discuss the meaning of a transaction, a block and the state in more detail.

4.1. World State. The world state (*state*), is a mapping between addresses (160-bit identifiers) and account states (a data structure serialised as RLP, see Appendix B). Though not stored on the blockchain, it is assumed that the implementation will maintain this mapping in a modified Merkle Patricia tree (*trie*, see Appendix D). The trie requires a simple database backend that maintains a mapping of bytearrays to bytearrays; we name this underlying database the state database. This has a number of benefits; firstly the root node of this structure is cryptographically dependent on all internal data and as such its hash can be used as a secure identity for the entire system state. Secondly, being an immutable data structure, it allows any previous state (whose root hash is known) to be recalled by simply altering the root hash accordingly. Since we store all such root hashes in the blockchain, we are able to trivially revert to old states.

The account state comprises the following four fields:

nonce: A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account. For account of address a in state σ , this would be formally denoted $\sigma[a]_n$.

balance: A scalar value equal to the number of Wei owned by this address. Formally denoted $\sigma[a]_b$.

storageRoot: A 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values. The hash is formally denoted $\sigma[a]_s$.

codeHash: The hash of the EVM code of this account—this is the code that gets executed should this address receive a message call; it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their corresponding hashes for later retrieval. This hash is formally denoted $\sigma[a]_c$, and thus the code may be denoted as \mathbf{b} , given that $\text{KEC}(\mathbf{b}) = \sigma[a]_c$.

Since I typically wish to refer not to the trie's root hash but to the underlying set of key/value pairs stored within, I define a convenient equivalence:

$$(6) \quad \text{TRIE}(L_I^*(\sigma[a]_s)) \equiv \sigma[a]_s$$

The collapse function for the set of key/value pairs in the trie, L_I^* , is defined as the element-wise transformation of the base function L_I , given as:

$$(7) \quad L_I((k, v)) \equiv (\text{KEC}(k), \text{RLP}(v))$$

where:

$$(8) \quad k \in \mathbb{B}_{32} \quad \wedge \quad v \in \mathbb{P}$$

It shall be understood that $\sigma[a]_s$ is not a 'physical' member of the account and does not contribute to its later serialisation.

If the **codeHash** field is the Keccak-256 hash of the empty string, i.e. $\sigma[a]_c = \text{KEC}(\emptyset)$, then the node represents a simple account, sometimes referred to as a "non-contract" account.

Thus we may define a world-state collapse function L_S :

$$(9) \quad L_S(\sigma) \equiv \{p(a) : \sigma[a] \neq \emptyset\}$$

where

$$(10) \quad p(a) \equiv (\text{KEC}(a), \text{RLP}((\sigma[a]_n, \sigma[a]_b, \sigma[a]_s, \sigma[a]_c)))$$

This function, L_S , is used alongside the trie function to provide a short identity (hash) of the world state. We assume:

$$(11) \quad \forall a : \sigma[a] = \emptyset \vee (a \in \mathbb{B}_{20} \wedge v(\sigma[a]))$$

where v is the account validity function:

$$(12) \quad v(x) \equiv x_n \in \mathbb{P}_{256} \wedge x_b \in \mathbb{P}_{256} \wedge x_s \in \mathbb{B}_{32} \wedge x_c \in \mathbb{B}_{32}$$

4.2. Homestead. A significant block number for compatibility with the public network is the block marking the transition between the *Frontier* and *Homestead* phases of the platform, which we denote with the symbol N_H , defined thus

$$(13) \quad N_H \equiv 1,150,000$$

The protocol was upgraded at this block, so this symbol appears in some equations to account for the changes.

4.3. The Transaction. A transaction (formally, T) is a single cryptographically-signed instruction constructed by an actor externally to the scope of Ethereum. While it is assumed that the ultimate external actor will be human in nature, software tools will be used in its construction and dissemination¹. There are two types of transactions: those which result in message calls and those which result in the creation of new accounts with associated code (known informally as 'contract creation'). Both types specify a number of common fields:

nonce: A scalar value equal to the number of transactions sent by the sender; formally T_n .

gasPrice: A scalar value equal to the number of Wei to be paid per unit of *gas* for all computation costs incurred as a result of the execution of this transaction; formally T_p .

¹Notably, such 'tools' could ultimately become so causally removed from their human-based initiation—or humans may become so causally-neutral—that there could be a point at which they rightly be considered autonomous agents. e.g. contracts may offer bounties to humans for being sent transactions to initiate their execution.

gasLimit: A scalar value equal to the maximum amount of gas that should be used in executing this transaction. This is paid up-front, before any computation is done and may not be increased later; formally T_g .

to: The 160-bit address of the message call's recipient or, for a contract creation transaction, \emptyset , used here to denote the only member of \mathbb{B}_0 ; formally T_t .

value: A scalar value equal to the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account; formally T_v .

v, r, s: Values corresponding to the signature of the transaction and used to determine the sender of the transaction; formally T_w , T_r and T_s . This is expanded in Appendix F.

Additionally, a contract creation transaction contains:

init: An unlimited size byte array specifying the EVM-code for the account initialisation procedure, formally T_i .

init is an EVM-code fragment; it returns the **body**, a second fragment of code that executes each time the account receives a message call (either through a transaction or due to the internal execution of code). **init** is executed only once at account creation and gets discarded immediately thereafter.

In contrast, a message call transaction contains:

data: An unlimited size byte array specifying the input data of the message call, formally T_d .

Appendix F specifies the function, S , which maps transactions to the sender, and happens through the ECDSA of the SECP-256k1 curve, using the hash of the transaction (excluding the latter three signature fields) as the datum to sign. For the present we simply assert that the sender of a given transaction T can be represented with $S(T)$.

$$(14)$$

$$L_T(T) \equiv \begin{cases} (T_n, T_p, T_g, T_t, T_v, T_i, T_w, T_r, T_s) & \text{if } T_t = \emptyset \\ (T_n, T_p, T_g, T_t, T_v, T_d, T_w, T_r, T_s) & \text{otherwise} \end{cases}$$

Here, we assume all components are interpreted by the RLP as integer values, with the exception of the arbitrary length byte arrays T_i and T_d .

$$(15) \quad \begin{aligned} T_n &\in \mathbb{P}_{256} \quad \wedge \quad T_v \in \mathbb{P}_{256} \quad \wedge \quad T_p \in \mathbb{P}_{256} \quad \wedge \\ T_g &\in \mathbb{P}_{256} \quad \wedge \quad T_w \in \mathbb{P}_5 \quad \wedge \quad T_r \in \mathbb{P}_{256} \quad \wedge \\ T_s &\in \mathbb{P}_{256} \quad \wedge \quad T_d \in \mathbb{B} \quad \wedge \quad T_i \in \mathbb{B} \end{aligned}$$

where

$$(16) \quad \mathbb{P}_n = \{P : P \in \mathbb{P} \wedge P < 2^n\}$$

The address hash T_t is slightly different: it is either a 20-byte address hash or, in the case of being a contract-creation transaction (and thus formally equal to \emptyset), it is the RLP empty byte-series and thus the member of \mathbb{B}_0 :

$$(17) \quad T_t \in \begin{cases} \mathbb{B}_{20} & \text{if } T_t \neq \emptyset \\ \mathbb{B}_0 & \text{otherwise} \end{cases}$$

4.4. The Block. The block in Ethereum is the collection of relevant pieces of information (known as the block *header*), H , together with information corresponding to the comprised transactions, \mathbf{T} , and a set of other block headers \mathbf{U} that are known to have a parent equal to the present block's parent's parent (such blocks are known as *ommers*²). The block header contains several pieces of information:

- parentHash:** The Keccak 256-bit hash of the parent block's header, in its entirety; formally H_p .
- ommersHash:** The Keccak 256-bit hash of the ommer list portion of this block; formally H_o .
- beneficiary:** The 160-bit address to which all fees collected from the successful mining of this block be transferred; formally H_c .
- stateRoot:** The Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied; formally H_r .
- transactionsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with each transaction in the transactions list portion of the block; formally H_t .
- receiptsRoot:** The Keccak 256-bit hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list portion of the block; formally H_e .
- logsBloom:** The Bloom filter composed from indexable information (logger address and log topics) contained in each log entry from the receipt of each transaction in the transactions list; formally H_b .
- difficulty:** A scalar value corresponding to the difficulty level of this block. This can be calculated from the previous block's difficulty level and the timestamp; formally H_d .
- number:** A scalar value equal to the number of ancestor blocks. The genesis block has a number of zero; formally H_i .
- gasLimit:** A scalar value equal to the current limit of gas expenditure per block; formally H_l .
- gasUsed:** A scalar value equal to the total gas used in transactions in this block; formally H_g .
- timestamp:** A scalar value equal to the reasonable output of Unix's time() at this block's inception; formally H_s .
- extraData:** An arbitrary byte array containing data relevant to this block. This must be 32 bytes or fewer; formally H_x .
- mixHash:** A 256-bit hash which proves combined with the nonce that a sufficient amount of computation has been carried out on this block; formally H_m .
- nonce:** A 64-bit hash which proves combined with the mix-hash that a sufficient amount of computation has been carried out on this block; formally H_n .

The other two components in the block are simply a list of ommer block headers (of the same format as above) and a series of the transactions. Formally, we can refer to

²*ommer* is the most prevalent (not saying much) gender-neutral term to mean “sibling of parent”; see http://nonbinary.org/wiki/Gender_neutral_language#Family_Terms

a block B :

$$(18) \quad B \equiv (B_H, B_{\mathbf{T}}, B_{\mathbf{U}})$$

4.4.1. Transaction Receipt. In order to encode information about a transaction concerning which it may be useful to form a zero-knowledge proof, or index and search, we encode a receipt of each transaction containing certain information from concerning its execution. Each receipt, denoted $B_{\mathbf{R}}[i]$ for the i th transaction) is placed in an index-keyed trie and the root recorded in the header as H_e .

The transaction receipt is a tuple of four items comprising the post-transaction state, R_{σ} , the cumulative gas used in the block containing the transaction receipt as of immediately after the transaction has happened, R_u , the set of logs created through execution of the transaction, R_l and the Bloom filter composed from information in those logs, R_b :

$$(19) \quad R \equiv (R_{\sigma}, R_u, R_b, R_l)$$

The function L_R trivially prepares a transaction receipt for being transformed into an RLP-serialised byte array:

$$(20) \quad L_R(R) \equiv (\text{TRIE}(L_S(R_{\sigma})), R_u, R_b, R_l)$$

thus the post-transaction state, R_{σ} is encoded into a trie structure, the root of which forms the first item.

We assert R_u , the cumulative gas used is a positive integer and that the logs Bloom, R_b , is a hash of size 2048 bits (256 bytes):

$$(21) \quad R_u \in \mathbb{P} \quad \wedge \quad R_b \in \mathbb{B}_{256}$$

The log entries, R_l , is a series of log entries, termed, for example, (O_0, O_1, \dots) . A log entry, O , is a tuple of a logger's address, O_a , a series of 32-bytes log topics, O_t and some number of bytes of data, O_d :

$$(22) \quad O \equiv (O_a, (O_{t0}, O_{t1}, \dots), O_d)$$

$$(23) \quad O_a \in \mathbb{B}_{20} \quad \wedge \quad \forall_{t \in O_t} : t \in \mathbb{B}_{32} \quad \wedge \quad O_d \in \mathbb{B}$$

We define the Bloom filter function, M , to reduce a log entry into a single 256-byte hash:

$$(24) \quad M(O) \equiv \bigvee_{t \in \{O_a\} \cup O_t} (M_{3:2048}(t))$$

where $M_{3:2048}$ is a specialised Bloom filter that sets three bits out of 2048, given an arbitrary byte series. It does this through taking the low-order 11 bits of each of the first three pairs of bytes in a Keccak-256 hash of the byte series. Formally:

$$(25) M_{3:2048}(\mathbf{x} : \mathbf{x} \in \mathbb{B}) \equiv \mathbf{y} : \mathbf{y} \in \mathbb{B}_{256} \quad \text{where:}$$

$$(26) \quad \mathbf{y} = (0, 0, \dots, 0) \quad \text{except:}$$

$$(27) \quad \forall_{i \in \{0, 2, 4\}} : \mathcal{B}_{m(\mathbf{x}, i)}(\mathbf{y}) = 1$$

$$(28) \quad m(\mathbf{x}, i) \equiv \text{KEC}(\mathbf{x})[i, i + 1] \bmod 2048$$

where \mathcal{B} is the bit reference function such that $\mathcal{B}_j(\mathbf{x})$ equals the bit of index j (indexed from 0) in the byte array \mathbf{x} .

4.4.2. Holistic Validity. We can assert a block's validity if and only if it satisfies several conditions: it must be internally consistent with the ommer and transaction block hashes and the given transactions B_T (as specified in sec 11), when executed in order on the base state σ (derived from the final state of the parent block), result in a new state of the identity H_r :

$$(29) \quad \begin{aligned} H_r &\equiv \text{TRIE}(L_S(\Pi(\sigma, B))) & \wedge \\ H_o &\equiv \text{KEC}(\text{RLP}(L_H^*(B_U))) & \wedge \\ H_t &\equiv \text{TRIE}(\{\forall i < \|B_T\|, i \in \mathbb{P} : p(i, L_T(B_T[i]))\}) & \wedge \\ H_e &\equiv \text{TRIE}(\{\forall i < \|B_R\|, i \in \mathbb{P} : p(i, L_R(B_R[i]))\}) & \wedge \\ H_b &\equiv \bigvee_{r \in B_R} (r_b) \end{aligned}$$

where $p(k, v)$ is simply the pairwise RLP transformation, in this case, the first being the index of the transaction in the block and the second being the transaction receipt:

$$(30) \quad p(k, v) \equiv (\text{RLP}(k), \text{RLP}(v))$$

Furthermore:

$$(31) \quad \text{TRIE}(L_S(\sigma)) = P(B_H)_{H_r}$$

Thus $\text{TRIE}(L_S(\sigma))$ is the root node hash of the Merkle Patricia tree structure containing the key-value pairs of the state σ with values encoded using RLP, and $P(B_H)$ is the parent block of B , defined directly.

The values stemming from the computation of transactions, specifically the transaction receipts, B_R , and that defined through the transactions state-accumulation function, Π , are formalised later in section 11.4.

4.4.3. Serialisation. The function L_B and L_H are the preparation functions for a block and block header respectively. Much like the transaction receipt preparation function L_R , we assert the types and order of the structure for when the RLP transformation is required:

$$(32) \quad L_H(H) \equiv (H_p, H_o, H_c, H_r, H_t, H_e, H_b, H_d, H_i, H_l, H_g, H_s, H_x, H_m, H_n)$$

$$(33) \quad L_B(B) \equiv (L_H(B_H), L_T^*(B_T), L_H^*(B_U))$$

With L_T^* and L_H^* being element-wise sequence transformations, thus:

$$(34) \quad f^*((x_0, x_1, \dots)) \equiv (f(x_0), f(x_1), \dots) \quad \text{for any function } f$$

The component types are defined thus:

$$(35) \quad \begin{aligned} H_p &\in \mathbb{B}_{32} & \wedge & H_o \in \mathbb{B}_{32} & \wedge & H_c \in \mathbb{B}_{20} & \wedge \\ H_r &\in \mathbb{B}_{32} & \wedge & H_t \in \mathbb{B}_{32} & \wedge & H_e \in \mathbb{B}_{32} & \wedge \\ H_b &\in \mathbb{B}_{256} & \wedge & H_d \in \mathbb{P} & \wedge & H_i \in \mathbb{P} & \wedge \\ H_l &\in \mathbb{P} & \wedge & H_g \in \mathbb{P} & \wedge & H_s \in \mathbb{P}_{256} & \wedge \\ H_x &\in \mathbb{B} & \wedge & H_m \in \mathbb{B}_{32} & \wedge & H_n \in \mathbb{B}_8 & \end{aligned}$$

where

$$(36) \quad \mathbb{B}_n = \{B : B \in \mathbb{B} \wedge \|B\| = n\}$$

We now have a rigorous specification for the construction of a formal block structure. The RLP function RLP (see Appendix B) provides the canonical method for transforming this structure into a sequence of bytes ready for transmission over the wire or storage locally.

4.4.4. Block Header Validity. We define $P(B_H)$ to be the parent block of B , formally:

$$(37) \quad P(H) \equiv B' : \text{KEC}(\text{RLP}(B'_H)) = H_p$$

The block number is the parent's block number incremented by one:

$$(38) \quad H_i \equiv P(H)_{H_i} + 1$$

The canonical difficulty of a block of header H is defined as $D(H)$:

$$(39) \quad D(H) \equiv \begin{cases} D_0 & \text{if } H_i = 0 \\ \max(D_0, P(H)_{H_d} + x \times \varsigma_1 + \epsilon) & \text{if } H_i < N_H \\ \max(D_0, P(H)_{H_d} + x \times \varsigma_2 + \epsilon) & \text{otherwise} \end{cases}$$

where:

$$(40) \quad D_0 \equiv 131072$$

$$(41) \quad x \equiv \left\lfloor \frac{P(H)_{H_d}}{2048} \right\rfloor$$

$$(42) \quad \varsigma_1 \equiv \begin{cases} 1 & \text{if } H_s < P(H)_{H_s} + 13 \\ -1 & \text{otherwise} \end{cases}$$

$$(43) \quad \varsigma_2 \equiv \max \left(1 - \left\lfloor \frac{H_s - P(H)_{H_s}}{10} \right\rfloor, -99 \right)$$

$$(44) \quad \epsilon \equiv \left\lfloor 2^{\lfloor H_i \div 100000 \rfloor - 2} \right\rfloor$$

The canonical gas limit H_l of a block of header H must fulfil the relation:

$$(45) \quad H_l < P(H)_{H_l} + \left\lfloor \frac{P(H)_{H_l}}{1024} \right\rfloor \quad \wedge$$

$$(46) \quad H_l > P(H)_{H_l} - \left\lfloor \frac{P(H)_{H_l}}{1024} \right\rfloor \quad \wedge$$

$$(47) \quad H_l \geq 125000$$

H_s is the timestamp of block H and must fulfil the relation:

$$(48) \quad H_s > P(H)_{H_s}$$

This mechanism enforces a homeostasis in terms of the time between blocks; a smaller period between the last two blocks results in an increase in the difficulty level and thus additional computation required, lengthening the likely next period. Conversely, if the period is too large, the difficulty, and expected time to the next block, is reduced.

The nonce, H_n , must satisfy the relations:

$$(49) \quad n \leq \frac{2^{256}}{H_d} \quad \wedge \quad m = H_m$$

with $(n, m) = \text{PoW}(H_H, H_n, \mathbf{d})$.

Where H_H is the new block's header H , but *without* the nonce and mix-hash components, \mathbf{d} being the current DAG, a large data set needed to compute the mix-hash, and PoW is the proof-of-work function (see section 11.5): this evaluates to an array with the first item being the mix-hash, to proof that a correct DAG has been used, and the second item being a pseudo-random number cryptographically dependent on H and \mathbf{d} . Given an approximately uniform distribution in the range $[0, 2^{64}]$, the expected time to find a solution is proportional to the difficulty, H_d .

This is the foundation of the security of the blockchain and is the fundamental reason why a malicious node cannot propagate newly created blocks that would otherwise overwrite (“rewrite”) history. Because the nonce must satisfy this requirement, and because its satisfaction depends on the contents of the block and in turn its composed transactions, creating new, valid, blocks is difficult and, over time, requires approximately the total compute power of the trustworthy portion of the mining peers.

Thus we are able to define the block header validity function $V(H)$:

$$\begin{aligned}
 (50) \quad V(H) &\equiv n \leq \frac{2^{256}}{H_d} \wedge m = H_m \wedge \\
 (51) \quad H_d &= D(H) \wedge \\
 (52) \quad H_g &\leq H_l \wedge \\
 (53) \quad H_l &< P(H)_{H_l} + \left\lfloor \frac{P(H)_{H_l}}{1024} \right\rfloor \wedge \\
 (54) \quad H_l &> P(H)_{H_l} - \left\lfloor \frac{P(H)_{H_l}}{1024} \right\rfloor \wedge \\
 (55) \quad H_l &\geq 125000 \wedge \\
 (56) \quad H_s &> P(H)_{H_s} \wedge \\
 (57) \quad H_i &= P(H)_{H_i} + 1 \wedge \\
 (58) \quad \|H_x\| &\leq 32
 \end{aligned}$$

where $(n, m) = \text{PoW}(H_H, H_n, \mathbf{d})$

Noting additionally that **extraData** must be at most 32 bytes.

5. GAS AND PAYMENT

In order to avoid issues of network abuse and to sidestep the inevitable questions stemming from Turing completeness, all programmable computation in Ethereum is subject to fees. The fee schedule is specified in units of **gas** (see Appendix G for the fees associated with various computation). Thus any given fragment of programmable computation (this includes creating contracts, making message calls, utilising and accessing account storage and executing operations on the virtual machine) has a universally agreed cost in terms of gas.

Every transaction has a specific amount of gas associated with it: **gasLimit**. This is the amount of gas which is implicitly purchased from the sender’s account balance. The purchase happens at the according **gasPrice**, also specified in the transaction. The transaction is considered invalid if the account balance cannot support such a purchase. It is named **gasLimit** since any unused gas at the end of the transaction is refunded (at the same rate of purchase) to the sender’s account. Gas does not exist outside of the execution of a transaction. Thus for accounts with trusted code associated, a relatively high gas limit may be set and left alone.

In general, Ether used to purchase gas that is not refunded is delivered to the *beneficiary* address, the address of an account typically under the control of the miner. Transactors are free to specify any **gasPrice** that they wish, however miners are free to ignore transactions as they choose. A higher gas price on a transaction will therefore cost the sender more in terms of Ether and deliver a greater value to the miner and thus will more likely be selected for inclusion by more miners. Miners, in general, will choose to advertise the minimum gas price for which

they will execute transactions and transactors will be free to canvas these prices in determining what gas price to offer. Since there will be a (weighted) distribution of minimum acceptable gas prices, transactors will necessarily have a trade-off to make between lowering the gas price and maximising the chance that their transaction will be mined in a timely manner.

6. TRANSACTION EXECUTION

The execution of a transaction is the most complex part of the Ethereum protocol: it defines the state transition function Υ . It is assumed that any transactions executed first pass the initial tests of intrinsic validity. These include:

- (1) The transaction is well-formed RLP, with no additional trailing bytes;
- (2) the transaction signature is valid;
- (3) the transaction nonce is valid (equivalent to the sender account’s current nonce);
- (4) the gas limit is no smaller than the intrinsic gas, g_0 , used by the transaction;
- (5) the sender account balance contains at least the cost, v_0 , required in up-front payment.

Formally, we consider the function Υ , with T being a transaction and σ the state:

$$(59) \quad \sigma' = \Upsilon(\sigma, T)$$

Thus σ' is the post-transactional state. We also define Υ^g to evaluate to the amount of gas used in the execution of a transaction and Υ^l to evaluate to the transaction’s accrued log items, both to be formally defined later.

6.1. Substate. Throughout transaction execution, we accrue certain information that is acted upon immediately following the transaction. We call this *transaction substate*, and represent it as A , which is a tuple:

$$(60) \quad A \equiv (A_s, A_l, A_r)$$

The tuple contents include A_s , the suicide set: a set of accounts that will be discarded following the transaction’s completion. A_l is the log series: this is a series of archived and indexable ‘checkpoints’ in VM code execution that allow for contract-calls to be easily tracked by onlookers external to the Ethereum world (such as decentralised application front-ends). Finally there is A_r , the refund balance, increased through using the SSTORE instruction in order to reset contract storage to zero from some non-zero value. Though not immediately refunded, it is allowed to partially offset the total execution costs.

For brevity, we define the empty substate A^0 to have no suicides, no logs and a zero refund balance:

$$(61) \quad A^0 \equiv (\emptyset, (), 0)$$

6.2. Execution. We define intrinsic gas g_0 , the amount of gas this transaction requires to be paid prior to execution, as follows:

$$(62) \quad g_0 \equiv \sum_{i \in T_l, T_d} \begin{cases} G_{txdatazero} & \text{if } i = 0 \\ G_{txdatanonzero} & \text{otherwise} \end{cases}$$

$$(63) \quad + \begin{cases} G_{txcreate} & \text{if } T_t = \emptyset \wedge H_i \geq N_H \\ 0 & \text{otherwise} \end{cases}$$

$$(64) \quad + G_{transaction}$$

where T_i, T_d means the series of bytes of the transaction's associated data and initialisation EVM-code, depending on whether the transaction is for contract-creation or message-call. $G_{txcreate}$ is added if the transaction is contract-creating, but not if a result of EVM-code or before the *Homestead transition*. G is fully defined in Appendix G.

The up-front cost v_0 is calculated as:

$$(65) \quad v_0 \equiv T_g T_p + T$$

The validity is determined as:

$$(66) \quad \begin{aligned} S(T) &\neq \emptyset \wedge \\ \sigma[S(T)] &\neq \emptyset \wedge \\ T_n &= \sigma[S(T)]_n \wedge \\ g_0 &\leqslant T_g \wedge \\ v_0 &\leqslant \sigma[S(T)]_b \wedge \\ T_g &\leqslant B_{Hl} - \ell(B_R)_u \end{aligned}$$

Note the final condition; the sum of the transaction's gas limit, T_g , and the gas utilised in this block prior, given by $\ell(B_R)_u$, must be no greater than the block's **gasLimit**, B_{Hl} .

The execution of a valid transaction begins with an irrevocable change made to the state: the nonce of the account of the sender, $S(T)$, is incremented by one and the balance is reduced by part of the up-front cost, $T_g T_p$. The gas available for the proceeding computation, g , is defined as $T_g - g_0$. The computation, whether contract creation or a message call, results in an eventual state (which may legally be equivalent to the current state), the change to which is deterministic and never invalid: there can be no invalid transactions from this point.

We define the checkpoint state σ_0 :

$$(67) \quad \sigma_0 \equiv \sigma \text{ except:}$$

$$(68) \quad \sigma_0[S(T)]_b \equiv \sigma[S(T)]_b - T_g T_p$$

$$(69) \quad \sigma_0[S(T)]_n \equiv \sigma[S(T)]_n + 1$$

Evaluating σ_P from σ_0 depends on the transaction type; either contract creation or message call; we define the tuple of post-execution provisional state σ_P , remaining gas g' and substate A :

$$(70) \quad (\sigma_P, g', A) \equiv \begin{cases} \Lambda(\sigma_0, S(T), T_o, \\ \quad g, T_p, T_v, T_i, 0) & \text{if } T_t = \emptyset \\ \Theta_3(\sigma_0, S(T), T_o, \\ \quad T_t, T, g, T_p, T_v, T_v, T_d, 0) & \text{otherwise} \end{cases}$$

where g is the amount of gas remaining after deducting the basic amount required to pay for the existence of the transaction:

$$(71) \quad g \equiv T_g - g_0$$

and T_o is the original transactor, which can differ from the sender in the case of a message call or contract creation not directly triggered by a transaction but coming from the execution of EVM-code.

Note we use Θ_3 to denote the fact that only the first three components of the function's value are taken; the final represents the message-call's output value (a byte array) and is unused in the context of transaction evaluation.

After the message call or contract creation is processed, the state is finalised by determining the amount to be refunded, g^* from the remaining gas, g' , plus some allowance

from the refund counter, to the sender at the original rate.

$$(72) \quad g^* \equiv g' + \min\left\{\left\lfloor \frac{T_g - g'}{2} \right\rfloor, A_r \right\}$$

The total refundable amount is the legitimately remaining gas g' , added to A_r , with the latter component being capped up to a maximum of half (rounded down) of the total amount used $T_g - g'$.

The Ether for the gas is given to the miner, whose address is specified as the beneficiary of the present block B . So we define the pre-final state σ^* in terms of the provisional state σ_P :

$$(73) \quad \sigma^* \equiv \sigma_P \text{ except}$$

$$(74) \quad \sigma^*[S(T)]_b \equiv \sigma_P[S(T)]_b + g^* T_p$$

$$(75) \quad \sigma^*[m]_b \equiv \sigma_P[m]_b + (T_g - g^*) T_p$$

$$(76) \quad m \equiv B_{Hc}$$

The final state, σ' , is reached after deleting all accounts that appear in the suicide list:

$$(77) \quad \sigma' \equiv \sigma^* \text{ except}$$

$$(78) \quad \forall i \in A_s : \sigma'[i] \equiv \emptyset$$

And finally, we specify Υ^g , the total gas used in this transaction and Υ^1 , the logs created by this transaction:

$$(79) \quad \Upsilon^g(\sigma, T) \equiv T_g - g'$$

$$(80) \quad \Upsilon^1(\sigma, T) \equiv A_1$$

These are used to help define the transaction receipt, discussed later.

7. CONTRACT CREATION

There are a number of intrinsic parameters used when creating an account: sender (s), original transactor (o), available gas (g), gas price (p), endowment (v) together with an arbitrary length byte array, i , the initialisation EVM code and finally the present depth of the message-call/contract-creation stack (e).

We define the creation function formally as the function Λ , which evaluates from these values, together with the state σ to the tuple containing the new state, remaining gas and accrued transaction substate (σ', g', A) , as in section 6:

$$(81) \quad (\sigma', g', A) \equiv \Lambda(\sigma, s, o, g, p, v, i, e)$$

The address of the new account is defined as being the rightmost 160 bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the nonce. Thus we define the resultant address for the new account a :

$$(82) \quad a \equiv \mathcal{B}_{96..255}\left(\text{KEC}\left(\text{RLP}\left((s, \sigma[s]_n - 1)\right)\right)\right)$$

where KEC is the Keccak 256-bit hash function, RLP is the RLP encoding function, $\mathcal{B}_{a..b}(X)$ evaluates to binary value containing the bits of indices in the range $[a, b]$ of the binary data X and $\sigma[x]$ is the address state of x or \emptyset if none exists. Note we use one fewer than the sender's nonce value; we assert that we have incremented the sender account's nonce prior to this call, and so the value used is the sender's nonce at the beginning of the responsible transaction or VM operation.

The account's nonce is initially defined as zero, the balance as the value passed, the storage as empty and the code hash as the Keccak 256-bit hash of the empty string;

the sender's balance is also reduced by the value passed. Thus the mutated state becomes σ^* :

$$(83) \quad \sigma^* \equiv \sigma \text{ except:}$$

$$(84) \quad \sigma^*[a] \equiv (0, v + v', \text{TRIE}(\emptyset), \text{KEC}(()))$$

$$(85) \quad \sigma^*[s]_b \equiv \sigma[s]_b - v$$

where v' is the account's pre-existing value, in the event it was previously in existence:

$$(86) \quad v' \equiv \begin{cases} 0 & \text{if } \sigma[a] = \emptyset \\ \sigma[a]_b & \text{otherwise} \end{cases}$$

Finally, the account is initialised through the execution of the initialising EVM code \mathbf{i} according to the execution model (see section 9). Code execution can effect several events that are not internal to the execution state: the account's storage can be altered, further accounts can be created and further message calls can be made. As such, the code execution function Ξ evaluates to a tuple of the resultant state σ^{**} , available gas remaining g^{**} , the accrued substate A and the body code of the account \mathbf{o} .

$$(87) \quad (\sigma^{**}, g^{**}, A, \mathbf{o}) \equiv \Xi(\sigma^*, g, I)$$

where I contains the parameters of the execution environment as defined in section 9, that is:

$$(88) \quad I_a \equiv a$$

$$(89) \quad I_o \equiv o$$

$$(90) \quad I_p \equiv p$$

$$(91) \quad I_d \equiv ()$$

$$(92) \quad I_s \equiv s$$

$$(93) \quad I_v \equiv v$$

$$(94) \quad I_b \equiv \mathbf{i}$$

$$(95) \quad I_e \equiv e$$

I_d evaluates to the empty tuple as there is no input data to this call. I_H has no special treatment and is determined from the blockchain.

Code execution depletes gas, and gas may not go below zero, thus execution may exit before the code has come to a natural halting state. In this (and several other) exceptional cases we say an Out-of-Gas exception has occurred: The evaluated state is defined as being the empty set, \emptyset , and the entire create operation should have no effect on the state, effectively leaving it as it was immediately prior to attempting the creation.

If the initialization code completes successfully, a final contract-creation cost is paid, the code-deposit cost, c , proportional to the size of the created contract's code:

$$(96) \quad c \equiv G_{codedeposit} \times |\mathbf{o}|$$

If there is not enough gas remaining to pay this, i.e. $g^{**} < c$, then we also declare an Out-of-Gas exception.

The gas remaining will be zero in any such exceptional condition, i.e. if the creation was conducted as the reception of a transaction, then this doesn't affect payment of the intrinsic cost of contract creation; it is paid regardless. However, the value of the transaction is not transferred to the aborted contract's address when we are Out-of-Gas.

If such an exception does not occur, then the remaining gas is refunded to the originator and the now-altered

state is allowed to persist. Thus formally, we may specify the resultant state, gas and substate as (σ', g', A) where:

$$(97)$$

$$g' \equiv \begin{cases} 0 & \text{if } \sigma^{**} = \emptyset \\ g^{**} & \text{if } g^{**} < c \wedge H_i < N_H \\ g^{**} - c & \text{otherwise} \end{cases}$$

$$(98)$$

$$\sigma' \equiv \begin{cases} \sigma & \text{if } \sigma^{**} = \emptyset \\ \sigma^{**} & \text{if } g^{**} < c \wedge H_i < N_H \\ \sigma^{**} \text{ except:} & \\ \sigma'[a]_c = \text{KEC}(\mathbf{o}) & \text{otherwise} \end{cases}$$

The exception in the determination of σ' dictates that \mathbf{o} , the resultant byte sequence from the execution of the initialisation code, specifies the final body code for the newly-created account.

Note that the intention from block N_H onwards (*Homestead*) is that the result is either a successfully created new contract with its endowment, or no new contract with no transfer of value. Before *Homestead*, if there is not enough gas to pay c , an account at the new contract's address is created, along with all the initialisation side-effects, and the value is transferred, but no contract code is deployed.

7.1. Subtleties. Note that while the initialisation code is executing, the newly created address exists but with no intrinsic body code. Thus any message call received by it during this time causes no code to be executed. If the initialisation execution ends with a SUICIDE instruction, the matter is moot since the account will be deleted before the transaction is completed. For a normal STOP code, or if the code returned is otherwise empty, then the state is left with a zombie account, and any remaining balance will be locked into the account forever.

8. MESSAGE CALL

In the case of executing a message call, several parameters are required: sender (s), transaction originator (o), recipient (r), the account whose code is to be executed (c , usually the same as recipient), available gas (g), value (v) and gas price (p) together with an arbitrary length byte array, \mathbf{d} , the input data of the call and finally the present depth of the message-call/contract-creation stack (e).

Aside from evaluating to a new state and transaction substate, message calls also have an extra component—the output data denoted by the byte array \mathbf{o} . This is ignored when executing transactions, however message calls can be initiated due to VM-code execution and in this case this information is used.

$$(99) \quad (\sigma', g', A, \mathbf{o}) \equiv \Theta(\sigma, s, o, r, c, g, p, v, \tilde{v}, \mathbf{d}, e)$$

Note that we need to differentiate between the value that is to be transferred, v , from the value apparent in the execution context, \tilde{v} , for the DELEGATECALL instruction.

We define σ_1 , the first transitional state as the original state but with the value transferred from sender to recipient:

$$(100) \quad \sigma_1[r]_b \equiv \sigma[r]_b + v \quad \wedge \quad \sigma_1[s]_b \equiv \sigma[s]_b - v$$

Throughout the present work, it is assumed that if $\sigma_1[r]$ was originally undefined, it will be created as an account with no code or state and zero balance and nonce. Thus the previous equation should be taken to mean:

$$(101) \quad \sigma_1 \equiv \sigma'_1 \text{ except:}$$

$$(102) \quad \sigma_1[s]_b \equiv \sigma'_1[s]_b - v$$

$$(103) \quad \text{and } \sigma'_1 \equiv \sigma \text{ except:}$$

$$(104) \quad \begin{cases} \sigma'_1[r] \equiv (v, 0, \text{KEC}(\emptyset), \text{TRIE}(\emptyset)) & \text{if } \sigma[r] = \emptyset \\ \sigma'_1[r]_b \equiv \sigma[r]_b + v & \text{otherwise} \end{cases}$$

The account's associated code (identified as the fragment whose Keccak hash is $\sigma[c]_c$) is executed according to the execution model (see section 9). Just as with contract creation, if the execution halts in an exceptional fashion (i.e. due to an exhausted gas supply, stack underflow, invalid jump destination or invalid instruction), then no gas is refunded to the caller and the state is reverted to the point immediately prior to balance transfer (i.e. σ).

$$(105) \quad \sigma' \equiv \begin{cases} \sigma & \text{if } \sigma^{**} = \emptyset \\ \sigma^{**} & \text{otherwise} \end{cases}$$

$$(106) \quad (\sigma^{**}, g', s, o) \equiv \begin{cases} \Xi_{\text{ECREC}}(\sigma_1, g, I) & \text{if } r = 1 \\ \Xi_{\text{SHA256}}(\sigma_1, g, I) & \text{if } r = 2 \\ \Xi_{\text{RIPE160}}(\sigma_1, g, I) & \text{if } r = 3 \\ \Xi_{\text{ID}}(\sigma_1, g, I) & \text{if } r = 4 \\ \Xi(\sigma_1, g, I) & \text{otherwise} \end{cases}$$

$$(107) \quad I_a \equiv r$$

$$(108) \quad I_o \equiv o$$

$$(109) \quad I_p \equiv p$$

$$(110) \quad I_d \equiv d$$

$$(111) \quad I_s \equiv s$$

$$(112) \quad I_v \equiv \tilde{v}$$

$$(113) \quad I_e \equiv e$$

$$(114) \text{Let } \text{KEC}(I_b) = \sigma[c]_c$$

It is assumed that the client will have stored the pair $(\text{KEC}(I_b), I_b)$ at some point prior in order to make the determination of I_b feasible.

As can be seen, there are four exceptions to the usage of the general execution framework Ξ for evaluation of the message call: these are four so-called 'precompiled' contracts, meant as a preliminary piece of architecture that may later become *native extensions*. The four contracts in addresses 1, 2, 3 and 4 execute the elliptic curve public key recovery function, the SHA2 256-bit hash scheme, the RIPEMD 160-bit hash scheme and the identity function respectively.

Their full formal definition is in Appendix E.

9. EXECUTION MODEL

The execution model specifies how the system state is altered given a series of bytecode instructions and a small tuple of environmental data. This is specified through a formal model of a virtual state machine, known as the Ethereum Virtual Machine (EVM). It is a *quasi-Turing-complete* machine; the *quasi* qualification comes from the fact that the computation is intrinsically bounded through

a parameter, *gas*, which limits the total amount of computation done.

9.1. Basics. The EVM is a simple stack-based architecture. The word size of the machine (and thus size of stack item) is 256-bit. This was chosen to facilitate the Keccak-256 hash scheme and elliptic-curve computations. The memory model is a simple word-addressed byte array. The stack has a maximum size of 1024. The machine also has an independent storage model; this is similar in concept to the memory but rather than a byte array, it is a word-addressable word array. Unlike memory, which is volatile, storage is non volatile and is maintained as part of the system state. All locations in both storage and memory are well-defined initially as zero.

The machine does not follow the standard von Neumann architecture. Rather than storing program code in generally-accessible memory or storage, it is stored separately in a virtual ROM interactable only through a specialised instruction.

The machine can have exceptional execution for several reasons, including stack underflows and invalid instructions. Like the out-of-gas (OOG) exception, they do not leave state changes intact. Rather, the machine halts immediately and reports the issue to the execution agent (either the transaction processor or, recursively, the spawning execution environment) which will deal with it separately.

9.2. Fees Overview. Fees (denominated in gas) are charged under three distinct circumstances, all three as prerequisite to the execution of an operation. The first and most common is the fee intrinsic to the computation of the operation (see Appendix G). Secondly, gas may be deducted in order to form the payment for a subordinate message call or contract creation; this forms part of the payment for CREATE, CALL and CALLCODE. Finally, gas may be paid due to an increase in the usage of the memory.

Over an account's execution, the total fee for memory-usage payable is proportional to smallest multiple of 32 bytes that are required such that all memory indices (whether for read or write) are included in the range. This is paid for on a just-in-time basis; as such, referencing an area of memory at least 32 bytes greater than any previously indexed memory will certainly result in an additional memory usage fee. Due to this fee it is highly unlikely addresses will ever go above 32-bit bounds. That said, implementations must be able to manage this eventuality.

Storage fees have a slightly nuanced behaviour—to incentivise minimisation of the use of storage (which corresponds directly to a larger state database on all nodes), the execution fee for an operation that clears an entry in the storage is not only waived, a qualified refund is given; in fact, this refund is effectively paid up-front since the initial usage of a storage location costs substantially more than normal usage.

See Appendix H for a rigorous definition of the EVM gas cost.

9.3. Execution Environment. In addition to the system state σ , and the remaining gas for computation g , there are several pieces of important information used in

the execution environment that the execution agent must provide; these are contained in the tuple I :

- I_a , the address of the account which owns the code that is executing.
- I_o , the sender address of the transaction that originated this execution.
- I_p , the price of gas in the transaction that originated this execution.
- I_d , the byte array that is the input data to this execution; if the execution agent is a transaction, this would be the transaction data.
- I_s , the address of the account which caused the code to be executing; if the execution agent is a transaction, this would be the transaction sender.
- I_v , the value, in Wei, passed to this account as part of the same procedure as execution; if the execution agent is a transaction, this would be the transaction value.
- I_b , the byte array that is the machine code to be executed.
- I_H , the block header of the present block.
- I_e , the depth of the present message-call or contract-creation (i.e. the number of CALLs or CREATEs being executed at present).

The execution model defines the function Ξ , which can compute the resultant state σ' , the remaining gas g' , the suicide list s , the log series l , the refunds r and the resultant output, o , given these definitions:

$$(115) \quad (\sigma', g', s, l, r, o) \equiv \Xi(\sigma, g, I)$$

9.4. Execution Overview. We must now define the Ξ function. In most practical implementations this will be modelled as an iterative progression of the pair comprising the full system state, σ and the machine state, μ . Formally, we define it recursively with a function X . This uses an iterator function O (which defines the result of a single cycle of the state machine) together with functions Z which determines if the present state is an exceptional halting state of the machine and H , specifying the output data of the instruction if and only if the present state is a normal halting state of the machine.

The empty sequence, denoted $()$, is not equal to the empty set, denoted \emptyset ; this is important when interpreting the output of H , which evaluates to \emptyset when execution is to continue but a series (potentially empty) when execution should halt.

$$(116) \quad \Xi(\sigma, g, I) \equiv X_{0,1,2,4}((\sigma, \mu, A^0, I))$$

$$(117) \quad \mu_g \equiv g$$

$$(118) \quad \mu_{pc} \equiv 0$$

$$(119) \quad \mu_m \equiv (0, 0, \dots)$$

$$(120) \quad \mu_i \equiv 0$$

$$(121) \quad \mu_s \equiv ()$$

$$(122) \quad X((\sigma, \mu, A, I)) \equiv \begin{cases} (\emptyset, \mu, A^0, I, ()) & \text{if } Z(\sigma, \mu, I) \\ O(\sigma, \mu, A, I) \cdot o & \text{if } o \neq \emptyset \\ X(O(\sigma, \mu, A, I)) & \text{otherwise} \end{cases}$$

where

$$(123) \quad o \equiv H(\mu, I)$$

$$(124) \quad (a, b, c) \cdot d \equiv (a, b, c, d)$$

Note that we must drop the fourth value in the tuple returned by X to correctly evaluate Ξ , hence the subscript $X_{0,1,2,4}$.

X is thus cycled (recursively here, but implementations are generally expected to use a simple iterative loop) until either Z becomes true indicating that the present state is exceptional and that the machine must be halted and any changes discarded or until H becomes a series (rather than the empty set) indicating that the machine has reached a controlled halt.

9.4.1. Machine State. The machine state μ is defined as the tuple (g, pc, m, i, s) which are the gas available, the program counter $pc \in \mathbb{P}_{256}$, the memory contents, the active number of words in memory (counting continuously from position 0), and the stack contents. The memory contents μ_m are a series of zeroes of size 2^{256} .

For the ease of reading, the instruction mnemonics, written in small-caps (e.g. ADD), should be interpreted as their numeric equivalents; the full table of instructions and their specifics is given in Appendix H.

For the purposes of defining Z , H and O , we define w as the current operation to be executed:

$$(125) \quad w \equiv \begin{cases} I_b[\mu_{pc}] & \text{if } \mu_{pc} < \|I_b\| \\ \text{STOP} & \text{otherwise} \end{cases}$$

We also assume the fixed amounts of δ and α , specifying the stack items removed and added, both subscriptable on the instruction and an instruction cost function C evaluating to the full cost, in gas, of executing the given instruction.

9.4.2. Exceptional Halting. The exceptional halting function Z is defined as:

$$(126) \quad Z(\sigma, \mu, I) \equiv \begin{aligned} \mu_g &< C(\sigma, \mu, I) \quad \vee \\ \delta_w &= \emptyset \quad \vee \\ \|\mu_s\| &< \delta_w \quad \vee \\ (w \in \{\text{JUMP, JUMPI}\} \quad \wedge \\ \mu_s[0] &\notin D(I_b)) \quad \vee \\ \|\mu_s\| - \delta_w + \alpha_w &> 1024 \end{aligned}$$

This states that the execution is in an exceptional halting state if there is insufficient gas, if the instruction is invalid (and therefore its δ subscript is undefined), if there are insufficient stack items, if a JUMP/JUMPI destination is invalid or the new stack size would be larger than 1024. The astute reader will realise that this implies that no instruction can, through its execution, cause an exceptional halt.

9.4.3. Jump Destination Validity. We previously used D as the function to determine the set of valid jump destinations given the code that is being run. We define this as any position in the code occupied by a JUMPDEST instruction.

All such positions must be on valid instruction boundaries, rather than sitting in the data portion of PUSH operations and must appear within the explicitly defined portion of the code (rather than in the implicitly defined STOP operations that trail it).

Formally:

$$(127) \quad D(\mathbf{c}) \equiv D_J(\mathbf{c}, 0)$$

where:

$$(128) \quad D_J(\mathbf{c}, i) \equiv \begin{cases} \{\} & \text{if } i \geq |\mathbf{c}| \\ \{i\} \cup D_J(\mathbf{c}, N(i, \mathbf{c}[i])) & \text{if } \mathbf{c}[i] = \text{JUMPDEST} \\ D_J(\mathbf{c}, N(i, \mathbf{c}[i])) & \text{otherwise} \end{cases}$$

where N is the next valid instruction position in the code, skipping the data of a PUSH instruction, if any:

$$(129) \quad N(i, w) \equiv \begin{cases} i + w - \text{PUSH1} + 2 & \text{if } w \in [\text{PUSH1}, \text{PUSH32}] \\ i + 1 & \text{otherwise} \end{cases}$$

9.4.4. Normal Halting. The normal halting function H is defined:

$$(130) \quad H(\boldsymbol{\mu}, I) \equiv \begin{cases} H_{\text{RETURN}}(\boldsymbol{\mu}) & \text{if } w = \text{RETURN} \\ () & \text{if } w \in \{\text{STOP}, \text{SUICIDE}\} \\ \emptyset & \text{otherwise} \end{cases}$$

The data-returning halt operation, RETURN, has a special function H_{RETURN} , defined in Appendix H.

9.5. The Execution Cycle. Stack items are added or removed from the left-most, lower-indexed portion of the series; all other items remain unchanged:

$$(131) \quad O((\sigma, \boldsymbol{\mu}, A, I)) \equiv (\sigma', \boldsymbol{\mu}', A', I)$$

$$(132) \quad \Delta \equiv \alpha_w - \delta_w$$

$$(133) \quad \|\boldsymbol{\mu}'_s\| \equiv \|\boldsymbol{\mu}_s\| + \Delta$$

$$(134) \quad \forall x \in [\alpha_w, \|\boldsymbol{\mu}'_s\|] : \boldsymbol{\mu}'_s[x] \equiv \boldsymbol{\mu}_s[x + \Delta]$$

The gas is reduced by the instruction's gas cost and for most instructions, the program counter increments on each cycle, for the three exceptions, we assume a function J , subscripted by one of two instructions, which evaluates to the according value:

$$(135) \quad \boldsymbol{\mu}'_g \equiv \boldsymbol{\mu}_g - C(\sigma, \boldsymbol{\mu}, I)$$

$$(136) \quad \boldsymbol{\mu}'_{pc} \equiv \begin{cases} J_{\text{JUMP}}(\boldsymbol{\mu}) & \text{if } w = \text{JUMP} \\ J_{\text{JUMPI}}(\boldsymbol{\mu}) & \text{if } w = \text{JUMPI} \\ N(\boldsymbol{\mu}_{pc}, w) & \text{otherwise} \end{cases}$$

In general, we assume the memory, suicide list and system state don't change:

$$(137) \quad \boldsymbol{\mu}'_m \equiv \boldsymbol{\mu}_m$$

$$(138) \quad \boldsymbol{\mu}'_i \equiv \boldsymbol{\mu}_i$$

$$(139) \quad A' \equiv A$$

$$(140) \quad \sigma' \equiv \sigma$$

However, instructions do typically alter one or several components of these values. Altered components listed by instruction are noted in Appendix H, alongside values for α and δ and a formal description of the gas requirements.

10. BLOCKTREE TO BLOCKCHAIN

The canonical blockchain is a path from root to leaf through the entire block tree. In order to have consensus over which path it is, conceptually we identify the path that has had the most computation done upon it, or, the *heaviest* path. Clearly one factor that helps determine the heaviest path is the block number of the leaf, equivalent to the number of blocks, not counting the unmined genesis block, in the path. The longer the path, the greater the total mining effort that must have been done in order to

arrive at the leaf. This is akin to existing schemes, such as that employed in Bitcoin-derived protocols.

Since a block header includes the difficulty, the header alone is enough to validate the computation done. Any block contributes toward the total computation or *total difficulty* of a chain.

Thus we define the total difficulty of block B recursively as:

$$(141) \quad B_t \equiv B'_t + B_d$$

$$(142) \quad B' \equiv P(B_H)$$

As such given a block B , B_t is its total difficulty, B' is its parent block and B_d is its difficulty.

11. BLOCK FINALISATION

The process of finalising a block involves four stages:

- (1) Validate (or, if mining, determine) ommers;
- (2) validate (or, if mining, determine) transactions;
- (3) apply rewards;
- (4) verify (or, if mining, compute a valid) state and nonce.

11.1. Ommer Validation. The validation of ommer headers means nothing more than verifying that each ommer header is both a valid header and satisfies the relation of N th-generation ommer to the present block where $N \leq 6$. The maximum of ommer headers is two. Formally:

$$(143) \quad \|B_U\| \leq 2 \bigwedge_{U \in B_U} V(U) \wedge k(U, P(B_H)_H, 6)$$

where k denotes the “is-kin” property:

$$(144)$$

$$k(U, H, n) \equiv \begin{cases} \text{false} & \text{if } n = 0 \\ s(U, H) \\ \vee k(U, P(H)_H, n - 1) & \text{otherwise} \end{cases}$$

and s denotes the “is-sibling” property:

$$(145)$$

$$s(U, H) \equiv (P(H) = P(U) \wedge H \neq U \wedge U \notin B(H)_U)$$

where $B(H)$ is the block of the corresponding header H .

11.2. Transaction Validation. The given `gasUsed` must correspond faithfully to the transactions listed: B_{H_g} , the total gas used in the block, must be equal to the accumulated gas used according to the final transaction:

$$(146) \quad B_{H_g} = \ell(\mathbf{R})_u$$

11.3. Reward Application. The application of rewards to a block involves raising the balance of the accounts of the beneficiary address of the block and each ommer by a certain amount. We raise the block's beneficiary account by R_b ; for each ommer, we raise the block's beneficiary by an additional $\frac{1}{32}$ of the block reward and the beneficiary of the ommer gets rewarded depending on the block number. Formally we define the function Ω :

$$(147) \quad \Omega(B, \sigma) \equiv \sigma' : \sigma' = \sigma \text{ except:}$$

$$(148) \quad \sigma'[B_{H_c}]_b = \sigma[B_{H_c}]_b + (1 + \frac{\|B_U\|}{32})R_b$$

$$(149) \quad \forall U \in B_U :$$

$$\sigma'[U_c]_b = \sigma[U_c]_b + (1 + \frac{1}{8}(U_i - B_{H_i}))R_b$$

If there are collisions of the beneficiary addresses between ommers and the block (i.e. two ommers with the

same beneficiary address or an owner with the same beneficiary address as the present block), additions are applied cumulatively.

We define the block reward as 5 Ether:

$$(150) \quad \text{Let } R_b = 5 \times 10^{18}$$

11.4. State & Nonce Validation. We may now define the function, Γ , that maps a block B to its initiation state:

$$(151) \quad \Gamma(B) \equiv \begin{cases} \sigma_0 & \text{if } P(B_H) = \emptyset \\ \sigma_i : \text{TRIE}(L_S(\sigma_i)) = P(B_H)_{H_r} & \text{otherwise} \end{cases}$$

Here, $\text{TRIE}(L_S(\sigma_i))$ means the hash of the root node of a trie of state σ_i ; it is assumed that implementations will store this in the state database, trivial and efficient since the trie is by nature an immutable data structure.

And finally define Φ , the block transition function, which maps an incomplete block B to a complete block B' :

$$(152) \quad \Phi(B) \equiv B' : B' = B^* \text{ except:}$$

$$(153) \quad B'_n = n : x \leq \frac{2^{256}}{H_d}$$

$$(154) \quad B'_m = m \text{ with } (x, m) = \text{PoW}(B_R^*, n, \mathbf{d})$$

$$(155) \quad B^* \equiv B \text{ except: } B_r^* = r(\Pi(\Gamma(B)), B)$$

With \mathbf{d} being a dataset as specified in appendix J.

As specified at the beginning of the present work, Π is the state-transition function, which is defined in terms of Ω , the block finalisation function and Υ , the transaction-evaluation function, both now well-defined.

As previously detailed, $\mathbf{R}[n]_\sigma$, $\mathbf{R}[n]_1$ and $\mathbf{R}[n]_u$ are the n th corresponding states, logs and cumulative gas used after each transaction ($\mathbf{R}[n]_b$, the fourth component in the tuple, has already been defined in terms of the logs). The former is defined simply as the state resulting from applying the corresponding transaction to the state resulting from the previous transaction (or the block's initial state in the case of the first such transaction):

$$(156) \quad \mathbf{R}[n]_\sigma = \begin{cases} \Gamma(B) & \text{if } n < 0 \\ \Upsilon(\mathbf{R}[n-1]_\sigma, B_T[n]) & \text{otherwise} \end{cases}$$

In the case of $\mathbf{R}[n]_u$, we take a similar approach defining each item as the gas used in evaluating the corresponding transaction summed with the previous item (or zero, if it is the first), giving us a running total:

$$(157) \quad \mathbf{R}[n]_u = \begin{cases} 0 & \text{if } n < 0 \\ \Upsilon^g(\mathbf{R}[n-1]_\sigma, B_T[n]) + \mathbf{R}[n-1]_u & \text{otherwise} \end{cases}$$

For $\mathbf{R}[n]_1$, we utilise the Υ^1 function that we conveniently defined in the transaction execution function.

$$(158) \quad \mathbf{R}[n]_1 = \Upsilon^1(\mathbf{R}[n-1]_\sigma, B_T[n])$$

Finally, we define Π as the new state given the block reward function Ω applied to the final transaction's resultant state, $\ell(B_R)_\sigma$:

$$(159) \quad \Pi(\sigma, B) \equiv \Omega(B, \ell(\mathbf{R})_\sigma)$$

Thus the complete block-transition mechanism, less PoW, the proof-of-work function is defined.

11.5. Mining Proof-of-Work. The mining proof-of-work (PoW) exists as a cryptographically secure nonce that proves beyond reasonable doubt that a particular amount of computation has been expended in the determination of some token value n . It is utilised to enforce the blockchain security by giving meaning and credence to the notion of difficulty (and, by extension, total difficulty). However, since mining new blocks comes with an attached reward, the proof-of-work not only functions as a method of securing confidence that the blockchain will remain canonical into the future, but also as a wealth distribution mechanism.

For both reasons, there are two important goals of the proof-of-work function; firstly, it should be as accessible as possible to as many people as possible. The requirement of, or reward from, specialised and uncommon hardware should be minimised. This makes the distribution model as open as possible, and, ideally, makes the act of mining a simple swap from electricity to Ether at roughly the same rate for anyone around the world.

Secondly, it should not be possible to make super-linear profits, and especially not so with a high initial barrier. Such a mechanism allows a well-funded adversary to gain a troublesome amount of the network's total mining power and as such gives them a super-linear reward (thus skewing distribution in their favour) as well as reducing the network security.

One plague of the Bitcoin world is ASICs. These are specialised pieces of compute hardware that exist only to do a single task. In Bitcoin's case the task is the SHA256 hash function. While ASICs exist for a proof-of-work function, both goals are placed in jeopardy. Because of this, a proof-of-work function that is ASIC-resistant (i.e. difficult or economically inefficient to implement in specialised compute hardware) has been identified as the proverbial silver bullet.

Two directions exist for ASIC resistance; firstly make it sequential memory-hard, i.e. engineer the function such that the determination of the nonce requires a lot of memory and bandwidth such that the memory cannot be used in parallel to discover multiple nonces simultaneously. The second is to make the type of computation it would need to do general-purpose; the meaning of "specialised hardware" for a general-purpose task set is, naturally, general purpose hardware and as such commodity desktop computers are likely to be pretty close to "specialised hardware" for the task. For Ethereum 1.0 we have chosen the first path.

More formally, the proof-of-work function takes the form of PoW:

$$(160) \quad m = H_m \wedge n \leq \frac{2^{256}}{H_d} \text{ with } (m, n) = \text{PoW}(H_R, H_n, \mathbf{d})$$

Where H_R is the new block's header but *without* the nonce and mix-hash components; H_n is the nonce of the header; \mathbf{d} is a large data set needed to compute the mix-Hash and H_d is the new block's difficulty value (i.e. the block difficulty from section 10). PoW is the proof-of-work function which evaluates to an array with the first item being the mixHash and the second item being a pseudo-random number cryptographically dependent on H and \mathbf{d} . The underlying algorithm is called Ethash and is described below.

11.5.1. Ethash. Ethash is the planned PoW algorithm for Ethereum 1.0. It is the latest version of Dagger-Hashimoto, introduced by Buterin [2013b] and Dryja [2014], although it can no longer appropriately be called that since many of the original features of both algorithms have been drastically changed in the last month of research and development. The general route that the algorithm takes is as follows:

There exists a seed which can be computed for each block by scanning through the block headers up until that point. From the seed, one can compute a pseudorandom cache, $J_{cacheinit}$ bytes in initial size. Light clients store the cache. From the cache, we can generate a dataset, $J_{datasetinit}$ bytes in initial size, with the property that each item in the dataset depends on only a small number of items from the cache. Full clients and miners store the dataset. The dataset grows linearly with time.

Mining involves grabbing random slices of the dataset and hashing them together. Verification can be done with low memory by using the cache to regenerate the specific pieces of the dataset that you need, so you only need to store the cache. The large dataset is updated once every J_{epoch} blocks, so the vast majority of a miner's effort will be reading the dataset, not making changes to it. The mentioned parameters as well as the algorithm is explained in detail in appendix J.

12. IMPLEMENTING CONTRACTS

There are several patterns of contracts engineering that allow particular useful behaviours; two of these that I will briefly discuss are data feeds and random numbers.

12.1. Data Feeds. A data feed contract is one which provides a single service: it gives access to information from the external world within Ethereum. The accuracy and timeliness of this information is not guaranteed and it is the task of a secondary contract author—the contract that utilises the data feed—to determine how much trust can be placed in any single data feed.

The general pattern involves a single contract within Ethereum which, when given a message call, replies with some timely information concerning an external phenomenon. An example might be the local temperature of New York City. This would be implemented as a contract that returned that value of some known point in storage. Of course this point in storage must be maintained with the correct such temperature, and thus the second part of the pattern would be for an external server to run an Ethereum node, and immediately on discovery of a new block, creates a new valid transaction, sent to the contract, updating said value in storage. The contract's code would accept such updates only from the identity contained on said server.

12.2. Random Numbers. Providing random numbers within a deterministic system is, naturally, an impossible task. However, we can approximate with pseudo-random numbers by utilising data which is generally unknowable at the time of transacting. Such data might include the block's hash, the block's timestamp and the block's beneficiary address. In order to make it hard for malicious miner to control those values, one should use the BLOCKHASH operation in order to use hashes of the previous 256 blocks as pseudo-random numbers. For a series of such numbers,

a trivial solution would be to add some constant amount and hashing the result.

13. FUTURE DIRECTIONS

The state database won't be forced to maintain all past state trie structures into the future. It should maintain an age for each node and eventually discard nodes that are neither recent enough nor checkpoints; checkpoints, or a set of nodes in the database that allow a particular block's state trie to be traversed, could be used to place a maximum limit on the amount of computation needed in order to retrieve any state throughout the blockchain.

Blockchain consolidation could be used in order to reduce the amount of blocks a client would need to download to act as a full, mining, node. A compressed archive of the trie structure at given points in time (perhaps one in every 10,000th block) could be maintained by the peer network, effectively recasting the genesis block. This would reduce the amount to be downloaded to a single archive plus a hard maximum limit of blocks.

Finally, blockchain compression could perhaps be conducted: nodes in state trie that haven't sent/received a transaction in some constant amount of blocks could be thrown out, reducing both Ether-leakage and the growth of the state database.

13.1. Scalability. Scalability remains an eternal concern. With a generalised state transition function, it becomes difficult to partition and parallelise transactions to apply the divide-and-conquer strategy. Unaddressed, the dynamic value-range of the system remains essentially fixed and as the average transaction value increases, the less valuable of them become ignored, being economically pointless to include in the main ledger. However, several strategies exist that may potentially be exploited to provide a considerably more scalable protocol.

Some form of hierarchical structure, achieved by either consolidating smaller lighter-weight chains into the main block or building the main block through the incremental combination and adhesion (through proof-of-work) of smaller transaction sets may allow parallelisation of transaction combination and block-building. Parallelism could also come from a prioritised set of parallel blockchains, consolidated each block and with duplicate or invalid transactions thrown out accordingly.

Finally, verifiable computation, if made generally available and efficient enough, may provide a route to allow the proof-of-work to be the verification of final state.

14. CONCLUSION

I have introduced, discussed and formally defined the protocol of Ethereum. Through this protocol the reader may implement a node on the Ethereum network and join others in a decentralised secure social operating system. Contracts may be authored in order to algorithmically specify and autonomously enforce rules of interaction.

15. ACKNOWLEDGEMENTS

Many thanks to Aeron Buchanan for authoring the Homestead revisions, Christoph Jentzsch for authoring the Ethash algorithm and Yoichi Hirai for doing most of the

EIP-150 changes. Important maintenance, useful corrections and suggestions were provided by a number of others from the Ethereum DEV organisation and Ethereum community at large including Gustav Simonsson, Paweł Bylica, Jutta Steiner, Nick Savers, Viktor Trón, Marko Simovic, Giacomo Tazzari and, of course, Vitalik Buterin.

REFERENCES

- Jacob Aron. BitCoin software finds new life. *New Scientist*, 213(2847):20, 2012.
- Adam Back. Hashcash - Amortizable Publicly Auditable Cost-Functions. 2002. URL [{http://www.hashcash.org/papers/amortizable.pdf}](http://www.hashcash.org/papers/amortizable.pdf).
- Roman Boutellier and Mareike Heinzen. Pirates, Pioneers, Innovators and Imitators. In *Growth Through Innovation*, pages 85–96. Springer, 2014.
- Vitalik Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. 2013a. URL [{http://ethereum.org/ethereum.html}](http://ethereum.org/ethereum.html).
- Vitalik Buterin. Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alternative. 2013b. URL [{http://vitalik.ca/ethereum/dagger.html}](http://vitalik.ca/ethereum/dagger.html).
- Thaddeus Dryja. Hashimoto: I/O bound proof of work. 2014. URL [{https://mirrorx.com/files/hashimoto.pdf}](https://mirrorx.com/files/hashimoto.pdf).
- Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *In 12th Annual International Cryptology Conference*, pages 139–147, 1992.
- Phong Vo Glenn Fowler, Landon Curt Noll. FowlerNollVo hash function. 1991. URL [{https://en.wikipedia.org/wiki/Fowler%20%93Noll%20%93Vo_hash_function#cite_note-2}](https://en.wikipedia.org/wiki/Fowler%20%93Noll%20%93Vo_hash_function#cite_note-2).
- Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 119–132. Springer, 2004.
- Sergio Demian Lerner. Strict Memory Hard Hashing Functions. 2014. URL [{http://www.hashcash.org/papers/memohash.pdf}](http://www.hashcash.org/papers/memohash.pdf).
- Mark Miller. The Future of Law. In *paper delivered at the Extro 3 Conference (August 9)*, 1997.
- Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1:2012, 2008.
- Meni Rosenfeld. Overview of Colored Coins. 2012. URL [{https://bitcoil.co.il/BitcoinX.pdf}](https://bitcoil.co.il/BitcoinX.pdf).
- Yonatan Sompolinsky and Aviv Zohar. Accelerating Bitcoin’s Transaction Processing. Fast Money Grows on Trees, Not Chains, 2013. URL [{Cryptology ePrint Archive, Report 2013/881} . http://eprint.iacr.org/](http://eprint.iacr.org/).
- Simon Sprankel. Technical Basis of Digital Currencies, 2013.
- Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.
- Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gn Sirer. Karma: A secure economic framework for peer-to-peer resource sharing, 2003.
- J. R. Willett. MasterCoin Complete Specification. 2013. URL [{https://github.com/mastercoin-MSC/spec}](https://github.com/mastercoin-MSC/spec).

APPENDIX A. TERMINOLOGY

External Actor: A person or other entity able to interface to an Ethereum node, but external to the world of Ethereum. It can interact with Ethereum through depositing signed Transactions and inspecting the blockchain and associated state. Has one (or more) intrinsic Accounts.

Address: A 160-bit code used for identifying Accounts.

Account: Accounts have an intrinsic balance and transaction count maintained as part of the Ethereum state. They also have some (possibly empty) EVM Code and a (possibly empty) Storage State associated with them. Though homogenous, it makes sense to distinguish between two practical types of account: those with empty associated EVM Code (thus the account balance is controlled, if at all, by some external entity) and those with non-empty associated EVM Code (thus the account represents an Autonomous Object). Each Account has a single Address that identifies it.

Transaction: A piece of data, signed by an External Actor. It represents either a Message or a new Autonomous Object. Transactions are recorded into each block of the blockchain.

Autonomous Object: A notional object existent only within the hypothetical state of Ethereum. Has an intrinsic address and thus an associated account; the account will have non-empty associated EVM Code. Incorporated only as the Storage State of that account.

Storage State: The information particular to a given Account that is maintained between the times that the Account’s associated EVM Code runs.

Message: Data (as a set of bytes) and Value (specified as Ether) that is passed between two Accounts, either through the deterministic operation of an Autonomous Object or the cryptographically secure signature of the Transaction.

Message Call: The act of passing a message from one Account to another. If the destination account is associated with non-empty EVM Code, then the VM will be started with the state of said Object and the Message acted upon. If the message sender is an Autonomous Object, then the Call passes any data returned from the VM operation.

Gas: The fundamental network cost unit. Paid for exclusively by Ether (as of PoC-4), which is converted freely to and from Gas as required. Gas does not exist outside of the internal Ethereum computation engine; its price is set by the Transaction and miners are free to ignore Transactions whose Gas price is too low.

Contract: Informal term used to mean both a piece of EVM Code that may be associated with an Account or an Autonomous Object.

Object: Synonym for Autonomous Object.

App: An end-user-visible application hosted in the Ethereum Browser.

Ethereum Browser: (aka Ethereum Reference Client) A cross-platform GUI of an interface similar to a simplified browser (a la Chrome) that is able to host sandboxed applications whose backend is purely on the Ethereum protocol.

Ethereum Virtual Machine: (aka EVM) The virtual machine that forms the key part of the execution model for an Account's associated EVM Code.

Ethereum Runtime Environment: (aka ERE) The environment which is provided to an Autonomous Object executing in the EVM. Includes the EVM but also the structure of the world state on which the EVM relies for certain I/O instructions including CALL & CREATE.

EVM Code: The bytecode that the EVM can natively execute. Used to formally specify the meaning and ramifications of a message to an Account.

EVM Assembly: The human-readable form of EVM-code.

LLL: The Lisp-like Low-level Language, a human-writable language used for authoring simple contracts and general low-level language toolkit for trans-compiling to.

APPENDIX B. RECURSIVE LENGTH PREFIX

This is a serialisation method for encoding arbitrarily structured binary data (byte arrays).

We define the set of possible structures \mathbb{T} :

$$(161) \quad \mathbb{T} \equiv \mathbb{L} \cup \mathbb{B}$$

$$(162) \quad \mathbb{L} \equiv \{\mathbf{t} : \mathbf{t} = (\mathbf{t}[0], \mathbf{t}[1], \dots) \wedge \forall_{n < \|\mathbf{t}\|} \mathbf{t}[n] \in \mathbb{T}\}$$

$$(163) \quad \mathbb{B} \equiv \{\mathbf{b} : \mathbf{b} = (\mathbf{b}[0], \mathbf{b}[1], \dots) \wedge \forall_{n < \|\mathbf{b}\|} \mathbf{b}[n] \in \mathbb{O}\}$$

Where \mathbb{O} is the set of bytes. Thus \mathbb{B} is the set of all sequences of bytes (otherwise known as byte-arrays, and a leaf if imagined as a tree), \mathbb{L} is the set of all tree-like (sub-)structures that are not a single leaf (a branch node if imagined as a tree) and \mathbb{T} is the set of all byte-arrays and such structural sequences.

We define the RLP function as RLP through two sub-functions, the first handling the instance when the value is a byte array, the second when it is a sequence of further values:

$$(164) \quad \text{RLP}(\mathbf{x}) \equiv \begin{cases} R_b(\mathbf{x}) & \text{if } \mathbf{x} \in \mathbb{B} \\ R_l(\mathbf{x}) & \text{otherwise} \end{cases}$$

If the value to be serialised is a byte-array, the RLP serialisation takes one of three forms:

- If the byte-array contains solely a single byte and that single byte is less than 128, then the input is exactly equal to the output.
- If the byte-array contains fewer than 56 bytes, then the output is equal to the input prefixed by the byte equal to the length of the byte array plus 128.
- Otherwise, the output is equal to the input prefixed by the minimal-length byte-array which when interpreted as a big-endian integer is equal to the length of the input byte array, which is itself prefixed by the number of bytes required to faithfully encode this length value plus 183.

Formally, we define R_b :

$$(165) \quad R_b(\mathbf{x}) \equiv \begin{cases} \mathbf{x} & \text{if } \|\mathbf{x}\| = 1 \wedge \mathbf{x}[0] < 128 \\ (128 + \|\mathbf{x}\|) \cdot \mathbf{x} & \text{else if } \|\mathbf{x}\| < 56 \\ (183 + \|\text{BE}(\|\mathbf{x}\|)\|) \cdot \text{BE}(\|\mathbf{x}\|) \cdot \mathbf{x} & \text{otherwise} \end{cases}$$

$$(166) \quad \text{BE}(x) \equiv (b_0, b_1, \dots) : b_0 \neq 0 \wedge x = \sum_{n=0}^{n < \|\mathbf{b}\|} b_n \cdot 256^{\|\mathbf{b}\|-1-n}$$

$$(167) \quad (a) \cdot (b, c) \cdot (d, e) = (a, b, c, d, e)$$

Thus BE is the function that expands a positive integer value to a big-endian byte array of minimal length and the dot operator performs sequence concatenation.

If instead, the value to be serialised is a sequence of other items then the RLP serialisation takes one of two forms:

- If the concatenated serialisations of each contained item is less than 56 bytes in length, then the output is equal to that concatenation prefixed by the byte equal to the length of this byte array plus 192.
- Otherwise, the output is equal to the concatenated serialisations prefixed by the minimal-length byte-array which when interpreted as a big-endian integer is equal to the length of the concatenated serialisations byte array, which is itself prefixed by the number of bytes required to faithfully encode this length value plus 247.

Thus we finish by formally defining R_l :

$$(168) \quad R_l(\mathbf{x}) \equiv \begin{cases} (192 + \|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{if } \|s(\mathbf{x})\| < 56 \\ (247 + \|\text{BE}(\|s(\mathbf{x})\|)\|) \cdot \text{BE}(\|s(\mathbf{x})\|) \cdot s(\mathbf{x}) & \text{otherwise} \end{cases}$$

$$(169) \quad s(\mathbf{x}) \equiv \text{RLP}(\mathbf{x}_0) \cdot \text{RLP}(\mathbf{x}_1) \dots$$

If RLP is used to encode a scalar, defined only as a positive integer (\mathbb{P} or any x for \mathbb{P}_x), it must be specified as the shortest byte array such that the big-endian interpretation of it is equal. Thus the RLP of some positive integer i is defined as:

$$(170) \quad \text{RLP}(i : i \in \mathbb{P}) \equiv \text{RLP}(\text{BE}(i))$$

When interpreting RLP data, if an expected fragment is decoded as a scalar and leading zeroes are found in the byte sequence, clients are required to consider it non-canonical and treat it in the same manner as otherwise invalid RLP data, dismissing it completely.

There is no specific canonical encoding format for signed or floating-point values.

APPENDIX C. HEX-PREFIX ENCODING

Hex-prefix encoding is an efficient method of encoding an arbitrary number of nibbles as a byte array. It is able to store an additional flag which, when used in the context of the trie (the only context in which it is used), disambiguates between node types.

It is defined as the function HP which maps from a sequence of nibbles (represented by the set \mathbb{Y}) together with a boolean value to a sequence of bytes (represented by the set \mathbb{B}):

$$(171) \quad \text{HP}(\mathbf{x}, t) : \mathbf{x} \in \mathbb{Y} \equiv \begin{cases} (16f(t), 16\mathbf{x}[0] + \mathbf{x}[1], 16\mathbf{x}[2] + \mathbf{x}[3], \dots) & \text{if } \|\mathbf{x}\| \text{ is even} \\ (16(f(t) + 1) + \mathbf{x}[0], 16\mathbf{x}[1] + \mathbf{x}[2], 16\mathbf{x}[3] + \mathbf{x}[4], \dots) & \text{otherwise} \end{cases}$$

$$(172) \quad f(t) \equiv \begin{cases} 2 & \text{if } t \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus the high nibble of the first byte contains two flags; the lowest bit encoding the oddness of the length and the second-lowest encoding the flag t . The low nibble of the first byte is zero in the case of an even number of nibbles and the first nibble in the case of an odd number. All remaining nibbles (now an even number) fit properly into the remaining bytes.

APPENDIX D. MODIFIED MERKLE PATRICIA TREE

The modified Merkle Patricia tree (trie) provides a persistent data structure to map between arbitrary-length binary data (byte arrays). It is defined in terms of a mutable data structure to map between 256-bit binary fragments and arbitrary-length binary data, typically implemented as a database. The core of the trie, and its sole requirement in terms of the protocol specification is to provide a single value that identifies a given set of key-value pairs, which may be either a 32 byte sequence or the empty byte sequence. It is left as an implementation consideration to store and maintain the structure of the trie in a manner that allows effective and efficient realisation of the protocol.

Formally, we assume the input value \mathcal{J} , a set containing pairs of byte sequences:

$$(173) \quad \mathcal{J} = \{(\mathbf{k}_0 \in \mathbb{B}, \mathbf{v}_0 \in \mathbb{B}), (\mathbf{k}_1 \in \mathbb{B}, \mathbf{v}_1 \in \mathbb{B}), \dots\}$$

When considering such a sequence, we use the common numeric subscript notation to refer to a tuple's key or value, thus:

$$(174) \quad \forall_{I \in \mathcal{J}} I \equiv (I_0, I_1)$$

Any series of bytes may also trivially be viewed as a series of nibbles, given an endian-specific notation; here we assume big-endian. Thus:

$$(175) \quad y(\mathcal{J}) = \{(\mathbf{k}'_0 \in \mathbb{Y}, \mathbf{v}_0 \in \mathbb{B}), (\mathbf{k}'_1 \in \mathbb{Y}, \mathbf{v}_1 \in \mathbb{B}), \dots\}$$

$$(176) \quad \forall_n \quad \forall_{i: i < 2\|\mathbf{k}_n\|} \quad \mathbf{k}'_n[i] \equiv \begin{cases} \lfloor \mathbf{k}_n[i \div 2] \div 16 \rfloor & \text{if } i \text{ is even} \\ \mathbf{k}_n[\lfloor i \div 2 \rfloor] \bmod 16 & \text{otherwise} \end{cases}$$

We define the function TRIE , which evaluates to the root of the trie that represents this set when encoded in this structure:

$$(177) \quad \text{TRIE}(\mathcal{J}) \equiv \text{KEC}(c(\mathcal{J}, 0))$$

We also assume a function n , the trie's node cap function. When composing a node, we use RLP to encode the structure. As a means of reducing storage complexity, for nodes whose composed RLP is fewer than 32 bytes, we store the RLP directly; for those larger we assert prescience of the byte array whose Keccak hash evaluates to our reference. Thus we define in terms of c , the node composition function:

$$(178) \quad n(\mathcal{J}, i) \equiv \begin{cases} () & \text{if } \mathcal{J} = \emptyset \\ c(\mathcal{J}, i) & \text{if } \|c(\mathcal{J}, i)\| < 32 \\ \text{KEC}(c(\mathcal{J}, i)) & \text{otherwise} \end{cases}$$

In a manner similar to a radix tree, when the trie is traversed from root to leaf, one may build a single key-value pair. The key is accumulated through the traversal, acquiring a single nibble from each branch node (just as with a radix tree). Unlike a radix tree, in the case of multiple keys sharing the same prefix or in the case of a single key having

a unique suffix, two optimising nodes are provided. Thus while traversing, one may potentially acquire multiple nibbles from each of the other two node types, extension and leaf. There are three kinds of nodes in the trie:

Leaf: A two-item structure whose first item corresponds to the nibbles in the key not already accounted for by the accumulation of keys and branches traversed from the root. The hex-prefix encoding method is used and the second parameter to the function is required to be *true*.

Extension: A two-item structure whose first item corresponds to a series of nibbles of size greater than one that are shared by at least two distinct keys past the accumulation of nibbles keys and branches as traversed from the root. The hex-prefix encoding method is used and the second parameter to the function is required to be *false*.

Branch: A 17-item structure whose first sixteen items correspond to each of the sixteen possible nibble values for the keys at this point in their traversal. The 17th item is used in the case of this being a terminator node and thus a key being ended at this point in its traversal.

A branch is then only used when necessary; no branch nodes may exist that contain only a single non-zero entry. We may formally define this structure with the structural composition function c :

$$(179) \quad c(\mathcal{J}, i) \equiv \begin{cases} \text{RLP}\left((\text{HP}(I_0[i..(\|I_0\| - 1)], \text{true}), I_1)\right) & \text{if } \|\mathcal{J}\| = 1 \text{ where } \exists I : I \in \mathcal{J} \\ \text{RLP}\left((\text{HP}(I_0[i..(j - 1)], \text{false}), n(\mathcal{J}, j))\right) & \text{if } i \neq j \text{ where } j = \arg \max_x : \exists I : \|I\| = x : \forall I \in \mathcal{J} : I_0[0..(x - 1)] = 1 \\ \text{RLP}\left((u(0), u(1), \dots, u(15), v)\right) & \text{otherwise where } u(j) \equiv n(\{I : I \in \mathcal{J} \wedge I_0[i] = j\}, i + 1) \\ & v = \begin{cases} I_1 & \text{if } \exists I : I \in \mathcal{J} \wedge \|I_0\| = i \\ () & \text{otherwise} \end{cases} \end{cases}$$

D.1. Trie Database. Thus no explicit assumptions are made concerning what data is stored and what is not, since that is an implementation-specific consideration; we simply define the identity function mapping the key-value set \mathcal{J} to a 32-byte hash and assert that only a single such hash exists for any \mathcal{J} , which though not strictly true is accurate within acceptable precision given the Keccak hash's collision resistance. In reality, a sensible implementation will not fully recompute the trie root hash for each set.

A reasonable implementation will maintain a database of nodes determined from the computation of various tries or, more formally, it will memoise the function c . This strategy uses the nature of the trie to both easily recall the contents of any previous key-value set and to store multiple such sets in a very efficient manner. Due to the dependency relationship, Merkle-proofs may be constructed with an $O(\log N)$ space requirement that can demonstrate a particular leaf must exist within a trie of a given root hash.

APPENDIX E. PRECOMPILED CONTRACTS

For each precompiled contract, we make use of a template function, Ξ_{PRE} , which implements the out-of-gas checking.

$$(180) \quad \Xi_{\text{PRE}}(\sigma, g, I) \equiv \begin{cases} (\emptyset, 0, A^0, ()) & \text{if } g < g_r \\ (\sigma, g - g_r, A^0, \mathbf{o}) & \text{otherwise} \end{cases}$$

The precompiled contracts each use these definitions and provide specifications for the \mathbf{o} (the output data) and g_r , the gas requirements.

For the elliptic curve DSA recover VM execution function, we also define \mathbf{d} to be the input data, well-defined for an infinite length by appending zeroes as required. Importantly in the case of an invalid signature ($\text{ECDSARECOVER}(h, v, r, s) = \emptyset$), then we have no output.

$$(181) \quad \Xi_{\text{ECREC}} \equiv \Xi_{\text{PRE}} \text{ where:}$$

$$(182) \quad g_r = 3000$$

$$(183) \quad |\mathbf{o}| = \begin{cases} 0 & \text{if } \text{ECDSARECOVER}(h, v, r, s) = \emptyset \\ 32 & \text{otherwise} \end{cases}$$

$$(184) \quad \text{if } |\mathbf{o}| = 32 :$$

$$(185) \quad \mathbf{o}[0..11] = 0$$

$$(186) \quad \mathbf{o}[12..31] = \text{KEC}(\text{ECDSARECOVER}(h, v, r, s))[12..31] \text{ where:}$$

$$(187) \quad \mathbf{d}[0..(|I_d| - 1)] = I_d$$

$$(188) \quad \mathbf{d}[|I_d|..] = (0, 0, \dots)$$

$$(189) \quad h = \mathbf{d}[0..31]$$

$$(190) \quad v = \mathbf{d}[32..63]$$

$$(191) \quad r = \mathbf{d}[64..95]$$

$$(192) \quad s = \mathbf{d}[96..127]$$

The two hash functions, RIPEMD-160 and SHA2-256 are more trivially defined as an almost pass-through operation. Their gas usage is dependent on the input data size, a factor rounded up to the nearest number of words.

$$(193) \quad \Xi_{\text{SHA256}} \equiv \Xi_{\text{PRE}} \text{ where:}$$

$$(194) \quad g_r = 60 + 12 \lceil \frac{|I_d|}{32} \rceil$$

$$(195) \quad o[0..31] = \text{SHA256}(I_d)$$

$$(196) \quad \Xi_{\text{RIPEMD160}} \equiv \Xi_{\text{PRE}} \text{ where:}$$

$$(197) \quad g_r = 600 + 120 \lceil \frac{|I_d|}{32} \rceil$$

$$(198) \quad o[0..11] = 0$$

$$(199) \quad o[12..31] = \text{RIPEMD160}(I_d)$$

(200)

For the purposes here, we assume we have well-defined standard cryptographic functions for RIPEMD-160 and SHA2-256 of the form:

$$(201) \quad \text{SHA256}(i \in \mathbb{B}) \equiv o \in \mathbb{B}_{32}$$

$$(202) \quad \text{RIPEMD160}(i \in \mathbb{B}) \equiv o \in \mathbb{B}_{20}$$

Finally, the fourth contract, the identity function Ξ_{ID} simply defines the output as the input:

$$(203) \quad \Xi_{\text{ID}} \equiv \Xi_{\text{PRE}} \text{ where:}$$

$$(204) \quad g_r = 15 + 3 \lceil \frac{|I_d|}{32} \rceil$$

$$(205) \quad o = I_d$$

APPENDIX F. SIGNING TRANSACTIONS

The method of signing transactions is similar to the ‘Electrum style signatures’; it utilises the SECP-256k1 curve as described by Gura et al. [2004].

It is assumed that the sender has a valid private key p_r , which is a randomly selected positive integer (represented as a byte array of length 32 in big-endian form) in the range $[1, \text{secp256k1n} - 1]$.

We assert the functions ECDSASIGN, ECDSARESTORE and ECDSAPUBKEY. These are formally defined in the literature.

$$(206) \quad \text{ECDSAPUBKEY}(p_r \in \mathbb{B}_{32}) \equiv p_u \in \mathbb{B}_{64}$$

$$(207) \quad \text{ECDSASIGN}(e \in \mathbb{B}_{32}, p_r \in \mathbb{B}_{32}) \equiv (v \in \mathbb{B}_1, r \in \mathbb{B}_{32}, s \in \mathbb{B}_{32})$$

$$(208) \quad \text{ECDSARECOVER}(e \in \mathbb{B}_{32}, v \in \mathbb{B}_1, r \in \mathbb{B}_{32}, s \in \mathbb{B}_{32}) \equiv p_u \in \mathbb{B}_{64}$$

Where p_u is the public key, assumed to be a byte array of size 64 (formed from the concatenation of two positive integers each $< 2^{256}$) and p_r is the private key, a byte array of size 32 (or a single positive integer in the aforementioned range). It is assumed that v is the ‘recovery id’, a 1 byte value specifying the sign and finiteness of the curve point; this value is in the range of [27, 30], however we declare the upper two possibilities, representing infinite values, invalid.

We declare that a signature is invalid unless all the following conditions are true:

$$(209) \quad 0 < r < \text{secp256k1n}$$

$$(210) \quad 0 < s < \begin{cases} \text{secp256k1n} & \text{if } H_i < N_H \\ \text{secp256k1n} \div 2 & \text{otherwise} \end{cases}$$

$$(211) \quad v \in \{27, 28\}$$

where:

$$(212) \quad \text{secp256k1n} = 115792089237316195423570985008687907852837564279074904382605163141518161494337$$

For a given private key, p_r , the Ethereum address $A(p_r)$ (a 160-bit value) to which it corresponds is defined as the right most 160-bits of the Keccak hash of the corresponding ECDSA public key:

$$(213) \quad A(p_r) = \mathcal{B}_{96..255}(\text{KEC}(\text{ECDSAPUBKEY}(p_r)))$$

The message hash, $h(T)$, to be signed is the Keccak hash of the transaction without the latter three signature components, formally described as T_r , T_s and T_w :

$$(214) \quad L_S(T) \equiv \begin{cases} (T_n, T_p, T_g, T_t, T_v, T_i) & \text{if } T_t = 0 \\ (T_n, T_p, T_g, T_t, T_v, T_d) & \text{otherwise} \end{cases}$$

$$(215) \quad h(T) \equiv \text{KEC}(L_S(T))$$

The signed transaction $G(T, p_r)$ is defined as:

$$(216) \quad G(T, p_r) \equiv T \text{ except:}$$

$$(217) \quad (T_w, T_r, T_s) = \text{ECDSASIGN}(h(T), p_r)$$

We may then define the sender function S of the transaction as:

$$(218) \quad S(T) \equiv \mathcal{B}_{96..255}(\text{KEC}(\text{ECDSARECOVER}(h(T), T_w, T_r, T_s)))$$

The assertion that the sender of the signed transaction equals the address of the signer should be self-evident:

$$(219) \quad \forall T : \forall p_r : S(G(T, p_r)) \equiv A(p_r)$$

APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero} .
G_{base}	2	Amount of gas to pay for operations of the set W_{base} .
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low} .
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid} .
G_{high}	10	Amount of gas to pay for operations of the set W_{high} .
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbytē}$	10	Partial payment when multiplied by $\lceil \log_{256}(\text{exponent}) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the Homestead transition.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

APPENDIX H. VIRTUAL MACHINE SPECIFICATION

When interpreting 256-bit binary values as integers, the representation is big-endian.

When a 256-bit machine datum is converted to and from a 160-bit address or hash, the rightwards (low-order for BE) 20 bytes are used and the left most 12 are discarded or filled with zeroes, thus the integer values (when the bytes are interpreted as big-endian) are equivalent.

H.1. Gas Cost. The general gas cost function, C , is defined as:

(220)

$$C(\sigma, \mu, I) \equiv C_{mem}(\mu'_i) - C_{mem}(\mu_i) + \begin{cases} C_{SSTORE}(\sigma, \mu) & \text{if } w = SSTORE \\ G_{exp} & \text{if } w = EXP \wedge \mu_s[1] = 0 \\ G_{exp} + G_{expbyte} \times (1 + \lfloor \log_{256}(\mu_s[1]) \rfloor) & \text{if } w = EXP \wedge \mu_s[1] > 0 \\ G_{verylow} + G_{copy} \times \lceil \mu_s[2] \div 32 \rceil & \text{if } w = CALLDATACOPY \vee CODECOPY \\ G_{extcode} + G_{copy} \times \lceil \mu_s[3] \div 32 \rceil & \text{if } w = EXTCODECOPY \\ G_{log} + G_{logdata} \times \mu_s[1] & \text{if } w = LOG0 \\ G_{log} + G_{logdata} \times \mu_s[1] + G_{logtopic} & \text{if } w = LOG1 \\ G_{log} + G_{logdata} \times \mu_s[1] + 2G_{logtopic} & \text{if } w = LOG2 \\ G_{log} + G_{logdata} \times \mu_s[1] + 3G_{logtopic} & \text{if } w = LOG3 \\ G_{log} + G_{logdata} \times \mu_s[1] + 4G_{logtopic} & \text{if } w = LOG4 \\ C_{CALL}(\sigma, \mu) & \text{if } w = CALL \vee CALLCODE \vee DELEGATECALL \\ C_{SUICIDE}(\sigma, \mu) & \text{if } w = SUICIDE \\ G_{create} & \text{if } w = CREATE \\ G_{sha3} + G_{sha3word} \lceil s[1] \div 32 \rceil & \text{if } w = SHA3 \\ G_{jumpdest} & \text{if } w = JUMPDEST \\ G_{sload} & \text{if } w = SLOAD \\ G_{zero} & \text{if } w \in W_{zero} \\ G_{base} & \text{if } w \in W_{base} \\ G_{verylow} & \text{if } w \in W_{verylow} \\ G_{low} & \text{if } w \in W_{low} \\ G_{mid} & \text{if } w \in W_{mid} \\ G_{high} & \text{if } w \in W_{high} \\ G_{extcode} & \text{if } w \in W_{extcode} \\ G_{balance} & \text{if } w = BALANCE \\ G_{blockhash} & \text{if } w = BLOCKHASH \end{cases}$$

$$(221) \quad w \equiv \begin{cases} I_b[\mu_{pc}] & \text{if } \mu_{pc} < \|I_b\| \\ \text{STOP} & \text{otherwise} \end{cases}$$

where:

$$(222) \quad C_{mem}(a) \equiv G_{memory} \cdot a + \left\lfloor \frac{a^2}{512} \right\rfloor$$

with C_{CALL} , $C_{SUICIDE}$ and C_{SSTORE} as specified in the appropriate section below. We define the following subsets of instructions:

$$W_{zero} = \{\text{STOP, RETURN}\}$$

$$W_{base} = \{\text{ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, POP, PC, MSIZE, GAS}\}$$

$$W_{verylow} = \{\text{ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}\}$$

$$W_{low} = \{\text{MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}\}$$

$$W_{mid} = \{\text{ADDMOD, MULMOD, JUMP}\}$$

$$W_{high} = \{\text{JUMPI}\}$$

$$W_{extcode} = \{\text{EXTCODESIZE}\}$$

Note the memory cost component, given as the product of G_{memory} and the maximum of 0 & the ceiling of the number of words in size that the memory must be over the current number of words, μ_i in order that all accesses reference valid memory whether for read or write. Such accesses must be for non-zero number of bytes.

Referencing a zero length range (e.g. by attempting to pass it as the input range to a CALL) does not require memory to be extended to the beginning of the range. μ'_i is defined as this new maximum number of words of active memory; special-cases are given where these two are not equal.

Note also that C_{mem} is the memory cost function (the expansion function being the difference between the cost before and after). It is a polynomial, with the higher-order coefficient divided and floored, and thus linear up to 724B of memory used, after which it costs substantially more.

While defining the instruction set, we defined the memory-expansion for range function, M , thus:

$$(223) \quad M(s, f, l) \equiv \begin{cases} s & \text{if } l = 0 \\ \max(s, \lceil (f + l) \div 32 \rceil) & \text{otherwise} \end{cases}$$

Another useful function is “all but one 64th” function L defined as:

$$(224) \quad L(n) \equiv n - \lfloor n/64 \rfloor$$

H.2. Instruction Set. As previously specified in section 9, these definitions take place in the final context there. In particular we assume O is the EVM state-progression function and define the terms pertaining to the next cycle’s state (σ', μ') such that:

$$(225) \quad O(\sigma, \mu, A, I) \equiv (\sigma', \mu', A', I) \quad \text{with exceptions, as noted}$$

Here given are the various exceptions to the state transition rules given in section 9 specified for each instruction, together with the additional instruction-specific definitions of J and C . For each instruction, also specified is α , the additional items placed on the stack and δ , the items removed from stack, as defined in section 9.

0s: Stop and Arithmetic Operations

All arithmetic is modulo 2^{256} unless otherwise noted.

Value	Mnemonic	δ	α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\mu'_s[0] \equiv \mu_s[0] + \mu_s[1]$
0x02	MUL	2	1	Multiplication operation. $\mu'_s[0] \equiv \mu_s[0] \times \mu_s[1]$
0x03	SUB	2	1	Subtraction operation. $\mu'_s[0] \equiv \mu_s[0] - \mu_s[1]$
0x04	DIV	2	1	Integer division operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$
0x05	SDIV	2	1	Signed integer division operation (truncated). $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ -2^{255} & \text{if } \mu_s[0] = -2^{255} \wedge \mu_s[1] = -1 \\ \text{sgn}(\mu_s[0] \div \mu_s[1]) \lfloor \mu_s[0] \div \mu_s[1] \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two’s complement signed 256-bit integers. Note the overflow semantic when -2^{255} is negated.
0x06	MOD	2	1	Modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \mu_s[0] \bmod \mu_s[1] & \text{otherwise} \end{cases}$
0x07	SMOD	2	1	Signed modulo remainder operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[1] = 0 \\ \text{sgn}(\mu_s[0]) \lfloor \mu_s[0] \bmod \mu_s[1] \rfloor & \text{otherwise} \end{cases}$ Where all values are treated as two’s complement signed 256-bit integers.
0x08	ADDMOD	3	1	Modulo addition operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[2] = 0 \\ (\mu_s[0] + \mu_s[1]) \bmod \mu_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x09	MULMOD	3	1	Modulo multiplication operation. $\mu'_s[0] \equiv \begin{cases} 0 & \text{if } \mu_s[2] = 0 \\ (\mu_s[0] \times \mu_s[1]) \bmod \mu_s[2] & \text{otherwise} \end{cases}$ All intermediate calculations of this operation are not subject to the 2^{256} modulo.
0x0a	EXP	2	1	Exponential operation. $\mu'_s[0] \equiv \mu_s[0]^{\mu_s[1]}$
0x0b	SIGNEXTEND	2	1	Extend length of two’s complement signed integer. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \begin{cases} \mu_s[1]_t & \text{if } i \leq t \text{ where } t = 256 - 8(\mu_s[0] + 1) \\ \mu_s[1]_i & \text{otherwise} \end{cases}$ $\mu_s[x]_i$ gives the i th bit (counting from zero) of $\mu_s[x]$

10s: Comparison & Bitwise Logic Operations

Value	Mnemonic	δ	α	Description
0x10	LT	2	1	Less-than comparision. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] < \mu_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x11	GT	2	1	Greater-than comparision. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] > \mu_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x12	SLT	2	1	Signed less-than comparision. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] < \mu_s[1] \\ 0 & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x13	SGT	2	1	Signed greater-than comparision. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] > \mu_s[1] \\ 0 & \text{otherwise} \end{cases}$ Where all values are treated as two's complement signed 256-bit integers.
0x14	EQ	2	1	Equality comparision. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] = \mu_s[1] \\ 0 & \text{otherwise} \end{cases}$
0x15	ISZERO	1	1	Simple not operator. $\mu'_s[0] \equiv \begin{cases} 1 & \text{if } \mu_s[0] = 0 \\ 0 & \text{otherwise} \end{cases}$
0x16	AND	2	1	Bitwise AND operation. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \mu_s[0]_i \wedge \mu_s[1]_i$
0x17	OR	2	1	Bitwise OR operation. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \mu_s[0]_i \vee \mu_s[1]_i$
0x18	XOR	2	1	Bitwise XOR operation. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \mu_s[0]_i \oplus \mu_s[1]_i$
0x19	NOT	1	1	Bitwise NOT operation. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \begin{cases} 1 & \text{if } \mu_s[0]_i = 0 \\ 0 & \text{otherwise} \end{cases}$
0x1a	BYTE	2	1	Retrieve single byte from word. $\forall i \in [0..255] : \mu'_s[0]_i \equiv \begin{cases} \mu_s[1]_{(i+8\mu_s[0])} & \text{if } i < 8 \wedge \mu_s[0] < 32 \\ 0 & \text{otherwise} \end{cases}$ For Nth byte, we count from the left (i.e. N=0 would be the most significant in big endian).

20s: SHA3

Value	Mnemonic	δ	α	Description
0x20	SHA3	2	1	Compute Keccak-256 hash. $\mu'_s[0] \equiv \text{Keccak}(\mu_m[\mu_s[0] \dots (\mu_s[0] + \mu_s[1] - 1)])$ $\mu'_i \equiv M(\mu_i, \mu_s[0], \mu_s[1])$

30s: Environmental Information

Value	Mnemonic	δ	α	Description
0x30	ADDRESS	0	1	Get address of currently executing account. $\mu'_s[0] \equiv I_a$
0x31	BALANCE	1	1	Get balance of the given account. $\mu'_s[0] \equiv \begin{cases} \sigma[\mu_s[0]]_b & \text{if } \sigma[\mu_s[0] \bmod 2^{160}] \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$
0x32	ORIGIN	0	1	Get execution origination address. $\mu'_s[0] \equiv I_o$ This is the sender of original transaction; it is never an account with non-empty associated code.
0x33	CALLER	0	1	Get caller address. $\mu'_s[0] \equiv I_s$ This is the address of the account that is directly responsible for this execution.
0x34	CALLVALUE	0	1	Get deposited value by the instruction/transaction responsible for this execution. $\mu'_s[0] \equiv I_v$
0x35	CALLDATALOAD	1	1	Get input data of current environment. $\mu'_s[0] \equiv I_d[\mu_s[0] \dots (\mu_s[0] + 31)]$ with $I_d[x] = 0$ if $x \geq \parallel I_d \parallel$ This pertains to the input data passed with the message call instruction or transaction.
0x36	CALLDATASIZE	0	1	Get size of input data in current environment. $\mu'_s[0] \equiv \parallel I_d \parallel$ This pertains to the input data passed with the message call instruction or transaction.
0x37	CALLDATACOPY	3	0	Copy input data in current environment to memory. $\forall_{i \in \{0 \dots \mu_s[2]-1\}} \mu'_m[\mu_s[0] + i] \equiv \begin{cases} I_d[\mu_s[1] + i] & \text{if } \mu_s[1] + i < \parallel I_d \parallel \\ 0 & \text{otherwise} \end{cases}$ $\mu'_i \equiv M(\mu_i, \mu_s[0], \mu_s[2])$ This pertains to the input data passed with the message call instruction or transaction.
0x38	CODESIZE	0	1	Get size of code running in current environment. $\mu'_s[0] \equiv \parallel I_b \parallel$
0x39	CODECOPY	3	0	Copy code running in current environment to memory. $\forall_{i \in \{0 \dots \mu_s[2]-1\}} \mu'_m[\mu_s[0] + i] \equiv \begin{cases} I_b[\mu_s[1] + i] & \text{if } \mu_s[1] + i < \parallel I_b \parallel \\ STOP & \text{otherwise} \end{cases}$ $\mu'_i \equiv M(\mu_i, \mu_s[0], \mu_s[1])$
0x3a	GASPRICE	0	1	Get price of gas in current environment. $\mu'_s[0] \equiv I_p$ This is gas price specified by the originating transaction.
0x3b	EXTCODESIZE	1	1	Get size of an account's code. $\mu'_s[0] \equiv \parallel \sigma[\mu_s[0] \bmod 2^{160}]_c \parallel$
0x3c	EXTCODECOPY	4	0	Copy an account's code to memory. $\forall_{i \in \{0 \dots \mu_s[3]-1\}} \mu'_m[\mu_s[1] + i] \equiv \begin{cases} c[\mu_s[2] + i] & \text{if } \mu_s[2] + i < \parallel c \parallel \\ STOP & \text{otherwise} \end{cases}$ where $c \equiv \sigma[\mu_s[0] \bmod 2^{160}]_c$ $\mu'_i \equiv M(\mu_i, \mu_s[1], \mu_s[3])$

40s: Block Information

Value	Mnemonic	δ	α	Description
0x40	BLOCKHASH	1	1	Get the hash of one of the 256 most recent complete blocks. $\mu'_s[0] \equiv P(I_{H_p}, \mu_s[0], 0)$ where P is the hash of a block of a particular number, up to a maximum age. 0 is left on the stack if the looked for block number is greater than the current block number or more than 256 blocks behind the current block. $P(h, n, a) \equiv \begin{cases} 0 & \text{if } n > H_i \vee a = 256 \vee h = 0 \\ h & \text{if } n = H_i \\ P(H_p, n, a + 1) & \text{otherwise} \end{cases}$ and we assert the header H can be determined as its hash is the parent hash in the block following it.
0x41	COINBASE	0	1	Get the block's beneficiary address. $\mu'_s[0] \equiv I_{H_c}$
0x42	TIMESTAMP	0	1	Get the block's timestamp. $\mu'_s[0] \equiv I_{H_s}$
0x43	NUMBER	0	1	Get the block's number. $\mu'_s[0] \equiv I_{H_i}$
0x44	DIFFICULTY	0	1	Get the block's difficulty. $\mu'_s[0] \equiv I_{H_d}$
0x45	GASLIMIT	0	1	Get the block's gas limit. $\mu'_s[0] \equiv I_{H_l}$

50s: Stack, Memory, Storage and Flow Operations

Value	Mnemonic	δ	α	Description
0x50	POP	1	0	Remove item from stack.
0x51	MLOAD	1	1	Load word from memory. $\mu'_m[\mu_s[0] \dots (\mu_s[0] + 31)] \equiv \mu_m[\mu_s[0] \dots (\mu_s[0] + 31)]$ $\mu'_i \equiv \max(\mu_i, \lceil (\mu_s[0] + 32) \div 32 \rceil)$ The addition in the calculation of μ'_i is not subject to the 2^{256} modulo.
0x52	MSTORE	2	0	Save word to memory. $\mu'_m[\mu_s[0] \dots (\mu_s[0] + 31)] \equiv \mu_s[1]$ $\mu'_i \equiv \max(\mu_i, \lceil (\mu_s[0] + 32) \div 32 \rceil)$ The addition in the calculation of μ'_i is not subject to the 2^{256} modulo.
0x53	MSTORE8	2	0	Save byte to memory. $\mu'_m[\mu_s[0]] \equiv (\mu_s[1] \bmod 256)$ $\mu'_i \equiv \max(\mu_i, \lceil (\mu_s[0] + 1) \div 32 \rceil)$ The addition in the calculation of μ'_i is not subject to the 2^{256} modulo.
0x54	SLOAD	1	1	Load word from storage. $\mu'_s[0] \equiv \sigma[I_a]_s[\mu_s[0]]$
0x55	SSTORE	2	0	Save word to storage. $\sigma'[I_a]_s[\mu_s[0]] \equiv \mu_s[1]$ $C_{\text{SSTORE}}(\sigma, \mu) \equiv \begin{cases} G_{\text{sset}} & \text{if } \mu_s[1] \neq 0 \wedge \sigma[I_a]_s[\mu_s[0]] = 0 \\ G_{\text{sreset}} & \text{otherwise} \end{cases}$ $A'_r \equiv A_r + \begin{cases} R_{\text{sclear}} & \text{if } \mu_s[1] = 0 \wedge \sigma[I_a]_s[\mu_s[0]] \neq 0 \\ 0 & \text{otherwise} \end{cases}$
0x56	JUMP	1	0	Alter the program counter. $J_{\text{JUMP}}(\mu) \equiv \mu_s[0]$ This has the effect of writing said value to μ_{pc} . See section 9.
0x57	JUMPI	2	0	Conditionally alter the program counter. $J_{\text{JUMPI}}(\mu) \equiv \begin{cases} \mu_s[0] & \text{if } \mu_s[1] \neq 0 \\ \mu_{pc} + 1 & \text{otherwise} \end{cases}$ This has the effect of writing said value to μ_{pc} . See section 9.
0x58	PC	0	1	Get the value of the program counter <i>prior</i> to the increment corresponding to this instruction. $\mu'_s[0] \equiv \mu_{pc}$
0x59	MSIZE	0	1	Get the size of active memory in bytes. $\mu'_s[0] \equiv 32\mu_i$
0x5a	GAS	0	1	Get the amount of available gas, including the corresponding reduction for the cost of this instruction. $\mu'_s[0] \equiv \mu_g$
0x5b	JUMPDEST	0	0	Mark a valid destination for jumps. This operation has no effect on machine state during execution.

60s & 70s: Push Operations

Value	Mnemonic	δ	α	Description
0x60	PUSH1	0	1	Place 1 byte item on stack. $\mu'_s[0] \equiv c(\mu_{pc} + 1)$ where $c(x) \equiv \begin{cases} I_b[x] & \text{if } x < \ I_b\ \\ 0 & \text{otherwise} \end{cases}$ The bytes are read in line from the program code's bytes array. The function c ensures the bytes default to zero if they extend past the limits. The byte is right-aligned (takes the lowest significant place in big endian).
0x61	PUSH2	0	1	Place 2-byte item on stack. $\mu'_s[0] \equiv c((\mu_{pc} + 1) \dots (\mu_{pc} + 2))$ with $c(\mathbf{x}) \equiv (c(\mathbf{x}_0), \dots, c(\mathbf{x}_{\ \mathbf{x}\ -1}))$ with c as defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).
:	:	:	:	:
0x7f	PUSH32	0	1	Place 32-byte (full word) item on stack. $\mu'_s[0] \equiv c((\mu_{pc} + 1) \dots (\mu_{pc} + 32))$ where c is defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).

80s: Duplication Operations

Value	Mnemonic	δ	α	Description
0x80	DUP1	1	2	Duplicate 1st stack item. $\mu'_s[0] \equiv \mu_s[0]$
0x81	DUP2	2	3	Duplicate 2nd stack item. $\mu'_s[0] \equiv \mu_s[1]$
:	:	:	:	:
0x8f	DUP16	16	17	Duplicate 16th stack item. $\mu'_s[0] \equiv \mu_s[15]$

90s: Exchange Operations

Value	Mnemonic	δ	α	Description
0x90	SWAP1	2	2	Exchange 1st and 2nd stack items. $\mu'_s[0] \equiv \mu_s[1]$ $\mu'_s[1] \equiv \mu_s[0]$
0x91	SWAP2	3	3	Exchange 1st and 3rd stack items. $\mu'_s[0] \equiv \mu_s[2]$ $\mu'_s[2] \equiv \mu_s[0]$
:	:	:	:	:
0x9f	SWAP16	17	17	Exchange 1st and 17th stack items. $\mu'_s[0] \equiv \mu_s[16]$ $\mu'_s[16] \equiv \mu_s[0]$

a0s: Logging Operations

For all logging operations, the state change is to append an additional log entry on to the substate's log series:
 $A'_1 \equiv A_1 \cdot (I_a, t, \mu_m[\mu_s[0] \dots (\mu_s[0] + \mu_s[1] - 1)])$

The entry's topic series, t , differs accordingly:

Value	Mnemonic	δ	α	Description
0xa0	LOG0	2	0	Append log record with no topics. $t \equiv ()$
0xa1	LOG1	3	0	Append log record with one topic. $t \equiv (\mu_s[2])$
:	:	:	:	:
0xa4	LOG4	6	0	Append log record with four topics. $t \equiv (\mu_s[2], \mu_s[3], \mu_s[4], \mu_s[5])$

f0s: System operations

Value	Mnemonic	δ	α	Description
0xf0	CREATE	3	1	<p>Create a new account with associated code.</p> $\mathbf{i} \equiv \mu_m[\mu_s[1] \dots (\mu_s[1] + \mu_s[2] - 1)]$ $(\sigma', \mu'_g, A^+) \equiv \begin{cases} \Lambda(\sigma^*, I_a, I_o, L(\mu_g), I_p, \mu_s[0], \mathbf{i}, I_e + 1) & \text{if } \mu_s[0] \leq \sigma[I_a]_b \wedge I_e < 1024 \\ (\sigma, \mu_g, \emptyset) & \text{otherwise} \end{cases}$ $\sigma^* \equiv \sigma$ except $\sigma^*[I_a]_n = \sigma[I_a]_n + 1$ $A' \equiv A \uplus A^+$ which implies: $A'_s \equiv A_s \cup A_s^+ \wedge A'_l \equiv A_l \cdot A_l^+ \wedge A'_r \equiv A_r + A_r^+$ $\mu'_s[0] \equiv x$ where $x = 0$ if the code execution for this operation failed due to an exceptional halting $Z(\sigma^*, \mu, I) = \top$ or $I_e = 1024$ (the maximum call depth limit is reached) or $\mu_s[0] > \sigma[I_a]_b$ (balance of the caller is too low to fulfil the value transfer); and otherwise $x = A(I_a, \sigma[I_a]_n)$, the address of the newly created account, otherwise.
				$\mu'_i \equiv M(\mu_i, \mu_s[1], \mu_s[2])$ Thus the operand order is: value, input offset, input size.
0xf1	CALL	7	1	<p>Message-call into an account.</p> $\mathbf{i} \equiv \mu_m[\mu_s[3] \dots (\mu_s[3] + \mu_s[4] - 1)]$ $(\sigma', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\sigma, I_a, I_o, t, t, & \text{if } \mu_s[2] \leq \sigma[I_a]_b \wedge \\ C_{CALLGAS}(\mu), I_p, \mu_s[2], \mu_s[2], \mathbf{i}, I_e + 1) & I_e < 1024 \\ (\sigma, g, \emptyset, \mathbf{o}) & \text{otherwise} \end{cases}$ $n \equiv \min(\{\mu_s[6], \mathbf{o} \})$ $\mu'_m[\mu_s[5] \dots (\mu_s[5] + n - 1)] = \mathbf{o}[0 \dots (n - 1)]$ $\mu'_g \equiv \mu_g + g'$ $\mu'_s[0] \equiv x$ $A' \equiv A \uplus A^+$ $t \equiv \mu_s[1] \bmod 2^{160}$ where $x = 0$ if the code execution for this operation failed due to an exceptional halting $Z(\sigma, \mu, I) = \top$ or if $\mu_s[2] > \sigma[I_a]_b$ (not enough funds) or $I_e = 1024$ (call depth limit reached); $x = 1$ otherwise.
				$\mu'_i \equiv M(M(\mu_i, \mu_s[3], \mu_s[4]), \mu_s[5], \mu_s[6])$ Thus the operand order is: gas, to, value, in offset, in size, out offset, out size.
				$C_{CALL}(\sigma, \mu) \equiv C_{GASCAP}(\sigma, \mu) + C_{EXTRA}(\sigma, \mu)$ $C_{CALLGAS}(\sigma, \mu) \equiv \begin{cases} C_{GASCAP}(\sigma, \mu) + G_{callstipend} & \text{if } \mu_s[2] \neq 0 \\ C_{GASCAP}(\sigma, \mu) & \text{otherwise} \end{cases}$ $C_{GASCAP}(\sigma, \mu) \equiv \begin{cases} \min\{L(\mu_g - C_{EXTRA}(\sigma, \mu)), \mu_s[0]\} & \text{if } \mu_g \geq C_{EXTRA}(\sigma, \mu) \\ \mu_s[0] & \text{otherwise} \end{cases}$ $C_{EXTRA}(\sigma, \mu) \equiv G_{call} + C_{XFER}(\mu) + C_{NEW}(\sigma, \mu)$ $C_{XFER}(\mu) \equiv \begin{cases} G_{callvalue} & \text{if } \mu_s[2] \neq 0 \\ 0 & \text{otherwise} \end{cases}$ $C_{NEW}(\sigma, \mu) \equiv \begin{cases} G_{newaccount} & \text{if } \sigma[\mu_s[1]] \bmod 2^{160} = \emptyset \\ 0 & \text{otherwise} \end{cases}$
0xf2	CALLCODE	7	1	<p>Message-call into this account with an alternative account's code.</p> Exactly equivalent to CALL except: $(\sigma', g', A^+, \mathbf{o}) \equiv \begin{cases} \Theta(\sigma^*, I_a, I_o, I_a, t, & \text{if } \mu_s[2] \leq \sigma[I_a]_b \wedge \\ C_{CALLGAS}(\mu), I_p, \mu_s[2], \mu_s[2], \mathbf{i}, I_e + 1) & I_e < 1024 \\ (\sigma, g, \emptyset, \mathbf{o}) & \text{otherwise} \end{cases}$ Note the change in the fourth parameter to the call Θ from the 2nd stack value $\mu_s[1]$ (as in CALL) to the present address I_a . This means that the recipient is in fact the same account as at present, simply that the code is overwritten.
0xf3	RETURN	2	0	<p>Halt execution returning output data.</p> $H_{RETURN}(\mu) \equiv \mu_m[\mu_s[0] \dots (\mu_s[0] + \mu_s[1] - 1)]$ This has the effect of halting the execution at this point with output defined. See section 9.
				$\mu'_i \equiv M(\mu_i, \mu_s[0], \mu_s[1])$

0xf4	DELEGATECALL	6	1	Message-call into this account with an alternative account's code, but persisting the current values for <i>sender</i> and <i>value</i> . Compared with CALL, DELEGATECALL takes one fewer arguments. The omitted argument is $\mu_s[2]$. As a result, $\mu_s[3]$, $\mu_s[4]$, $\mu_s[5]$ and $\mu_s[6]$ in the definition of CALL should respectively be replaced with $\mu_s[2]$, $\mu_s[3]$, $\mu_s[4]$ and $\mu_s[5]$. Otherwise exactly equivalent to CALL except:
				$(\sigma', g', A^+, o) \equiv \begin{cases} \Theta(\sigma^*, I_s, I_o, I_a, t, \\ \mu_s[0], I_p, 0, I_v, i, I_e + 1) & \text{if } I_v \leq \sigma[I_a]_b \wedge I_e < 1024 \\ (\sigma, g, \emptyset, o) & \text{otherwise} \end{cases}$
				Note the changes (in addition to that of the fourth parameter) to the second and ninth parameters to the call Θ . This means that the recipient is in fact the same account as at present, simply that the code is overwritten and the context is almost entirely identical.
0xff	SUICIDE	1	0	Halt execution and register account for later deletion. $A'_s \equiv A_s \cup \{I_a\}$ $\sigma'[\mu_s[0]] \bmod 2^{160}]_b \equiv \sigma[\mu_s[0]] \bmod 2^{160}]_b + \sigma[I_a]_b$ $\sigma'[I_a]_b \equiv 0$ $A'_r \equiv A_r + \begin{cases} R_{suicide} & \text{if } I_a \notin A_s \\ 0 & \text{otherwise} \end{cases}$ $C_{SUICIDE}(\sigma, \mu) \equiv G_{suicide} + \begin{cases} G_{newaccount} & \text{if } \sigma[\mu_s[1]] \bmod 2^{160}] = \emptyset \\ 0 & \text{otherwise} \end{cases}$

APPENDIX I. GENESIS BLOCK

The genesis block is 15 items, and is specified thus:

$$(226) \quad ((0_{256}, \text{KEC}(\text{RLP}(())), 0_{160}, \text{stateRoot}, 0, 0, 0_{2048}, 2^{17}, 0, 0, 3141592, \text{time}, 0, 0_{256}, \text{KEC}((42))), (), ())$$

Where 0_{256} refers to the parent hash, a 256-bit hash which is all zeroes; 0_{160} refers to the beneficiary address, a 160-bit hash which is all zeroes; 0_{2048} refers to the log bloom, 2048-bit of all zeros; 2^{17} refers to the difficulty; the transaction trie root, receipt trie root, gas used, block number and extradata are both 0, being equivalent to the empty byte array. The sequences of both ommers and transactions are empty and represented by $()$. $\text{KEC}((42))$ refers to the Keccak hash of a byte array of length one whose first and only byte is of value 42, used for the nonce. $\text{KEC}(\text{RLP}(()))$ value refers to the hash of the ommer lists in RLP, both empty lists.

The proof-of-concept series include a development premine, making the state root hash some value *stateRoot*. Also *time* will be set to the intial timestamp of the genesis block. The latest documentation should be consulted for those values.

APPENDIX J. ETHASH

J.1. **Definitions.** We employ the following definitions:

Name	Value	Description
$J_{wordbytes}$	4	Bytes in word.
$J_{datasetinit}$	2^{30}	Bytes in dataset at genesis.
$J_{datasetgrowth}$	2^{23}	Dataset growth per epoch.
$J_{cacheinit}$	2^{24}	Bytes in cache at genesis.
$J_{cachegrowth}$	2^{17}	Cache growth per epoch.
J_{epoch}	30000	Blocks per epoch.
$J_{mixbytes}$	128	mix length in bytes.
$J_{hashbytes}$	64	Hash length in bytes.
$J_{parents}$	256	Number of parents of each dataset element.
$J_{cacherounds}$	3	Number of rounds in cache production.
$J_{accesses}$	64	Number of accesses in hashimoto loop.

J.2. **Size of dataset and cache.** The size for Ethash's cache $c \in \mathbb{B}$ and dataset $d \in \mathbb{B}$ depend on the epoch, which in turn depends on the block number.

$$(227) \quad E_{epoch}(H_i) = \left\lceil \frac{H_i}{J_{epoch}} \right\rceil$$

The size of the dataset growth by $J_{datasetgrowth}$ bytes, and the size of the cache by $J_{cachegrowth}$ bytes, every epoch. In order to avoid regularity leading to cyclic behavior, the size must be a prime number. Therefore the size is reduced by

a multiple of $J_{mixbytes}$, for the dataset, and $J_{hashbytes}$ for the cache. Let $d_{size} = \|\mathbf{d}\|$ be the size of the dataset. Which is calculated using

$$(228) \quad d_{size} = E_{prime}(J_{datasetinit} + J_{datasetgrowth} \cdot E_{epoch} - J_{mixbytes}, J_{mixbytes})$$

The size of the cache, c_{size} , is calculated using

$$(229) \quad c_{size} = E_{prime}(J_{cacheinit} + J_{cachegrowth} \cdot E_{epoch} - J_{hashbytes}, J_{hashbytes})$$

$$(230) \quad E_{prime}(x, y) = \begin{cases} x & \text{if } x/y \in \mathbb{P} \\ E_{prime}(x - 1 \cdot y, y) & \text{otherwise} \end{cases}$$

J.3. Dataset generation. In order to generate the dataset we need the cache \mathbf{c} , which is an array of bytes. It depends on the cache size c_{size} and the seed hash $\mathbf{s} \in \mathbb{B}_{32}$.

J.3.1. Seed hash. The seed hash is different for every epoch. For the first epoch it is the Keccak-256 hash of a series of 32 bytes of zeros. For every other epoch it is always the Keccak-256 hash of the previous seed hash:

$$(231) \quad \mathbf{s} = C_{seedhash}(H_i)$$

$$(232) \quad C_{seedhash}(H_i) = \begin{cases} \text{KEC}(\mathbf{0}_{32}) & \text{if } E_{epoch}(H_i) = 0 \\ \text{KEC}(C_{seedhash}(H_i - J_{epoch})) & \text{otherwise} \end{cases}$$

With $\mathbf{0}_{32}$ being 32 bytes of zeros.

J.3.2. Cache. The cache production process involves using the seed hash to first sequentially filling up c_{size} bytes of memory, then performing $J_{cacherounds}$ passes of the RandMemoHash algorithm created by Lerner [2014]. The initial cache \mathbf{c}' , being an array of arrays of single bytes, will be constructed as follows.

We define the array \mathbf{c}_i , consisting of 64 single bytes, as the i th element of the initial cache:

$$(233) \quad \mathbf{c}_i = \begin{cases} \text{KEC512}(\mathbf{s}) & \text{if } i = 0 \\ \text{KEC512}(\mathbf{c}_{i-1}) & \text{otherwise} \end{cases}$$

Therefore \mathbf{c}' can be defined as

$$(234) \quad \mathbf{c}'[i] = \mathbf{c}_i \quad \forall \quad i < n$$

$$(235) \quad n = \left\lfloor \frac{c_{size}}{J_{hashbytes}} \right\rfloor$$

The cache is calculated by performing $J_{cacherounds}$ rounds of the RandMemoHash algorithm to the initial cache \mathbf{c}' :

$$(236) \quad \mathbf{c} = E_{cacherounds}(\mathbf{c}', J_{cacherounds})$$

$$(237) \quad E_{cacherounds}(\mathbf{x}, y) = \begin{cases} \mathbf{x} & \text{if } y = 0 \\ E_{RMH}(\mathbf{x}) & \text{if } y = 1 \\ E_{cacherounds}(E_{RMH}(\mathbf{x}), y - 1) & \text{otherwise} \end{cases}$$

Where a single round modifies each subset of the cache as follows:

$$(238) \quad E_{RMH}(\mathbf{x}) = (E_{rmh}(\mathbf{x}, 0), E_{rmh}(\mathbf{x}, 1), \dots, E_{rmh}(\mathbf{x}, n - 1))$$

$$(239) \quad E_{rmh}(\mathbf{x}, i) = \text{KEC512}(\mathbf{x}'[(i - 1 + n) \bmod n] \oplus \mathbf{x}'[\mathbf{x}'[i][0] \bmod n]) \\ \text{with } \mathbf{x}' = \mathbf{x} \text{ except } \mathbf{x}'[j] = E_{rmh}(\mathbf{x}, j) \quad \forall \quad j < i$$

J.3.3. Full dataset calculation. Essentially, we combine data from $J_{parents}$ pseudorandomly selected cache nodes, and hash that to compute the dataset. The entire dataset is then generated by a number of items, each $J_{hashbytes}$ bytes in size:

$$(240) \quad \mathbf{d}[i] = E_{datasetitem}(\mathbf{c}, i) \quad \forall \quad i < \left\lfloor \frac{d_{size}}{J_{hashbytes}} \right\rfloor$$

In order to calculate the single item we use an algorithm inspired by the FNV hash (Glenn Fowler [1991]) in some cases as a non-associative substitute for XOR.

$$(241) \quad E_{FNV}(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot (0x01000193 \oplus \mathbf{y})) \bmod 2^{32}$$

The single item of the dataset can now be calculated as:

$$(242) \quad E_{datasetitem}(\mathbf{c}, i) = E_{parents}(\mathbf{c}, i, -1, \emptyset)$$

$$(243) \quad E_{parents}(\mathbf{c}, i, p, \mathbf{m}) = \begin{cases} E_{parents}(\mathbf{c}, i, p + 1, E_{mix}(\mathbf{m}, \mathbf{c}, i, p + 1)) & \text{if } p < J_{parents} - 2 \\ E_{mix}(\mathbf{m}, \mathbf{c}, i, p + 1) & \text{otherwise} \end{cases}$$

$$(244) \quad E_{mix}(\mathbf{m}, \mathbf{c}, i, p) = \begin{cases} \text{KEC512}(\mathbf{c}[i \bmod c_{size}] \oplus i) & \text{if } p = 0 \\ E_{FNV}(\mathbf{m}, \mathbf{c}[E_{FNV}(i \oplus p, \mathbf{m}[p \bmod \lfloor J_{hashbytes}/J_{wordbytes} \rfloor]) \bmod c_{size}]) & \text{otherwise} \end{cases}$$

J.4. Proof-of-work function. Essentially, we maintain a "mix" $J_{mixbytes}$ bytes wide, and repeatedly sequentially fetch $J_{mixbytes}$ bytes from the full dataset and use the E_{FNV} function to combine it with the mix. $J_{mixbytes}$ bytes of sequential access are used so that each round of the algorithm always fetches a full page from RAM, minimizing translation lookaside buffer misses which ASICs would theoretically be able to avoid.

If the output of this algorithm is below the desired target, then the nonce is valid. Note that the extra application of KEC at the end ensures that there exists an intermediate nonce which can be provided to prove that at least a small amount of work was done; this quick outer PoW verification can be used for anti-DDoS purposes. It also serves to provide statistical assurance that the result is an unbiased, 256 bit number.

The PoW-function returns an array with the compressed mix as its first item and the Keccak-256 hash of the concatenation of the compressed mix with the seed hash as the second item:

$$(245) \quad \text{PoW}(H_{\mathcal{H}}, H_n, \mathbf{d}) = \{\mathbf{m}_c(\text{KEC}(\text{RLP}(L_H(H_{\mathcal{H}}))), H_n, \mathbf{d}), \text{KEC}(\mathbf{s}_h(\text{KEC}(\text{RLP}(L_H(H_{\mathcal{H}}))), H_n) + \mathbf{m}_c(\text{KEC}(\text{RLP}(L_H(H_{\mathcal{H}}))), H_n, \mathbf{d}))\}$$

With $H_{\mathcal{H}}$ being the hash of the header without the nonce. The compressed mix \mathbf{m}_c is obtained as follows:

$$(246) \quad \mathbf{m}_c(\mathbf{h}, \mathbf{n}, \mathbf{d}) = E_{compress}(E_{accesses}(\mathbf{d}, \sum_{i=0}^{n_{mix}} \mathbf{s}_h(\mathbf{h}, \mathbf{n}), \mathbf{s}_h(\mathbf{h}, \mathbf{n}), -1), -4)$$

The seed hash being:

$$(247) \quad \mathbf{s}_h(\mathbf{h}, \mathbf{n}) = \text{KEC512}(\mathbf{h} + E_{revert}(\mathbf{n}))$$

$E_{revert}(\mathbf{n})$ returns the reverted bytes sequence of the nonce \mathbf{n} :

$$(248) \quad E_{revert}(\mathbf{n})[i] = \mathbf{n}[\|\mathbf{n}\| - i]$$

We note that the "+"-operator between two byte sequences results in the concatenation of both sequences.

The dataset \mathbf{d} is obtained as described in section J.3.3.

The number of replicated sequences in the mix is:

$$(249) \quad n_{mix} = \left\lfloor \frac{J_{mixbytes}}{J_{hashbytes}} \right\rfloor$$

In order to add random dataset nodes to the mix, the $E_{accesses}$ function is used:

$$(250) \quad E_{accesses}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i) = \begin{cases} E_{mixdataset}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i) & \text{if } i = J_{accesses} - 2 \\ E_{accesses}(E_{mixdataset}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i), \mathbf{s}, i + 1) & \text{otherwise} \end{cases}$$

$$(251) \quad E_{mixdataset}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i) = E_{FNV}(\mathbf{m}, E_{newdata}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i))$$

$E_{newdata}$ returns an array with n_{mix} elements:

$$(252) \quad E_{newdata}(\mathbf{d}, \mathbf{m}, \mathbf{s}, i)[j] = \mathbf{d}[E_{FNV}(i \oplus \mathbf{s}[0], \mathbf{m}[i \bmod \left\lfloor \frac{J_{mixbytes}}{J_{wordbytes}} \right\rfloor]) \bmod \left\lfloor \frac{d_{size}/J_{hashbytes}}{n_{mix}} \right\rfloor \cdot n_{mix} + j] \quad \forall j < n_{mix}$$

The mix is compressed as follows:

$$(253) \quad E_{compress}(\mathbf{m}, i) = \begin{cases} \mathbf{m} & \text{if } i \geq \|\mathbf{m}\| - 8 \\ E_{compress}(E_{FNV}(E_{FNV}(E_{FNV}(\mathbf{m}[i+4], \mathbf{m}[i+5]), \mathbf{m}[i+6]), \mathbf{m}[i+7]), i+8) & \text{otherwise} \end{cases}$$

The Ripple Protocol Consensus Algorithm

David Schwartz
david@ripple.com

Noah Youngs
nyoungs@nyu.edu

Arthur Britto
arthur@ripple.com

Abstract

While several consensus algorithms exist for the Byzantine Generals Problem, specifically as it pertains to distributed payment systems, many suffer from high latency induced by the requirement that all nodes within the network communicate synchronously. In this work, we present a novel consensus algorithm that circumvents this requirement by utilizing collectively-trusted subnetworks within the larger network. We show that the “trust” required of these subnetworks is in fact minimal and can be further reduced with principled choice of the member nodes. In addition, we show that minimal connectivity is required to maintain agreement throughout the whole network. The result is a low-latency consensus algorithm which still maintains robustness in the face of Byzantine failures. We present this algorithm in its embodiment in the Ripple Protocol.

Contents

1	Introduction	1
2	Definitions, Formalization and Previous Work	2
2.1	Ripple Protocol Components	2
2.2	Formalization	3
2.3	Existing Consensus Algorithms	3
2.4	Formal Consensus Goals	3
3	Ripple Consensus Algorithm	4
3.1	Definition	4
3.2	Correctness	4
3.3	Agreement	5
3.4	Utility	5
	Convergence • Heuristics and Procedures	
4	Simulation Code	7
5	Discussion	7
6	Acknowledgments	8
	References	8

1. Introduction

Interest and research in distributed consensus systems has increased markedly in recent years, with a central focus being on distributed payment networks. Such networks allow for fast, low-cost transactions which are not controlled by a centralized source. While the economic benefits and drawbacks of such a system are worthy of much research in and of themselves, this work focuses on some of the technical challenges that all distributed payment systems must face. While these problems are varied, we group them into three main categories: correctness, agreement, and utility.

By correctness, we mean that it is necessary for a distributed system to be able to discern the difference between a correct and fraudulent transaction. In traditional fiduciary settings, this is done through trust between institutions and cryptographic signatures that guarantee a transaction is indeed coming from the institution that it claims to be coming from. In distributed systems, however, there is no such trust, as the identity of any and all members in the network may not even be known. Therefore, alternative methods for correctness must be

utilized.

Agreement refers to the problem of maintaining a single global truth in the face of a decentralized accounting system. While similar to the correctness problem, the difference lies in the fact that while a malicious user of the network may be unable to create a fraudulent transaction (defying correctness), it may be able to create multiple correct transactions that are somehow unaware of each other, and thus combine to create a fraudulent act. For example, a malicious user may make two simultaneous purchases, with only enough funds in their account to cover each purchase individually, but not both together. Thus each transaction by itself is correct, but if executed simultaneously in such a way that the distributed network as a whole is unaware of both, a clear problem arises, commonly referred to as the “Double-Spend Problem” [1]. Thus the agreement problem can be summarized as the requirement that only one set of globally recognized transactions exist in the network.

Utility is a slightly more abstract problem, which we define generally as the “usefulness” of a distributed payment system, but which in practice most often simplifies to the latency of the system. A distributed system that is both correct and in agreement but which requires one year to process a transaction, for example, is obviously an inviable payment system. Additional aspects of utility may include the level of computing power required to participate in the correctness and agreement processes or the technical proficiency required of an end user to avoid being defrauded in the network.

Many of these issues have been explored long before the advent of modern distributed computer systems, via a problem known as the “Byzantine Generals Problem” [2]. In this problem, a group of generals each control a portion of an army and must coordinate an attack by sending messengers to each other. Because the generals are in unfamiliar and hostile territory, messengers may fail to reach their destination (just as nodes in a distributed network may fail, or send corrupted data instead of the intended message). An additional aspect of the problem is that some of the generals may be traitors, either individually, or conspiring together, and so messages may arrive which are intended to create a false plan that is doomed to failure for the loyal generals (just as malicious members of a distributed system may attempt to convince the system to accept fraudulent transactions, or multiple versions of the same truthful transaction that would result in a double-spend). Thus

a distributed payment system must be robust both in the face of standard failures, and so-called “Byzantine” failures, which may be coordinated and originate from multiple sources in the network.

In this work, we analyze one particular implementation of a distributed payment system: the Ripple Protocol. We focus on the algorithms utilized to achieve the above goals of correctness, agreement, and utility, and show that all are met (within necessary and predetermined tolerance thresholds, which are well-understood). In addition, we provide code that simulates the consensus process with parameterizable network size, number of malicious users, and message-sending latencies.

2. Definitions, Formalization and Previous Work

We begin by defining the components of the Ripple Protocol. In order to prove correctness, agreement, and utility properties, we first formalize those properties into axioms. These properties, when grouped together, form the notion of *consensus*: the state in which nodes in the network reach correct agreement. We then highlight some previous results relating to consensus algorithms, and finally state the goals of consensus for the Ripple Protocol within our formalization framework.

2.1 Ripple Protocol Components

We begin our description of the ripple network by defining the following terms:

- **Server:** A server is any entity running the Ripple Server software (as opposed to the Ripple Client software which only lets a user send and receive funds), which participates in the consensus process.
- **Ledger:** The ledger is a record of the amount of currency in each user’s account and represents the “ground truth” of the network. The ledger is repeatedly updated with transactions that successfully pass through the consensus process.
- **Last-Closed Ledger:** The last-closed ledger is the most recent ledger that has been ratified by the consensus process and thus represents the current state of the network.
- **Open Ledger:** The open ledger is the current operating status of a node (each node maintains its own open ledger). Transactions initiated by end users of a given server are applied to the open

ledger of that server, but transactions are not considered final until they have passed through the consensus process, at which point the open ledger becomes the last-closed ledger.

- **Unique Node List (UNL):** Each server, s , maintains a unique node list, which is a set of other servers that s queries when determining consensus. Only the votes of the other members of the UNL of s are considered when determining consensus (as opposed to every node on the network). Thus the UNL represents a subset of the network which when taken collectively, is “trusted” by s to not collude in an attempt to defraud the network. Note that this definition of “trust” does not require that each individual member of the UNL be trusted (see section 3.2).
- **Proposer:** Any server can broadcast transactions to be included in the consensus process, and every server attempts to include every valid transaction when a new consensus round starts. During the consensus process, however, only proposals from servers on the UNL of a server s are considered by s .

2.2 Formalization

We use the term *nonfaulty* to refer to nodes in the network that behave honestly and without error. Conversely, a *faulty* node is one which experiences errors which may be honest (due to data corruption, implementation errors, etc.), or malicious (Byzantine errors). We reduce the notion of validating a transaction to a simple binary decision problem: each node must decide from the information it has been given on the value 0 or 1.

As in Attiya, Dolev, and Gill, 1984 [3], we define consensus according to the following three axioms:

1. **(C1):** Every nonfaulty node makes a decision in finite time
2. **(C2):** All nonfaulty nodes reach the same decision value
3. **(C3):** 0 and 1 are both possible values for all non-faulty nodes. (This removes the trivial solution in which all nodes decide 0 or 1 regardless of the information they have been presented).

2.3 Existing Consensus Algorithms

There has been much research done on algorithms that achieve consensus in the face of Byzantine errors. This

previous work has included extensions to cases where all participants in the network are not known ahead of time, where the messages are sent asynchronously (there is no bound on the amount of time an individual node will take to reach a decision), and where there is a delineation between the notion of strong and weak consensus.

One pertinent result of previous work on consensus algorithms is that of Fischer, Lynch, and Patterson, 1985 [4], which proves that in the asynchronous case, non-termination is always a possibility for a consensus algorithm, even with just one faulty process. This introduces the necessity for time-based heuristics, to ensure convergence (or at least repeated iterations of non-convergence). We shall describe these heuristics for the Ripple Protocol in section 3.

The strength of a consensus algorithm is usually measured in terms of the fraction of faulty processes it can tolerate. It is provable that no solution to the Byzantine Generals problem (which already assumes synchronicity, and known participants) can tolerate more than $(n - 1)/3$ byzantine faults, or 33% of the network acting maliciously [2]. This solution does not, however, require verifiable authenticity of the messages delivered between nodes (digital signatures). If a guarantee on the unforgeability of messages is possible, algorithms exist with much higher fault tolerance in the synchronous case.

Several algorithms with greater complexity have been proposed for Byzantine consensus in the asynchronous case. FaB Paxos [5] will tolerate $(n - 1)/5$ Byzantine failures in a network of n nodes, amounting to a tolerance of up to 20% of nodes in the network colluding maliciously. Attiya, Doyev, and Gill [3] introduce a phase algorithm for the asynchronous case, which can tolerate $(n - 1)/4$ failures, or up to 25% of the network. Lastly, Alchieri et al., 2008 [6] present BFT-CUP, which achieves Byzantine consensus in the asynchronous case even with unknown participants, with the maximal bound of a tolerance of $(n - 1)/3$ failures, but with additional restrictions on the connectivity of the underlying network.

2.4 Formal Consensus Goals

Our goal in this work is to show that the consensus algorithm utilized by the Ripple Protocol will achieve consensus at each ledger-close (even if consensus is the trivial consensus of all transactions being rejected), and that the trivial consensus will only be reached with a known probability, even in the face of Byzantine failures.

Since each node in the network only votes on proposals from a trusted set of nodes (the other nodes in its UNL), and since each node may have differing UNLs, we also show that only one consensus will be reached amongst all nodes, regardless of UNL membership. This goal is also referred to as preventing a “fork” in the network: a situation in which two disjoint sets of nodes each reach consensus independently, and two different last-closed ledgers are observed by nodes on each node-set.

Lastly we will show that the Ripple Protocol can achieve these goals in the face of $(n - 1)/5$ failures, which is not the strongest result in the literature, but we will also show that the Ripple Protocol possesses several other desirable features that greatly enhance its utility.

3. Ripple Consensus Algorithm

The Ripple Protocol consensus algorithm (RPCA), is applied every few seconds by all nodes, in order to maintain the correctness and agreement of the network. Once consensus is reached, the current ledger is considered “closed” and becomes the last-closed ledger. Assuming that the consensus algorithm is successful, and that there is no fork in the network, the last-closed ledger maintained by all nodes in the network will be identical.

3.1 Definition

The RPCA proceeds in rounds. In each round:

- Initially, each server takes all valid transactions it has seen prior to the beginning of the consensus round that have not already been applied (these may include new transactions initiated by end-users of the server, transactions held over from a previous consensus process, etc.), and makes them public in the form of a list known as the “candidate set”.
- Each server then amalgamates the candidate sets of all servers on its UNL, and votes on the veracity of all transactions.
- Transactions that receive more than a minimum percentage of “yes” votes are passed on to the next round, if there is one, while transactions that do not receive enough votes will either be discarded, or included in the candidate set for the beginning of the consensus process on the next ledger.
- The final round of consensus requires a minimum percentage of 80% of a server’s UNL agreeing

on a transaction. All transactions that meet this requirement are applied to the ledger, and that ledger is closed, becoming the new last-closed ledger.

3.2 Correctness

In order to achieve correctness, given a maximal amount of Byzantine failures, it must be shown that it is impossible for a fraudulent transaction to be confirmed during consensus, unless the number of faulty nodes exceeds that tolerance. The proof of the correctness of the RPCA then follows directly: since a transaction is only approved if 80% of the UNL of a server agrees with it, as long as 80% of the UNL is honest, no fraudulent transactions will be approved. Thus for a UNL of n nodes in the network, the consensus protocol will maintain correctness so long as:

$$f \leq (n - 1)/5 \quad (1)$$

where f is the number Byzantine failures. In fact, even in the face of $(n - 1)/5 + 1$ Byzantine failures, correctness is still technically maintained. The consensus process will fail, but it will still not be possible to confirm a fraudulent transaction. Indeed it would take $(4n + 1)/5$ Byzantine failures for an incorrect transaction to be confirmed. We call this second bound the bound for *weak* correctness, and the former the bound for *strong* correctness.

It should also be noted that not all “fraudulent” transactions pose a threat, even if confirmed during consensus. Should a user attempt to double-spend funds in two transactions, for example, even if both transactions are confirmed during the consensus process, after the first transaction is applied, the second will fail, as the funds are no longer available. This robustness is due to the fact that transactions are applied deterministically, and that consensus ensures that all nodes in the network are applying the deterministic rules to the same set of transactions.

For a slightly different analysis, let us assume that the probability that any node will decide to collude and join a nefarious cartel is p_c . Then the probability of correctness is given by p^* , where:

$$p^* = \sum_{i=0}^{\lceil (n-1)/5 \rceil} \binom{n}{i} p_c^i (1-p_c)^{n-i} \quad (2)$$

This probability represents the likelihood that the size of the nefarious cartel will remain below the maximal

threshold of Byzantine failures, given p_c . Since this likelihood is a binomial distribution, values of p_c greater than 20% will result in expected cartels of size greater than 20% of the network, thwarting the consensus process. In practice, a UNL is not chosen randomly, but rather with the intent to minimize p_c . Since nodes are not anonymous but rather cryptographically identifiable, selecting a UNL of nodes from a mixture of continents, nations, industries, ideologies, etc. will produce values of p_c much lower than 20%. As an example, the probability of the Anti-Defamation League and the Westboro Baptist Church colluding to defraud the network, is certainly much, much smaller than 20%. Even if the UNL has a relatively large p_c , say 15%, the probability of correctness is extremely high even with only 200 nodes in the UNL: 97.8%.

A graphical representation of how the probability of incorrectness scales as a function of UNL size for differing values of p_c is depicted in Figure 1. Note that here the vertical axis represents the probability of a nefarious cartel thwarting consensus, and thus lower values indicate greater probability of consensus success. As can be seen in the figure, even with a p_c as high as 10%, the probability of consensus being thwarted very quickly becomes negligible as the UNL grows past 100 nodes.

3.3 Agreement

To satisfy the agreement requirement, it must be shown that all nonfaulty nodes reach consensus on the same set of transactions, regardless of their UNLs. Since the UNLs for each server can be different, agreement is not inherently guaranteed by the correctness proof. For example, if there are no restrictions on the membership of the UNL, and the size of the UNL is not larger than $0.2 * n_{total}$ where n_{total} is the number of nodes in the entire network, then a fork is possible. This is illustrated by a simple example (depicted in figure 2): imagine two cliques within the UNL graph, each larger than $0.2 * n_{total}$. By cliques, we mean a set of nodes where each node's UNL is the selfsame set of nodes. Because these two cliques do not share any members, it is possible for each to achieve a correct consensus independently of each other, violating agreement. If the connectivity of the two cliques surpasses $0.2 * n_{total}$, then a fork is no longer possible, as disagreement between the cliques would prevent consensus from being reached at the 80% agreement threshold that is required.

An upper bound on the connectivity required to

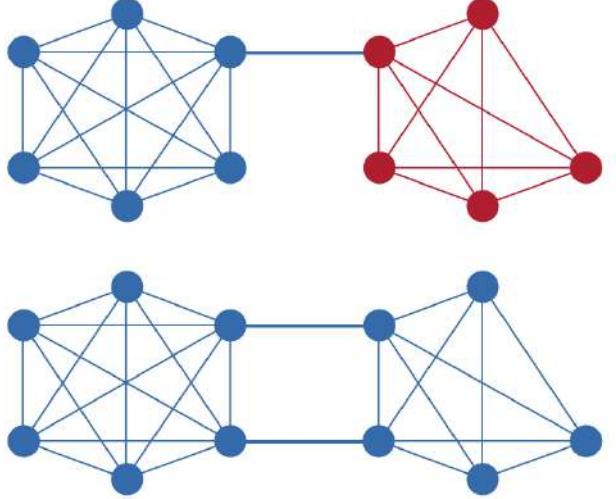


Figure 2. An example of the connectivity required to prevent a fork between two UNL cliques.

prove agreement is given by:

$$|UNL_i \cap UNL_j| \geq \frac{1}{5} \max(|UNL_i|, |UNL_j|) \forall i, j \quad (3)$$

This upper bound assumes a clique-like structure of UNLs, i.e. nodes form sets whose UNLs contain other nodes in those sets. This upper bound guarantees that no two cliques can reach consensus on conflicting transactions, since it becomes impossible to reach the 80% threshold required for consensus. A tighter bound is possible when indirect edges between UNLs are taken into account as well. For example, if the structure of the network is not clique-like, a fork becomes much more difficult to achieve, due to the greater entanglement of the UNLs of all nodes.

It is interesting to note that no assumptions are made about the nature of the intersecting nodes. The intersection of two UNLs may include faulty nodes, but so long as the size of the intersection is larger than the bound required to guarantee agreement, and the total number of faulty nodes is less than the bound required to satisfy strong correctness, then both correctness and agreement will be achieved. That is to say, agreement is dependent solely on the size of the intersection of nodes, not on the size of the intersection of nonfaulty nodes.

3.4 Utility

While many components of utility are subjective, one that is indeed provable is convergence: that the consensus process will terminate in finite time.

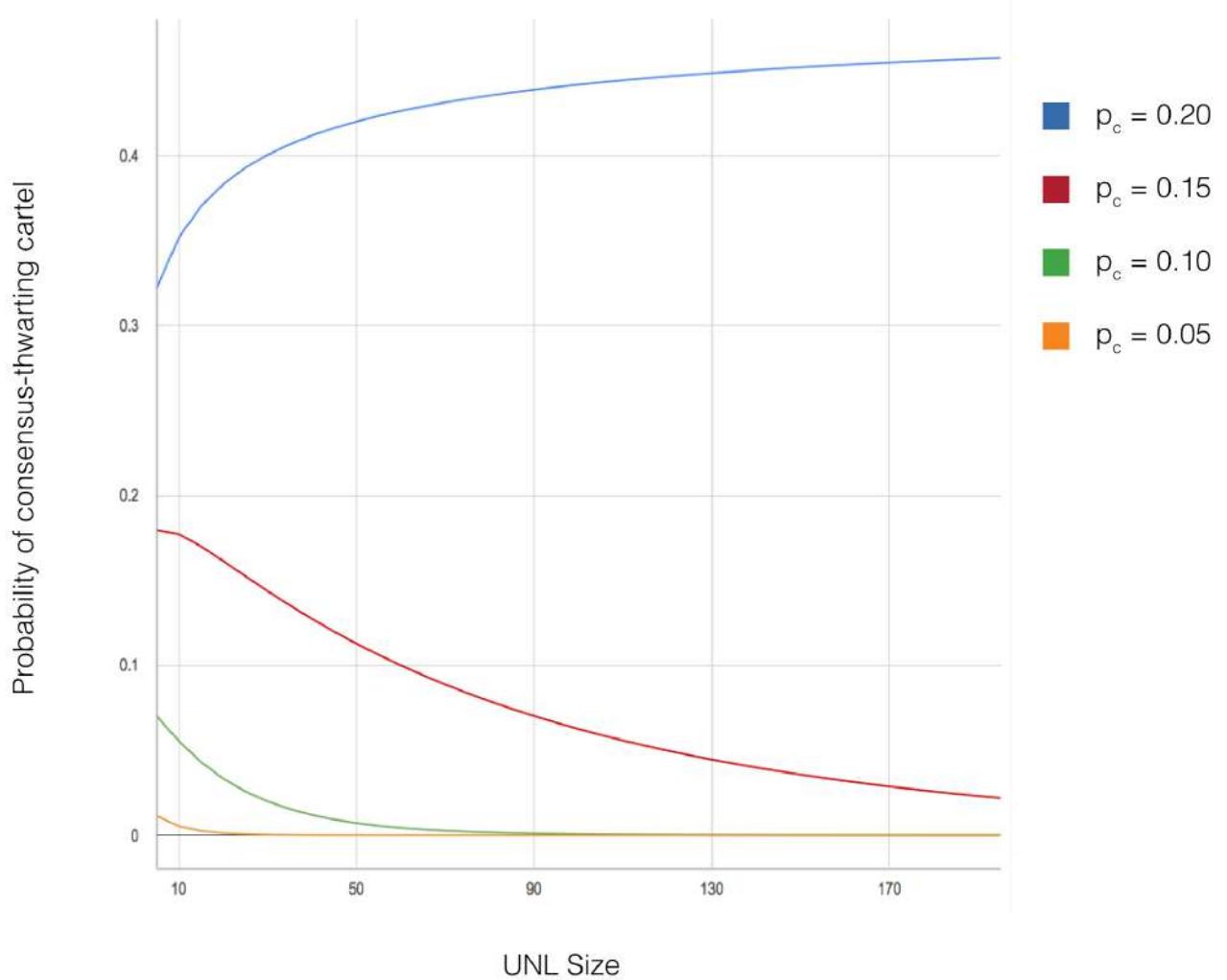


Figure 1. Probability of a nefarious cartel being able to thwart consensus as a function of the size of the UNL, for different values of p_c , the probability that any member of the UNL will decide to collude with others. Here, lower values indicate a higher probability of consensus success.

3.4.1 Convergence

We define convergence as the point in which the RPCA reaches consensus with strong correctness on the ledger, and that ledger then becomes the last-closed ledger. Note that while technically weak correctness still represents convergence of the algorithm, it is only convergence in the trivial case, as proposition **C3** is violated, and no transactions will ever be confirmed. From the results above, we know that strong correctness is always achievable in the face of up to $(n - 1)/5$ Byzantine failures, and that only one consensus will be achieved in the entire network so long as the UNL-connectedness condition is met (Equation 3). All that remains is to show that when both of these conditions are met, consensus is reached in finite time.

Since the consensus algorithm itself is deterministic, and has a preset number of rounds, t , before consensus is terminated, and the current set of transactions are declared approved or not-approved (even if at this point no transactions have more than the 80% required agreement, and the consensus is only the trivial consensus), the limiting factor for the termination of the algorithm is the communication latency between nodes. In order to bound this quantity, the response-time of nodes is monitored, and nodes whose latency grows larger than a preset bound b are removed from all UNLs. While this guarantees that consensus will terminate with an upper bound of tb , it is important to note that the bounds described for correctness and agreement above must be met by the *final* UNL, after all nodes that will be

dropped have been dropped. If the conditions hold for the initial UNLs for all nodes, but then some nodes are dropped from the network due to latency, the correctness and agreement guarantees do not automatically hold but must be satisfied by the new set of UNLs.

3.4.2 Heuristics and Procedures

As mentioned above, a latency bound heuristic is enforced on all nodes in the Ripple Network to guarantee that the consensus algorithm will converge. In addition, there are a few other heuristics and procedures that provide utility to the RPCA.

- There is a mandatory 2 second window for all nodes to propose their initial candidate sets in each round of consensus. While this does introduce a lower bound of 2 seconds to each consensus round, it also guarantees that all nodes with reasonable latency will have the ability to participate in the consensus process.
- As the votes are recorded in the ledger for each round of consensus, nodes can be flagged and removed from the network for some common, easily-identifiable malicious behaviors. These include nodes that vote “No” on every transaction, and nodes that consistently propose transactions which are not validated by consensus.
- A curated default UNL is provided to all users, which is chosen to minimize p_c , described in section 3.2. While users can and should select their own UNLs, this default list of nodes guarantees that even naive users will participate in a consensus process that achieves correctness and agreement with extremely high probability.
- A network split detection algorithm is also employed to avoid a fork in the network. While the consensus algorithm certifies that the transactions on the last-closed ledger are correct, it does not prohibit the possibility of more than one last-closed ledger existing on different subsections of the network with poor connectivity. To try and identify if such a split has occurred, each node monitors the size of the active members of its UNL. If this size suddenly drops below a preset threshold, it is possible that a split has occurred. In order to prevent a false positive in the case where a large section of a UNL has temporary latency, nodes are allowed to publish a “partial

validation”, in which they do not process or vote on transactions, but declare that are still participating in the consensus process, as opposed to a different consensus process on a disconnected subnetwork.

- While it would be possible to apply the RPCA in just one round of consensus, utility can be gained through multiple rounds, each with an increasing minimum-required percentage of agreement, before the final round with an 80% requirement. These rounds allow for detection of latent nodes in the case that a few such nodes are creating a bottleneck in the transaction rate of the network. These nodes will be able to initially keep up during the lower-requirement rounds but fall behind and be identified as the threshold increases. In the case of one round of consensus, it may be the case that so few transactions pass the 80% threshold, that even slow nodes can keep up, lowering the transaction rate of the entire network.

4. Simulation Code

The provided simulation code demonstrates a round of RPCA, with parameterizable features (the number of nodes in the network, the number of malicious nodes, latency of messages, etc.). The simulator begins in perfect disagreement (half of the nodes in the network initially propose “yes”, while the other half propose “no”), then proceeds with the consensus process, showing at each stage the number of yes/no votes in the network as nodes adjust their proposals based upon the proposals of their UNL members. Once the 80% threshold is reached, consensus is achieved. We encourage the reader to experiment with different values of the constants defined at the beginning of “Sim.cpp”, in order to become familiar with the consensus process under different conditions.

5. Discussion

We have described the RPCA, which satisfies the conditions of correctness, agreement, and utility which we have outlined above. The result is that the Ripple Protocol is able to process secure and reliable transactions in a matter of seconds: the length of time required for one round of consensus to complete. These transactions are provably secure up to the bounds outlined in section 3, which, while not the strongest available in the literature for Asynchronous Byzantine consensus, do

allow for rapid convergence and flexibility in network membership. When taken together, these qualities allow the Ripple Network to function as a fast and low-cost global payment network with well-understood security and reliability properties.

While we have shown that the Ripple Protocol is provably secure so long as the bounds described in equations 1 and 3 are met, it is worth noting that these are maximal bounds, and in practice the network may be secure under significantly less stringent conditions. It is also important to recognize, however, that satisfying these bounds is not inherent to the RPCA itself, but rather requires management of the UNLs of all users. The default UNL provided to all users is already sufficient, but should a user make changes to the UNL, it must be done with knowledge of the above bounds. In addition, some monitoring of the global network structure is required in order to ensure that the bound in equation 3 is met, and that agreement will always be satisfied.

We believe the RPCA represents a significant step forward for distributed payment systems, as the low-latency allows for many types of financial transactions previously made difficult or even impossible with other, higher latency consensus methods.

6. Acknowledgments

Ripple Labs would like to acknowledge all of the people involved in the development of the Ripple Protocol consensus algorithm. Specifically, Arthur Britto, for his work on transaction sets, Jed McCaleb, for the original Ripple Protocol consensus concept, and David Schwartz, for his work on the “failure to agree is agreement to defer” aspect of consensus. Ripple Labs would also like to acknowledge Noah Youngs for his efforts in preparing and reviewing this paper.

References

- [1] Nakamoto, Satoshi. “Bitcoin: A peer-to-peer electronic cash system.” Consulted 1.2012 (2008): 28.
- [2] Lamport, Leslie, Robert Shostak, and Marshall Pease. “The Byzantine generals problem.” ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982): 382-401.
- [3] Attiya, C., D. Dolev, and J. Gill. “Asynchronous Byzantine Agreement.” Proc. 3rd. Annual ACM Symposium on Principles of Distributed Computing. 1984.

- [4] Fischer, Michael J., Nancy A. Lynch, and Michael S. Paterson. “Impossibility of distributed consensus with one faulty process.” Journal of the ACM (JACM) 32.2 (1985): 374-382.
- [5] Martin, J-P., and Lorenzo Alvisi. “Fast byzantine consensus.” Dependable and Secure Computing, IEEE Transactions on 3.3 (2006): 202-215.
- [6] Alchieri, Eduardo AP, et al. “Byzantine consensus with unknown participants.” Principles of Distributed Systems. Springer Berlin Heidelberg, 2008. 22-40.

EOS - An Introduction

Ian Grigg

Abstract—Current technologies for blockchain fall short of providing what developers and end-users need in order to contract together and to build large scale businesses. We propose EOS, a performance-based and self-governing blockchain that provides an operating system for building large-scale consumer-facing distributed applications. This paper outlines the context, vision and software architecture underlying EOS, which we are building to serve a broad and diverse group of users with smart business.

Keywords—EOS, blockchain, smart contract.

I. INTRODUCTION

The notions of digital cash and smart contracting have been known for a long time, yet only in recent times have strides been taken with respect to implementation.

This paper introduces the EOS.IO software underlying EOS as a new platform for general value and contracting. EOS is presented against a backdrop of three existing champions because (a) they represent a broad range of opinions as to the Distributed Ledger Technologies (DLT) space, (b) are large enough to matter, and (c) are familiar to the author.

Bitcoin (Nakamoto 2008) seemed to be the word on a blockchain that promised the inspirations of both digital cash and smart contracts. Although it captured the attention of the cypherpunks, media and *holders*, it failed to make a mark on business. Ethereum (Woods 2014) attempted to fulfill the smart contract promise with an “unstoppable world computer” while Bitshares (Larimer et al 2014) strove to open up the market for tradeable assets. Hundreds of alternative Bitcoin blockchains or *altcoins* strove to make a small difference seem louder. Corda (Brown et al 2016) backed away from blockchain entirely and explored party to party workflow solutions.

We are tantalisingly close but no prize has yet been awarded - by the end-users. It is timely to then take a fresh look at what the demand is for, from their perspective, and lay down the basis and a vision towards creating a practical and performant

Ian Grigg is a financial cryptographer and partner at block.one. iang@block.one (see <http://iang.org/>). This work is licensed under Creative Commons Attribution 4.0 International License (CC BY). Caveats:

(i) This paper is primarily about the EOS.IO software that permits a community to stand up an EOS blockchain. As the software is open source and a community is free of any controls beyond their own Constitution, this paper may be indicative but cannot be authoritative on any particular EOS blockchain that a community might wish to stand up.

(ii) I have endeavoured to make this paper as independent as possible, but biases are ever-present and are what make life special. For the record, confidential information known to the author has been excluded, and would likely change some criticisms if included, for better or worse.

(iii) This present version is a DRAFT for which I solicit broad feedback! Nothing written herein is especially fixed for the EOS.IO software, and changes are to be expected.

blockchain trade infrastructure. First, we summarise the **Context** of today’s market for DLTs. Then, we look at a **Vision** of the end-user’s needs, and how to meet them. Then, we review an **Architecture** to meet the market demands.

Finally a quick **Comparison** with known systems and **Concluding remarks**. For more technical details on the EOS.IO software, readers are referred to “EOS.IO Technical White Paper” (Larimer 2017).

II. CONTEXT

The Market. The market is competitive for all products and DLTs or blockchains are no exception. What are the market offerings? Bitcoin might be seen as *the chain of security*, yet a strong chain is only as valuable as the business it is attached to. Perhaps recognising this, Ethereum touted *the worldwide unstoppable Turing computer*, a goal that might appeal to computer scientists but has seemed elusive to other disciplines. R3 built Corda to serve the needs of the financial institution, which is a large market but also an expensive and exclusive one.

This section examines those prior systems from the perspective of major architectural features or necessities, which suggests benchmarks or assumed starting points that industry looks to.

Consensus. With blockchains, we come to consensus over a block of transactions, such that no transaction conflicts with any other, neither in this block nor prior blocks. Also known as the Two Generals Problem, there is a rich history in bringing remote actors to agreement such that “*I know that what you see is what I see.*” See Figure 1.

Bitcoin established *proof of work* or the *Nakamoto signature* as the way to bring an open entry community together over a shared or distributed ledger in which all parties hold a complete copy. This mechanism runs a lottery amongst many *miners* to determine who mines each block. Tickets in the lottery are competed for by a SHA2 puzzle, and as this requires energy to produce, the winner of the lottery is rewarded with a fixed amount of Bitcoin. In effect, anyone can be a General, and the one that wins the lottery is the one that sets this moment’s plan of battle. Following Generals can choose to accept that plan or block, or reject if invalid.

The fully shared ledger and the cost of proof of work, running at 4% for Bitcoin and 11% for Ethereum at the time of this paper’s writing, have offended many. *Permissioned ledgers* (Swanson 2015) were proposed to not only block those we want to exclude from enjoying the benefits of our ledger, but also to bring us back to the computer science roots of efficient consensus - practical but centralised designs well known in database science. Also proposed from time to time are proof of stake, exotic cryptography and secure enclaves. Corda (Brown et al 2016) established that consensus could be a user choice

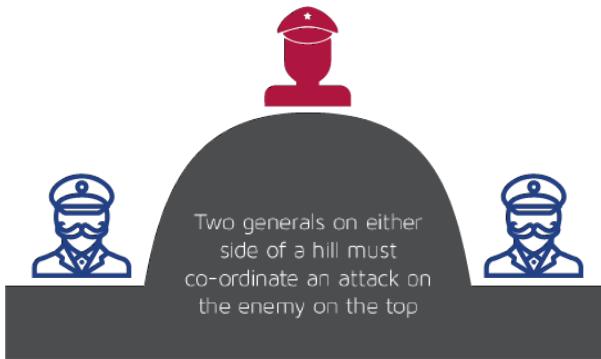


Fig. 1. The "Two Generals" Problem is fundamental in Computer Science

at select points within a contract of transactions. By allowing interchangeability of servers called *notaries* that can mediate the consensus by any of the above means, Corda reduces the network operating cost to a level comparable to today's IT infrastructure.

Value. Similarly, there are a wide variety of mechanisms to establish a fungible value such as cash. Smartcard money in the 1980s - 1990s was typically implemented through persistent internal data stores in each card that negotiated atomic dual-card transactions. In the same timeframe, David Chaum's eCash (Chaum 1983) popularised the notion of a coin, being a random number with a blinded signature that could be handed from user to user. Triple entry (Grigg 2005) established that each party could see the same receipt, each of which recorded a person to person transaction. Balance is calculated as the sum of receipts going in and out.

Bitcoin uses the UTXO or *unspent transaction output* concept, a state-driven layout. Each transaction record spends a set of previously unspent values, and creates new spendable values into the future. In contrast, Ethereum's virtual machine provided a database mechanism such that a currency could be constructed from a table, a significant improvement in flexibility, but opening up a wide surface area for attacks.

These five distinct mechanisms suggest that the way to account for value is not settled science.

State Transition. Bitcoin's block as a list of UTXOs, above, lays a claim to *state*, being the nature of those coins, that block, that chain, at that time. The duality of the UTXO design derives from the need of the lightweight or 'SPV' client to prove its incoming coins in a shared ledger: A receiving client with only limited access need only trace each single 'coin' from a block position back to its origin in order to determine that an incoming transaction is good. The receiver does not need to prove anything outside of the incoming coins, such as the sender's balance, in order to ensure complete control of the value.

This powerful statement that *the blockchain is a graph of state* was adopted broadly within the distributed ledger field. Even as Ethereum replaced the UTXO with its more powerful virtual machine, it accepted that state was the point of consensus over which all nodes need to reach. On arrival of a new hashed block, each validating node calculates and agrees

on the precise exit state resulting from all contracts found in each new block.

Contracts. Bitcoin added business logic to money by attaching validation 'scripts' to its transactions to suggest a limited form of contracting, which popularly became known as smart contracts (Szabo 1994, 1997). Ethereum's notion of the unstoppable worldwide Turing computer provided more fully powerful coding, messaging and data storage. Corda pared back these designs to validate and agree over UTXO-like state with command-driven changes, but also limit access to only the direct parties for confidentiality. Both Ethereum and Corda introduced more powerful high-level languages with which to express contracts.

Performance. Bitcoin has established a general limit of about 3 transactions per second (TPS), at which point transactions can be severely delayed. Ethereum seems to be stretched at 15 TPS, and a recent congestion event was marked by a \$2000 transaction fee to jump the queue. The limits on a blockchain's throughput are many: validating prior claimed blocks, processing the new block, and mining. Corda avoids these limits for the most part, as its consensus is via selectable, independent and localised *notaries*, as there is no need for wider consensus than the parties. Every system is encumbered by the physical limits of network propagation times.

Use Cases. Notwithstanding the hype surrounding blockchain, there is relatively little hard evidence of successful use cases. Bitcoin establishes a single currency, but the explosion of *altcoins*, the failure of colored coins, and the absence of any smart contracts of interest suggest clear limits. Ethereum tried to break those limits but to date success eludes, unless one considers the somewhat circular use case of raising funds on the promise of future use cases, as marked by steady traffic in ERC-20 contracts. Perhaps surprisingly, the progenitors of EOS number are two 'interesting' use cases that have reached production and scale, being a distributed exchange (Bitshares) and a social media site (Steem). The promise of smart contracts, however, remains elusive.

Governance. To this author, the critical discovery of Bitcoin is not that we can mediate with cryptography, or that the design is stable with decentralisation and open entry, but that *it must preserve these characteristics to survive*. Entry by all is not only key to the consensus model of hash-mining over the distributed ledger, it is also key to the survivability of the system. Previous digital cash systems failed because there was a centre, which was attacked in one way or another, showing a failure in governance. As if to provide further abundant evidence, centralised exchanges in the Bitcoin era are frequently attacked with thefts, contract breaches, denials of service, bankruptcies, seizures and enforced rule changes.

Then, the world divides generally into two: fully decentralised open entry systems typified by blockchains, and the converse typified by centralised and permissioned ledgers, with the space between the two being uncertain. Bifurcation over open entry raises the question of how the users govern, are governed, and how governance for the benefit works - in both cases.

The general approach in open entry starts with *caveat emptor*, which carefully sets a technical environment that is

capable of most of what is required, but with enforcement of rights limited by what can be automated in code. Sometimes labelled *trustlessness*, this regime draws a stark line between that which is technical and strong as a chain, and that which is at the user's discretion and therefore more dangerous. As time goes on, institutional approaches such as improvement proposals and centres of power such as foundations or teams arise to deal with some of the dangers to users, to a greater or lesser degree and success (Gupta 2014). Caveat emptor is typical of Bitcoin and Ethereum.

In contrast, in the *permissioned network* or *walled garden* approach, only those permitted can enter and act. In this scenario, parties open an account, are onboarded by an agent and can trade with a presumption of good behaviour. Implicitly or explicitly, enforcement of good behaviour is typically seen as out of scope at the technical level, although *dentity* typically plays an unclear part. The downside is that the wall around the garden can be expensive to erect and maintain, and every year the gatekeeper charges more. This approach is commonly assumed within heavily regulated markets such as banks and the like, and is used by Corda.

Neither of these world states are user friendly - users lose too much money through caveat emptor, and systems that start from 'permission' become systems that discriminate, either at the competitive level or the societal level. Users are routinely skeptical of either.

III. VISION

End-state Goals. What is it that our user needs? In the abstract, she wants to:

- Know her friends, business partners, and customers.
- Communicate with them.
- Be able to contract with them:
 - in the small, make peer to peer agreements, and
 - in the large, build a sophisticated business to be able to serve the market.
- Be able to retain and direct her value (pay bills, etc) as a necessary component of business.
 - Then, all has to be done safely and securely.
- Be able to invest in a predictable business. This is a complex issue, but appears to require three components.
 - Know that the ecosystem is advancing, and not at undue risk of failing.
 - Pay for development effort up front with reasonable payback in the future.
 - Because she knows that things - contracts, assets, transactions, intents - go wrong, she wants to be able to fix her difficulties. Including, with her friends, her business, and her assets, and quickly, cheaply and without undue escalation.

One caveat of arrogance: we assume her wants and her needs are synonymous. More precisely, we are making an entrepreneurial judgement call over what we believe the user *needs*, and she'll *want* it when she *learns* about it.

The Big Idea. It has become abundantly clear that for one reason or another, the promise of universal peer to peer

contracting and money has been excluded to the wider Internet. Bitcoin is too unsafe, and its smart contracts opaque. Ethereum is too scary, too hard, too geeky. Corda is 'big corporate.' Other systems have their weaknesses, all of them are restricted to the elite coder, and everyone has a different view.

What is needed is smart business for the everyday person. An everyday distributed application needs to live in a global blockchain that handles the open entry treasured by the Bitcoin discovery, has enough performance to build big business, is connected enough to bring people together and is safe and secure enough that Wall Street's *Gordon Gecko* can trade alongside Africa's *Mama Biashara*. Without drama, without fear, without missing out.

The Target. The vision before us is a single global contracting blockchain that can scale up to handle a long-tail of businesses negotiating contracts for mutual advantage in a safe and secure environment.

In more practical terms, while there is much of value on the Internet, we focus on what is mediated by the web, and leave aside mobile and applications for now. What does a builder of a web application want? We assume that the target user is the web entrepreneur, and therefore let's work backwards from that position.

Principal Features. Our design predicts a blockchain to handle thousands of transactions per second for business contracts that are captured in easy to use and easy to secure languages. The major features include:

- High performance messaging using event sourcing
- Delegated Proof of Stake
- Contracts as negotiation and intent - messaging at its heart
- Usability from the user to contract writer to developer to entrepreneur
- Governance for business and chain maintenance

The following section explores in more depth.

IV. THE ARCHITECTURE

The Philosophy. In large part the practical approach of the software underlying EOS is to extend the large-scale high-performance blockchain experience in Bitshares and Steem to support end-user business. Most of the elements have been proven to a lesser or greater extent, this architecture re-assembles them for a new purpose - to build distributed applications.

This section describes some important architectural differences that the software underlying EOS proposes against prior practice. For more technical details, readers are referred to the EOS.IO Technical White Paper (Larimer 2017).

The Message is the Medium. The EOS.IO software design switches from the more popular *consensus over state* to the less familiar *consensus over events* (Grigg, 2017-1). This approach marries the event sourcing pattern (Fowler, 2005) to a blockchain made of *events* rather than *state*.

In computer science, a deterministic state machine is built as a machine of code, state (memory), and events, both in and out. Every time something happens which causes a change, a practical machine saves intermediates to memory, and on



Fig. 2. A Coke Machine expressed as a state machine

restarting it recovers itself by reading back those intermediates. In building a practical state machine, we have a choice between saving *events* or saving *state*, which choice depends mostly on what we are trying to optimise.

In Figure 2, are we to save the red messages or the blue state? A machine saving state is more likely to be used in a context where we focus on what state it is in now, for example databases. A machine saving messages as intent is more likely to be useful when asking how we got to the state we are in now, for example protocols or legally significant logs such as triple entry accounting (Grigg 2005). Restart is faster with saved state, throughput is faster with saved messages.

Because users need performance, the design saves messages. Restart of a messaging or event sourced machine is similar to recovering from the beginning, therefore incredibly slow, and optimising startup means saving checkpoints - back to state again. But, and here is a crucial outcome, in saving that state, an actor remains bound by the saved messages, not the state, so we can optimise heavily and even recalculate the checkpoints if needed. Precisely how we optimise is too big a topic for this introduction, but suffice to predict that the combined techniques can in theory take blockchain from 3 transactions per second to 3 million.

Consensus. For consensus over messages, the EOS.IO architecture uses Delegated Proof of Stake (DPOS), a two-tier governance structure proven in Steem and Bitshares (Larimer 2014). In the first tier, block producers are elected into a round of 21, each producer gets one block per round, and is rewarded for the validation of incoming messages and production of the block of messages. A block released by one producer is validated by the next and the next and so forth; if not validated, it is not built upon. Similar longest-chain mechanics to Bitcoin are followed, and in short order, the producers converge on a longest chain. A block that is accepted by a quorum of producers is declared immutable, and the chain of immutable blocks becomes in effect a checkpoint.

Like proof of work, producers can censor (ignore) messages, or they can front-run by introducing their own from their superior knowledge of the future. To provide light-touch governance over bad acts by producers, each round of producers is continuously elected by the community using proof of stake (PoS). As this second tier blockchain-mediated election is over the producers and not the blocks, the so-called “nothing at stake” weakness does not apply.

In effect, a set of Generals is chosen for a campaign, and each get one turn. After the campaign, the civilian community asserts its view to replace any bad Generals.

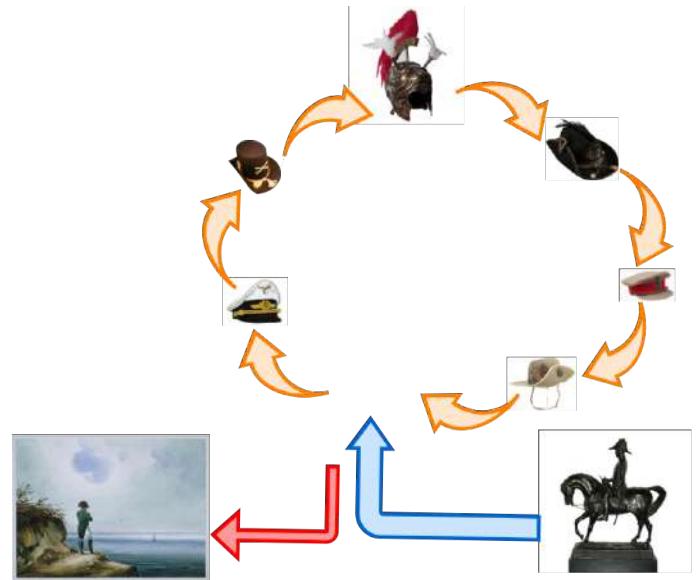


Fig. 3. Delegation allows replacement of Generals after a bad campaign

DPOS avoids the tax of mining, releasing that substantial value back to stakeholders. Value from block rewards would be initially captured entirely by the producers. However, because they are elected by the community, they are incentivised to share the rewards by a scheme that producers agree on amongst themselves, and promote to the community.

By constitution, the long term reward for producing blocks can be limited to for example 5% per annum (Larimer 2017-2). By custom, we suggest that the bulk of the value be returned to the community for the common good - software improvements, dispute resolution, and the like can be entertained. In the spirit of ‘eating our own dogfood,’ the design envisages that the community votes on a set of open entry contracts that act like ‘foundations’ for the benefit of the community. Known as *Community Benefit Contracts*, the mechanism highlights the importance of DPOS as enabling direct on-chain governance by the community (below).

The Contract. The architecture comes closer to the nature of contracting by treating contracts as a dynamic expression of negotiation, commitment and events, rather than the more static interpretation of ‘the four corners of the page’ or the performing code within a machine. We propose that messages are the natural element of contracting, as they better capture all phases of successful contracting: negotiation, intent, performance and breach of obligations are all events better captured as messages than, say, state.

A user writes a contract as a virtual construct of interlocking handlers of messages. A user can convert her account into a contracting agent by adding message handlers and using her account’s inbuilt database-like store to hold the internal position of her contracts. Several message handlers working together can mediate a flow of messages so as to perform a complete contract or legally sound agreement through its lifecycle.

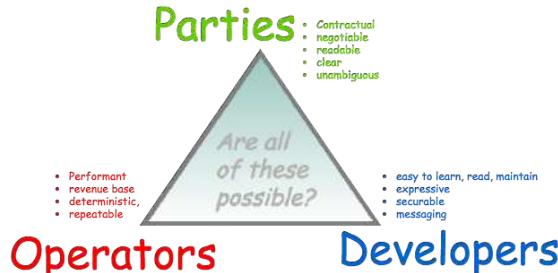


Fig. 4. Tensions between stakeholders in a blockchain

From the perspective of a contract, the arrival, acceptance and processing of a message is a simpler abstraction than state. Consider an order processing book as seen in a market for exchange: the book accepts bids to buy and offers to sell. When the time comes, it has to calculate a price at which to cross, and then issue accepted orders to both sides.

An order book in a messaging-based system is committing to its set of incoming messages and outgoing set of messages, which is a relatively tractable task. In contrast, in a fully state based system, all traders have to negotiate the acceptable state to all of many parties, including quantities and prices, before submitting a final state to the blockchain. This implies that traders would get to peek at the solution before agreeing, opening the door to game-playing. In practice, the only known way to solve this problem is with agents and messaging. An active agent receives committed messages, decides on the outcome, and sends out messages committing to that outcome.

Usability. The direct user of a blockchain is the developer who creates web apps for her end-users. To support an end-user, the software must support the developer, first and foremost, and it must do so in ways that help the developer to support her users. High impact support for the developer includes (a) the tools, (b) the language, and (c) the environment.

In the large, the EOS.IO developer will be supported by a web-based toolkit that provides a fully-serviced framework on which to build applications as distributed web-based systems coordinating over the blockchain. Accounts, naming, permissioning, recovery, database storage, scheduling, authentication and inter-app asynchronous communication are all built in. A goal of the architecture is to provide a fully-provisioned operating system for the builder of apps, focussed to the web because that's where the bulk of the users are.

Language. Within our context of industrial scale distributed applications, the language for writing contracts is high on the impact list. Most every other architectural feature in the EOS.IO software has solid foundation that is proven in Bitshares and Steem, whereas the addition of smart contracts stands out as uncharted territory.

It behoves us to analyse the language needs carefully. From the point of view of selecting technology for automated or smart contracting, the three stakeholders critical for success are: the parties, the developers and the operators.

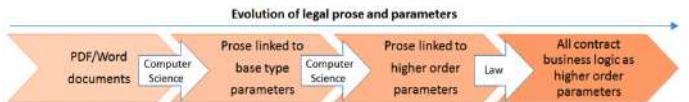


Figure 5: Parameters may become more sophisticated in the future, evolving from just simple base type parameters to also include more complex higher-order parameters. In the far future, if the encoding of business logic used in the parameters becomes acceptable to lawyers and admissible in court, then it could potentially replace the corresponding legal prose.

Fig. 5. Concepts in code automation and prose contracts will evolve (Clack1's Figure 5)

- The parties need a contract that is, first of all, an actual contract. Parties also want the contract to be negotiable, readable, clear, and unambiguous - they need their human intent to be captured faithfully. Preferably, contracts should also be supported by options for dispute resolution and enforceability.
- The developer needs the language and wider system to be easy to learn and write in, as well as expressive and securable, goals that often ignore higher semantics or contractual intent.
- Meanwhile the operators of the blockchain - producers of blocks and full-node app businesses - need the contract to be scalable and provide a reasonable basis for earning some revenue, interests that have little to do with human intent or developer expressibility.

Taking the parties' needs first, this pushes us in the direction of melding plaintext legal prose tightly with computer code, glued with some parameters to "drive the deal" and reuse the prose and code over many contracts (Grigg 2015). Many research efforts aim to merge the two contract views of code and prose together as either higher order parameters or a legally expressive domain specific language (Clack1 et al 2016 see their Figure 5) but none have as yet found this holy grail. This is an open research area with unsettled design choices (Clack2 et al 2016).

Along those lines, our first temptation was towards the developer: a source-interpreted scripting language based on Wren, and customised to manage the design of a contractual message handler. Example code snippet (Larimer 2017-1):

```
apply:
    // assuming all prior steps pass,
    // perform the state transition
    // that updates balances and/or
    // creates a new account for receiver
    var from = Balance[message.from]
    var to   = Balance.find( action.to )
    from.bal = from.bal - action.amount
    to.bal   = to.bal + action.amount
```

This hybrid of Wren is simple to learn, read, and reason about, making it ideal for automated contracting. However, it proved to be slow: a trial of trivial transactions capped out at 1,000 TPS, which brings us into collision with the needs of operators, our producers and application businesses.

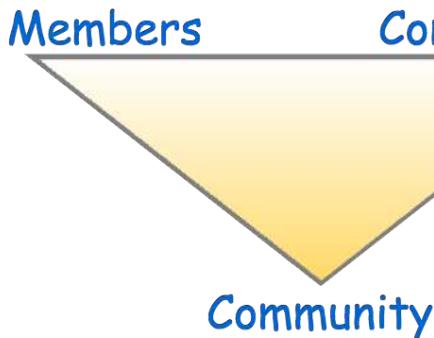


Fig. 6. Members forge a Community with a Constitution

As we are aiming for 100 times that level, the team switched to WebAssembly (WASM) which is a new intermediate language designed to do the job that Javascript currently does within browsers. WASM's first unoptimised trial within the EOS framework delivered about 50,000 TPS for a currency contract.

Yet, WASM switches the challenge from the operators to the parties - there are now 3 tangible views over any contract: legal prose, source code initially in C and intermediate code in WASM.

Thus it is a reasonable question to ask - what or where is the contract that the parties agreed to? I would like to face that question head on. In the two decades or so that I have seen contracts issued on the net, as Ricardian or otherwise, and the hundreds of issues that have arisen from these contracts, I have yet to see a dispute, or even a confusion where *what the contract said or meant* was key to the dispute. Even with *The DAO*, that ill-fated \$150 million lesson in how not to issue a contract, the proximate cause was (in) security, and regardless of which side of the fence one fell in identifying the *contractual significance of the hack*, the response was to arbitrarily change whatever needed to be changed to get the money back. There was no organised, formal or even a vestige of an attempt to resolve the dispute over interpretation of the facts, the meaning and the rights. It is an open question what proportion of disputes in court are over meanings and confusions, and what percentage are simply power plays and bullying, but I am not optimistic.

In the face of *The DAO* and other experiences, I suggest that the rule of one contract (Grigg 2004) looks dogmatic and overly constricting. Instead, at least for the unregulated part of the DLT space, there is opportunity to free up the components of the contract to achieve better performance, even at the expense of a little misalignment. Meanwhile, we should focus on governance, and making dispute resolution available and comfortable to the parties.

As of the time of writing, the set of languages available to the contract developer is a work in progress. Whether WASM or Wren or another, we will still need to structure the language for performance and usability. Each named message handler will need to identify sections for each of static, read-

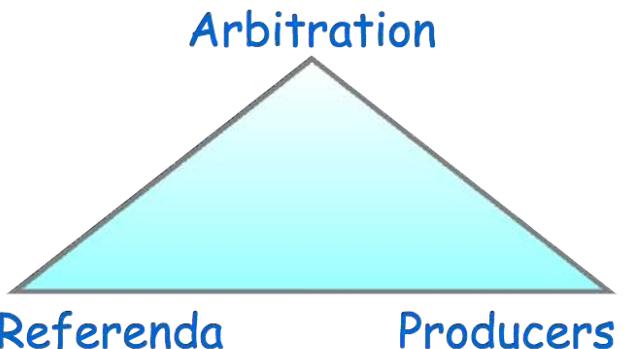


Fig. 7. Community can appoint governors to manage responsibilities

only and read-write code, each having different potentials for optimisation. To eliminate re-entrant issues, outgoing messages will be stacked until completion, or dropped on failure. We intend to add a SQL-like table structure to significantly ease adoption by those who are familiar with databases. Crypto will be external and mostly invisible.

As with the entire space for DLT, the competition continues internally. Wren is small and tight. WASM is only just out of standardisation. WASM's early tools target C and C++ which are popular but are more costly to write code in, in comparison to high level late-generation languages such as Wren. These challenges should not be insurmountable in the longer run as the WASM project is intended to work with most languages, and the bulk of the code in any DApp is outside the handlers, in the websites. The ability to accept many popular languages is enticing, an advantage available to Corda's JVM but not easily reachable by Bitcoin or Ethereum without a holistic approach to the developer cycle.

In conclusion, there are dramatic compromises in the choice of language and toolkits for the developer that go beyond mere codability. We would like an easy to read and reason scripting language that could speak in full contractual terms, be securable and be scaleable. But at the current state of the art, compromises have to be made.

Governance. Let us now turn to the environment. It is a reality that things go wrong with automated processing of contracts, to the distress of all. It is our hope to reduce both the frequency and the cost of those errors, but they cannot be eliminated entirely, and our approach is to build in remedial methods for when they do occur.

A blockchain based on EOS.IO software assumes that all who use the blockchain are members under a short Constitution (Larimer 2017-2) (Grigg 2017-3) and by agreeing to which, all members form a Community subject to the Constitution.

The Constitution sets down some basic rules for the benefit of the community. The Constitution empowers three arms of governance: arbitration for resolving disputes, block producers for choosing blocks, and referenda for community voice. Arranged in an interlocking triangle of governance, these three arms support and counterbalance each other. Referenda are used by the community to vote in the producers and arbitrators,

as well as changes to code and constitution. Arbitrators can deliver legally binding rulings to resolve disputes, and also for extraordinary changes such as hard forks. Block producers are at technical liberty to censor bad transactions or introduce remedial ones - but are mindful of community reaction. Arbitrators publish rulings, which producers might enforce, or users might seek external enforcement.

This counterbalanced arrangement ensures that no party or group has total power. Even founders or developers have only limited ability to affect the rights of the community members. Hard forks and other upgrades have a defined path, and individual disputes are channelled to a place where we can resolve and get back to business. A further benefit is that most of the above governance can be handled transparently, that is by writing contract handlers to accept and manage disputes, handle referenda and the like.

To make these institutions work, users have to agree to the Constitution, which empowers the producers to choose blocks, and reserves all disputes into the forum of arbitration. As well, the Constitution creates the legal rights expressed in the blockchain by stating that each member receives those rights properly accounted for, and in return each member supports the accounted rights of others. This trade of your rights for the rights of others becomes the cornerstone of the community, in that the community is defined by both the usage of the platform and the agreement to the Constitution.

And thus we have preserved open entry even as the Community governs itself internally. Even as a user transacts, all transactions from the first entry to the latest refer to the Constitution by hash, as a Ricardian Contract (Grigg 2004). As an explicit governance mechanism, the constitution creates more of a fenced field than a walled garden, and the gatekeeper is automated as a transaction or signpost at all points.

V. COMPARISONS

Bitcoin. As the platform that launched the first and most successful cryptocurrency, Bitcoin is a baseline. Yet, as the ‘first’ its flaws shine as bright as its success: The UTXO verification model means that complex smart business has to be mediated through external code. The state is nicely locked on chain, but the hard work of negotiation is done by the applications. It has no good framework for assets, especially as each transaction includes BTC, and is thus an affront to Gresham’s apocryphal warning against commingling of assets, *good money drives out bad*. Its lack of a thoughtful governance layer results that upgrades are very difficult, and the community is at war with itself. For example, the artificial limit of 3 TPS that kills its scalability is because of the absence of governance.

Ethereum. To rectify Bitcoin’s weaknesses, Ethereum establishes a Turing-complete virtual machine capability on a world-wide computer. It has several major shortfalls. Firstly, it has a dramatically restricting requirement to find consensus on state over thousands of program executions, leading to resource congestion at around 15 TPS. Secondly, the decision to go-it-alone on languages, VMs, toolkits and the like has caused a drag on developer capabilities. Thirdly, it suffers from the *ad-hocracy* of the Foundation that has emerged despite the refusal

of major stakeholders to recognise the need for governance. As an emergent business proposition, use of Ethereum has been dominated by raising funds for projects mostly aimed at finishing Ethereum as a platform, or competing with it. Few novel use cases have made their mark, suggesting that there is more work to do before the Ethereum concept of smart contracts bears fruit.

Corda. The primary distinguishing factor of Corda is that it is not a blockchain but a framework for party to party workflow. Instead of posting contracts and actions to a blockchain, parties exchange messages and come to consensus via notaries. It achieves confidentiality for parties, high performance unconstrained by chain coordination, and the ability for parties to control the contracts as they succeed and fail. Yet workflow works best with small numbers of parties, not large, and hence it is weaker on issuance of assets, especially cash and cash-denominated trading. Another weakness is that Corda’s walled garden approach for regulatory business stops it being an attractive mass market for small players.

VI. CONCLUSION

User experience. The direct users of a blockchain such as EOS are the entrepreneurs and developers who write contracts to implement distributed applications or DApps. Their users are the routine customers in retail, finance, logistics, media. Those latter customers do not need to know what a blockchain is. Hence the goal is to give the developers a platform that allows extensive business logic to be built, but the mechanisms of communication are hidden.

The DApp developer is given a fully capable accounts, permissioning and messaging platform in which to express the system. The user interface matches what users are familiar with - a webkit for building websites and of course access to the blockchain. This approach is expressed as “an operating system for blockchain.”

The fact that there is a blockchain can be hidden from the user, as exemplified by Steem, being just another blogging platform that happens to be distributed on a blockchain.

Use cases. An EOS blockchain is intended for high-performance messaging with business logic. Popular use cases will include supply chain, resource management, user-messaging such as social media, asset issuance and trading, accounting for remittances, and gaming.

A typical use case might be Uber. Ride-sharing is based on setting standards of behaviour for the driver and for the passenger. If drivers and passengers were part of the same community, there would be an immediate benefit - the base of liability and standards of behaviour would be covered under community constitution and dispute resolution, and their contracts could be bilateral rather than intermediated, thus minimising any regulatory difficulties.

Then, as the contracts can be bilateral, the business flow could be split up: tracking passengers in the market, tracking cars available, finding a match, negotiating a contract, performance, settlement, pricing, and social tracking could all be built as separate DApps that interact.

Community. To support business, we need to solve problems. And to scale the solving of problems, it has to be done

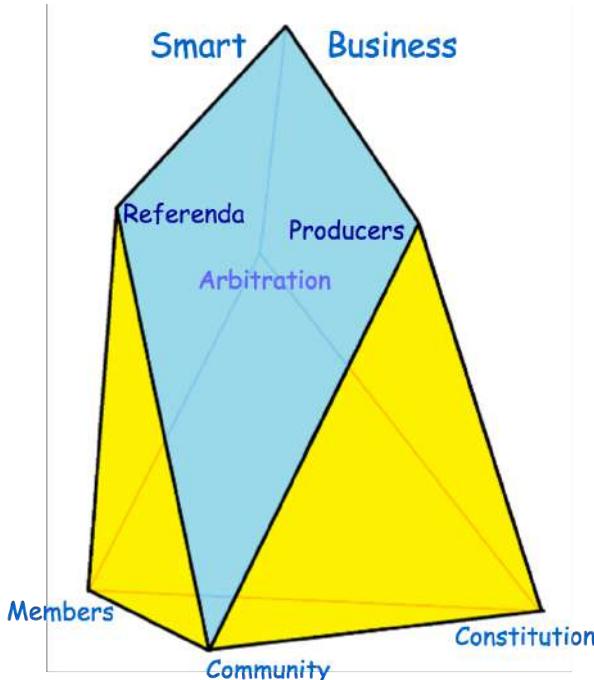


Fig. 8. The point is Smart Business

by the community itself, which means it has to be in the architecture. To advance community, we must preserve open entry, but on entry provide the tools that users find useful for governance. Users want to determine their risks and obligations to their counterparties.

When bound together as a community under a Constitution, users will know that the rights, liabilities and obligations of their counterparties are at least to a basic standard, as expressed in a constitution and as enforced in dispute resolution. In addition reliable names and a web of trust can reduce the anonymity of the Internet and give people a sense of belonging to something important.

ACKNOWLEDGMENT

This paper received useful feedback from Brendan Blumer, Arthur Doohan, Dan Larimer, Wendy Lee, Aaron Leibling, Konstantinos Sgantzos, Joseph VaughnPerling, Kokuei Yuan.

REFERENCES

- [1] Richard Brown, James Carlyle, Ian Grigg, Mike Hearn, "Corda: an Introduction" 2016
- [2] David Chaum, "Blind Signatures for Untraceable Payments", 1982 UC Santa Barbara
<http://blog.koehtopp.de/uploads/Chaum.BlindSigForPayment.1982.PDF>
- [3] Christopher D. Clack (1), Vikram A. Bakshi, Lee Braine "Smart Contract Templates: foundations, design landscape and research directions", 2016
- [4] Christopher D. Clack (2), Vikram A. Bakshi, Lee Braine "Smart Contract Templates: essential requirements and design options", 2016
- [5] Martin Fowler, "Event Sourcing", 2005
<https://martinfowler.com/eaaDev/EventSourcing.html>
- [6] Ian Grigg, "The Ricardian Contract," 2004
- [7] Ian Grigg, "Triple Entry Accounting," 2005
- [8] Ian Grigg, "The Sum of All Chains - Let's Converge," 2015
- [9] Ian Grigg, blog post "The Message is the Medium," 2017-1
- [10] Ian Grigg, blog post "Seeking Consensus on Consensus," 2017-2
- [11] Ian Grigg, blog post "A Principled Approach to Blockchain Governance" 2017-3
- [12] Vinay Gupta, interview "Bitcoin Cannot be divorced from pre-existing political theory," 2014
- [13] Daniel Larimer, "Delegated Proof-of-Stake (DPOS)" 2014.
- [14] Daniel Larimer, Charles Hoskinson, Stan Larimer, "A Peer-to-Peer Polymorphic Digital Asset Exchange" 2014.
- [15] Dan Larimer, "EOS.IO Technical White Paper" block.one 2017
<https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>
- [16] Dan Larimer, block post "Implementing a Hypothetical Currency Application on EOS," 2017-1
<https://steemit.com/eos/@eosio/implementing-a-hypothetical-currency-application-on-eos>
- [17] Dan Larimer, blog post "What could a blockchain Constitution look like?" 2017-2
- [18] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System " 2008
- [19] Tim Swanson, "Consensus-as-a-Service" 2015
<http://www.ofnumbers.com/wp-content/uploads/2015/04/Permissioned-distributed-ledgers.pdf>
- [20] Nick Szabo, "Smart Contracts", 1994
- [21] Nick Szabo, "Formalizing and Securing Relationships on Public Networks", 1997
- [22] Gavin Woods, "Ethereum: A Secure Decentralised Generalised Transaction Ledger", 2014



Tether: Fiat currencies on the Bitcoin blockchain

Abstract. A digital token backed by fiat currency provides individuals and organizations with a robust and decentralized method of exchanging value while using a familiar accounting unit. The innovation of blockchains is an auditable and cryptographically secured global ledger. Asset-backed token issuers and other market participants can take advantage of blockchain technology, along with embedded consensus systems, to transact in familiar, less volatile currencies and assets. In order to maintain accountability and to ensure stability in exchange price, we propose a method to maintain a one-to-one reserve ratio between a cryptocurrency token, called tethers, and its associated real-world asset, fiat currency. This method uses the Bitcoin blockchain, Proof of Reserves, and other audit methods to prove that issued tokens are fully backed and reserved at all times.

Table of Contents

[Table of Contents](#)

[Introduction](#)

[Technology Stack and Processes](#)

[Tether Technology Stack](#)

[Flow of Funds Process](#)

[Proof of Reserves Process](#)

[Implementation Weaknesses](#)

[Main Applications](#)

[For Exchanges](#)

[For Individuals](#)

[For Merchants](#)

[Future Innovations](#)

[Multi-sig and Smart Contracts](#)

[Proof of Solvency Innovations](#)

[Conclusion](#)

[Appendix](#)

[Audit Flaws: Exchanges and Wallets](#)

[Limitations of Existing Fiat-pegging Systems](#)

[Market Risk Examples](#)

[Legal and Compliance](#)

[Glossary of Terms](#)

[References](#)

Introduction

There exists a vast array of assets in the world which people freely choose as a store-of-value, a transactional medium, or an investment. We believe the Bitcoin blockchain is a better technology for transacting, storing, and accounting for these assets. Most estimates measure global wealth around 250 trillion dollars [1] with much of that being held by banks or similar financial institutions. The migration of these assets onto the Bitcoin blockchain represents a proportionally large opportunity.

Bitcoin was created as “an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party.”[2]. Bitcoin created a new class of digital currency, a decentralized digital currency or cryptocurrency¹.

Some of the primary advantages of cryptocurrencies are: low transaction costs, international borderless transferability and convertibility, trustless ownership and exchange, pseudo-anonymity, real-time transparency, and immunity from legacy banking system problems [3]. Common explanations for the current limited mainstream use of cryptocurrencies include: volatile price swings, inadequate mass-market understanding of the technology, and insufficient ease-of-use for non-technical users.

The idea for asset-pegged cryptocurrencies was initially popularized² in the Bitcoin community by the Mastercoin white paper authored by J.R. Willett in January 2012[4]. Today, we’re starting to see these ideas built with the likes of BitAssets, Ripple, Omni, Nxt, NuShares/Bits, and others. One should note that all Bitcoin exchanges and wallets (like Coinbase, Bitfinex, and Coinapult) which allow you to hold value as a fiat currency already provide a *similar* service in that users can avoid the volatility (or other traits) of a particular cryptocurrency by selling them for fiat currency, gold, or another asset. Further, almost all types of existing financial institutions, payment providers, etc, which allow you to hold fiat value (or other assets) subsequently provide a similar service. In this white paper we focus on applications wherein the fiat value is stored and transmitted with software that is open-source, cryptographically secure, and uses distributed ledger technology, i.e. a true cryptocurrency.

While the goal of any successful cryptocurrency is to completely eliminate the requirement of trust, each of the aforementioned implementations either rely on a trusted third party or have other technical, market-based, or process-based drawbacks and limitations³.

¹ For definitions throughout, see [Glossary of Terms](#)

² But has been discussed since Dr. Szabo’s proposed BitGold [5]

³ Summarized in the Appendix, here: [Limitations of Existing Fiat-pegging Systems](#)

In our solution, fiat-pegged cryptocurrencies are called “tethers”. All tethers will initially⁴ be issued on the Bitcoin blockchain via the Omni Layer protocol and so they exist as a cryptocurrency token. Each tether unit issued into circulation is backed in a one-to-one ratio (i.e. one Tether USDT is one US dollar) by the corresponding fiat currency unit held in deposit by Hong Kong based Tether Limited. Tethers may be redeemable/exchangeable for the underlying fiat currency pursuant to Tether Limited’s terms of service or, if the holder prefers, the equivalent spot value in Bitcoin. Once a tether has been issued, it can be transferred, stored, spent, etc just like bitcoins or any other cryptocurrency. The fiat currency on reserve has gained the properties of a cryptocurrency and its price is permanently *tethered* to the price of the fiat currency.

Our implementation has the following advantages over other fiat-pegged cryptocurrencies:

- Tethers exist on the Bitcoin blockchain rather than a less developed/tested “altcoin” blockchain nor within closed-source software running on centralized, private databases.
- Tethers can be used just like bitcoins, i.e. in a p2p, pseudo-anonymous, decentralized, cryptographically secure environment.
- Tethers can be integrated with merchants, exchanges, and wallets just as easily as Bitcoin or any other cryptocurrencies can be integrated.
- Tethers inherit the properties of the Omni Layer protocol which include: a decentralized exchange; browser-based, open-source, wallet encryption; Bitcoin-based transparency, accountability, multi-party security and reporting functions.
- Tether Limited employs a simple but effective approach for conducting Proof of Reserves which significantly reduces our counterparty risk as the custodian of the reserve assets.
- Tether issuance or redemption will not face any pricing or liquidity constraints. Users can buy or sell as many tethers as they want, quickly, and with very low fees.
- Tethers will not face any market risks⁵ such as Black Swan events, liquidity crunches, etc as reserves are maintained in a one-to-one ratio rather than relying on market forces.
- Tether’s one-to-one backing implementation is easier for non-technical users to understand as opposed to collateralization techniques or derivative strategies.

At any given time the balance of fiat currency held in our reserves will be equal to (or greater than) the number of tethers in circulation. This simple configuration most easily supports a reliable Proof of Reserves process; a process which is fundamental to maintaining the price-parity between tethers in circulation and the underlying fiat currency held in reserves. In this paper we provide evidence⁶ that shows exchange and

⁴ More Bitcoin 2.0 protocols will come soon, like Ripple, Nxt, etc

⁵ See Appendix, section: [Market Risk Examples](#)

⁶ See section: [Proof of Solvency Process](#)

wallet audits (in their current state) are very unreliable (i.e. flaws in Proof of Solvency[6] methods) and instead propose that exchanges and wallets *outsource* the custody of user funds to us via tethers.

Users can purchase tethers from Tether.to (our web-wallet) or from supported exchanges such as Bitfinex who support tethers as a deposit and withdrawal method. Users can also transact and store tethers with any Omni Layer enabled wallet like Ambisafe, Holy Transaction or Omni Wallet. Other exchanges, wallets, and merchants are encouraged to reach out to us about integrating tether as a surrogate for traditional fiat payment methods.

We recognize that our implementation isn't perfectly decentralized⁷ since Tether Limited must act as a centralized custodian of reserve assets (albeit tethers in circulation exist as a decentralized digital currency). However, we believe this implementation sets the foundation for building future innovations that will eliminate these weaknesses, create a robust platform for new products and services, and support the growth and utility of the Bitcoin blockchain over the long run. Some of these innovations include:

- Mobile payment facilitation between users and other parties, including other users and merchants
- Instant or near-instant fiat value transfer between decentralized parties (such as multiple exchanges)
- Introduction to the use of smart contracts and multi-signature capabilities to further improve the general security process, Proof of Reserves, and enable new features.

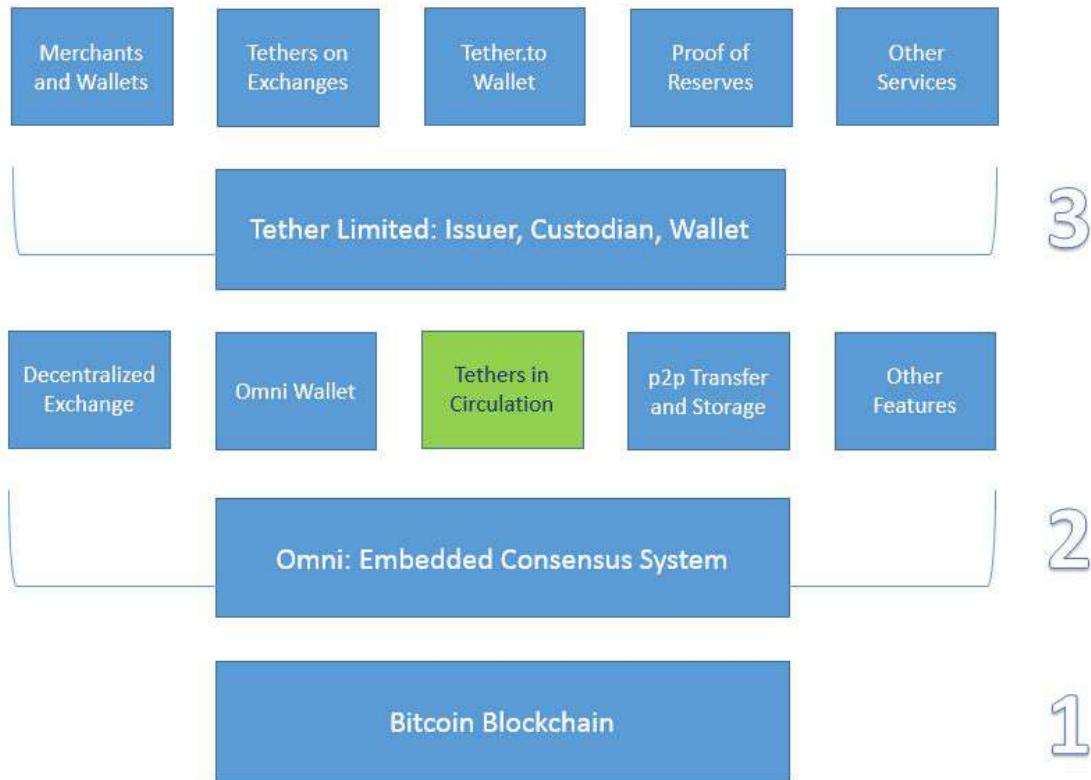
Technology Stack and Processes

Each tether issued into circulation will be backed in a one-to-one ratio with the equivalent amount of corresponding fiat currency held in reserves by Hong Kong based Tether Limited. As the custodian of the backing asset we are acting as a trusted third party responsible for that asset. This risk is mitigated by a simple implementation that collectively reduces the complexity of conducting both fiat and crypto audits while increasing the security, provability, and transparency of these audits.

Tether Technology Stack

The stack has 3 layers, and numerous features, best understood via a diagram

⁷ See section: [Implementation Weaknesses](#)



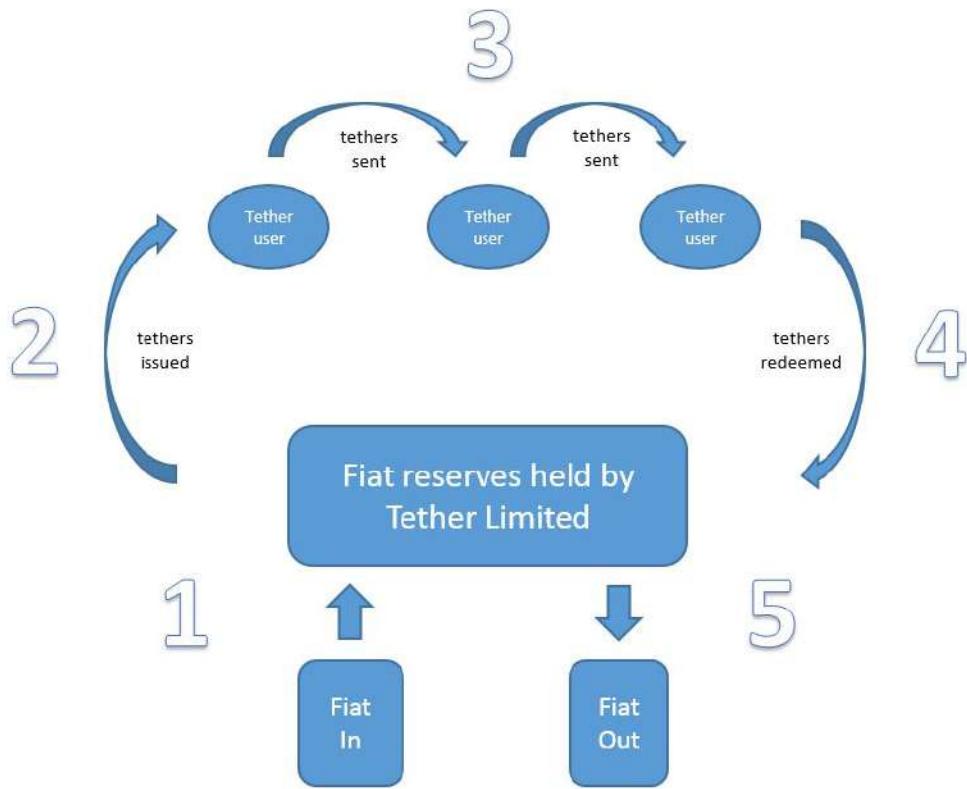
Here is a review of each layer.

- 1) The first layer is the Bitcoin blockchain. The Tether transactional ledger is embedded in the Bitcoin blockchain as meta-data via the embedded consensus system, Omni.
- 2) The second layer is the Omni Layer protocol. Omni is a foundational technology that can:
 - a) Grant (create) and revoke (destroy) digital tokens represented as meta-data embedded in the Bitcoin blockchain; in this case, fiat-pegged digital tokens, tethers.
 - b) Track and report the circulation of tethers via Omnichest.info (Omni asset ID #31, for example, represents TetherUSD) and Omnicore API.
 - c) Enable users to transact and store tethers and other assets/tokens in a:
 - i) p2p, pseudo-anonymous, cryptographically secure environment.
 - ii) open-source, browser-based, encrypted web-wallet: Omni Wallet.
 - iii) multi-signature and offline cold storage-supporting system
- 3) The third layer is Tether Limited, our business entity primarily responsible for:
 - a) Accepting fiat deposits and issuing the corresponding tethers
 - b) Sending fiat withdrawals and revoking the corresponding tethers
 - c) Custody of the fiat reserves that back all tethers in circulation

- d) Publicly reporting Proof of Reserves and other audit results
- e) Initiating and managing integrations with existing Bitcoin/blockchain wallets, exchanges, and merchants
- f) Operating Tether.to, a web-wallet which allows users to send, receive, store, and convert tethers conveniently.

Flow of Funds Process

There are five steps in the lifecycle of a tether, best understood via a diagram.



Step 1 - User deposits fiat currency into Tether Limited's bank account.

Step 2 - Tether Limited generates and credits the user's tether account. Tethers enter circulation. Amount of fiat currency deposited by user = amount of tethers issued to user (i.e. 10k USD deposited = 10k tetherUSD issued).

Step 3 - Users transact with tethers⁸. The user can transfer, exchange, and store tethers via a p2p open-source, pseudo-anonymous, Bitcoin-based platform.

Step 4 - The user deposits tethers with Tether Limited for redemption into fiat currency.

Step 5 - Tether Limited destroys the tethers and sends fiat currency to the user's bank account.

Users can obtain tethers outside of the aforementioned process via an exchange or another individual. Once a tether enters circulation it can be traded freely between any business or individual. For example, users can purchase tethers from Bitfinex, with more exchanges to follow soon.

The main concept to be conveyed by the Flow of Funds diagram is that Tether Limited is the only party who can issue tethers into circulation (create them) or take them out of circulation (destroy them). This is the main process by which the system solvency is maintained.

Proof of Reserves Process

Proof of Solvency, Proof of Reserves, Real-Time Transparency, and other similar phrases have been growing and resonating across the cryptocurrency industry.

Exchange and wallets audits, in their current form, are very unreliable. Insolvency has occurred numerous times in the Bitcoin ecosystem, either via hacks, mismanagement, or outright fraud. Users must be diligent with their exchange selection and vigilant in their use of exchanges. Even then, a savvy user will not be able to fully eliminate the risks. Further, there are exchange users like traders and businesses who must keep non-trivial fiat balances in exchanges at all times. In financial language, this is known as the “counterparty risk” of storing value with a third party.

We believe it's safe to conclude that exchange and wallet audits in their current form are not very reliable. These processes do not guarantee users that a custodian or exchange is solvent. Although there have been great contributions to improving the exchange audit processes, like the Merkle tree approach[6], major flaws⁹ still remain.

Tether's Proof of Reserves configuration is novel because it simplifies the process of proving that the total number of tethers in circulation (liabilities) are always fully backed by an equal amount of fiat currency held

⁸ See benefits of using tethers in the section: [Main Applications](#)

⁹ See section: [Audit Flaws: Exchanges and Wallets](#)

in reserve (assets). In our configuration, each tetherUSD in circulation represents one US dollar held in our reserves (i.e. a one-to-one ratio) which means the system is fully reserved when the sum of all tethers in existence (at any point in time) is exactly equal to the balance of USD held in our reserve. Since tethers live on the Bitcoin blockchain, the provability and accounting of tethers at any given point in time is trivial. Conversely, the corresponding total amount of USD held in our reserves is proved by publishing the bank balance and undergoing periodic audits by professionals. Find this implementation further detailed below:

- Tether Limited issues all tethers via the Omni Layer protocol. Omni operates on top of the Bitcoin blockchain and therefore all issued, redeemed, and existing tethers, including transactional history, are publicly auditable via the tools provided at Omnichest.info.
 - The Omnichest.info asset ID for tetherUSD is #31.
 - Here is a link: <http://omnichest.info/lookupsp.aspx?sp=31>
 - Let the total number of tethers issued under this asset ID be denoted as TUSDissue
 - Let the total number of tethers redeemed under this asset ID be denoted as TUSDredeem
 - Let the total number of tethers in circulation at any time be denoted as TUSD
 - $TUSD = TUSDissue - TUSDredeem$
 - TUSD = “Total Property Tokens” @ <http://omnichest.info/lookupsp.aspx?sp=31>
- Tether Limited has a bank account which will receive and send fiat currency to users who purchase/redeem tethers directly with us.
 - Let the total amount deposited into this account be denoted as DUSDdepo
 - Let the total amount withdrawn from this account be denoted as DUSDwithd
 - Let the dollar balance of this bank account be denoted as DUSD
 - $DUSD = DUSDdepo - DUSDwithd$
- Each tether issued will be backed by the equivalent amount of currency unit (one tetherUSD equals one dollar). By combining the above crypto and fiat accounting processes, we conclude the “Solvency Equation” for the Tether System.
 - The Solvency Equation is simply $TUSD = DUSD$.
 - Every tether issued or redeemed, as publicly recorded by the Bitcoin blockchain will correspond to a deposit or withdrawal of funds from the bank account.
 - The provability of TUSD relies on the Bitcoin blockchain as discussed previously.
 - The provability of DUSD will rely on several processes:
 - We publish the bank account balance on our website’s Transparency page.
 - Professional auditors will regularly verify, sign, and publish our underlying bank balance and financial transfer statement.

Users will be able to view this information from our Transparency Page, which will look like:



For clarity, we'd like to acknowledge that the Tether System¹⁰ is different than the Tether.to web-wallet in terms of Proof of Reserves. In this paper, we mostly focus on Proof of Reserves for the Tether System; i.e. all tethers in circulation at any point in time. The Tether.to wallet is a consumer facing web-wallet operating on closed-source code and centralized servers. Conducting a Proof of Reserves for this wallet is fundamentally different than what we've outlined for the Tether System.

We're planning the deployment of a PoR-based transparency solution for the Tether.to wallet. We believe it will be the most advanced PoR system in existence today. It overcomes almost all of the challenges outlined in the appendix¹¹ on this topic. Mind you, users can always secure tethers through managing the private keys themselves or through Omni Wallet.

Implementation Weaknesses

We understand that our implementation doesn't immediately create a fully trustless cryptocurrency system. Mainly because users must trust Tether Limited and our corresponding legacy banking institution to be the custodian of the reserve assets. However, almost all exchanges and wallets (assuming they hold USD/fiats) are subject to the same weaknesses. Users of these services are already subject to these risks. Here is a summary of the weaknesses in our approach:

- We could go bankrupt
- Our bank could go insolvent
- Our bank could freeze or confiscate the funds
- We could abscond with the reserve funds

¹⁰ See [Glossary of Terms](#)

¹¹ See [Audit Flaws: Exchanges and Wallets](#)

- Re-centralized of risk to a single point of failure

Observe that almost all digital currency exchanges and wallets (assuming they hold USD/fiat) already face many of these challenges. Therefore, users of these services are already subject to these risks. Below we describe how each of these concerns are being addressed.

We could go bankrupt - In this case, the business entity Tether Limited would go bankrupt but client funds would be safe, and subsequently, all tethers will remain redeemable. Most security breaches on Bitcoin businesses have targeted cryptocurrencies rather than bank accounts. Since all tethers exist on the Bitcoin blockchain they can be stored by individuals directly through securing their own private keys.

Our bank could go insolvent - This is a risk faced by all users of the legacy financial system and by all exchange operators. Tether Limited currently has accounts with Cathay United Bank and Hwatai Bank in Taiwan, both of whom are aware and confident that Tether's business model is acceptable. Additional banking partners are being established in other jurisdictions to further mitigate this concern.

Our bank could freeze or confiscate the funds - Our banks are aware of the nature of Bitcoin and are accepting of Bitcoin businesses. They also provide banking services to some of the largest Bitcoin exchanges globally. The KYC/AML processes we follow are also used by the other digital currency exchanges they currently bank. They have assured us we are in full compliance¹².

We could abscond with the reserve assets - The corporate charter is public¹³ as well as the business owners names, locations, and reputations. Ownership of the account is legally bound to the corporate charter. Any transfers in or out of the bank account will have the associated traces and are bound by rigid internal policies.

Re-centralization of risk to a single point of failure - We have some ideas on how to overcome this and we'll be sharing them in upcoming blog and product updates. There are many ways to tackle this problem. For now, this initial implementation gets us on the right track to realize these innovations in following versions. By leveraging the platforms we have chosen, we have reduced the centralization risk to one singular responsibility: the creation and redemption of tokens. All other aspects of the system are decentralized.

¹² See section on [Legal and Compliance](#) for more information

¹³ Same as footnote #10

Main Applications

In this section we'll summarize and discuss the main applications of tethers across the Bitcoin/blockchain ecosystem and for other consumers globally. We break up the beneficiaries into three user groups: Exchanges, Individuals, and Merchants.

The main benefits, applicable to all groups:

- Properties of Bitcoin bestowed upon other asset classes
- Less volatile, familiar unit of account
- World's assets migrate to the Bitcoin blockchain

For Exchanges

Exchange operators understand that accepting fiat deposits and withdrawals using legacy financial systems can be complicated, risky, slow, and expensive. Some of these issues include:

- Identifying the right payment providers for your exchange
 - irreversible transactions, fraud protection, lowest fees, etc
- Integrating the platform with banks who have no APIs
- Liaising with these banks to coordinate compliance, security, and to build trust
- Prohibitive costs for small value transfers
- 3-7 days for international wire transfers to clear
- Poor and unfavorable currency conversion fees

By offering tethers, an exchange can relieve themselves of the above complications and gain additional benefits, such as:

- Accept crypto-fiat as deposit/withdrawal/storage method rather than using a legacy bank or payment provider
 - Allows users to move fiat in and out of exchange more freely, quickly, cheaply
- Outsource fiat custodial risk to Tether Limited - just manage cryptos
- Easily add other tethered fiat currencies as trading pairs to the platform
- Secure customer assets purely through accepted crypto-processes
 - Multi-signature security, cold and hot wallets, HD wallets, etc

- Conduct audits easier and more securely in a purely crypto environment
- Anything one can do with Bitcoin as an exchange can be done with tethers

Exchange users know how risky it can be to hold fiat currencies on an exchange. With the growing number of insolvency events it can be quite dangerous. As mentioned previously, we believe that using tethers exposes exchange users to less counterparty risk than continually holding fiat on exchanges. Additionally, there are other benefits to holding tethers, explained in the next section.

For Individuals

There are many types of individual Bitcoin users in the world today. From traders looking to earn profits daily; to long term investors looking to store their Bitcoins securely; to tech-savvy shoppers looking to avoid credit card fees or maintain their privacy; to philosophical users looking to change the world; to those looking to remit payments globally more effectively; to those in third world countries looking for access to financial services for the first time; to developers looking to create new technologies; to all those who have found many uses for Bitcoin. For each of these individuals, we believe tethers are useful in similar ways, like:

- Transact in USD/fiat value, pseudo-anonymously, without any middlemen/intermediaries
- Cold store USD/fiat value by securing one's own private keys
- Avoid the risk of storing fiat on exchanges - move crypto-fiat in and out of exchanges easily
- Avoid having to open a fiat bank account to store fiat value
- Easily enhance applications that work with bitcoin to also support tether
- Anything one can do with Bitcoin as an individual one can also do with tether

For Merchants

Merchants want to focus on their business, not on payments. The lack of global, inexpensive, ubiquitous payment solutions continue to plague merchants around the world both large and small. Merchants deserve more. Here are some of the ways tether can help them:

- Price goods in USD/fiat value rather than Bitcoin (no moving conversion rates/purchase windows)
- Avoid conversion from Bitcoin to USD/fiat and associated fees and processes
- Prevent chargebacks, reduce fees, and gain greater privacy
- Provide novel services because of fiat-crypto features
 - Micropayments, gift cards, more
- Anything one can do with Bitcoin as a merchant one can also do with tether

Future Innovations

Multi-sig and Smart Contracts

Proof of Solvency Innovations

Conclusion

Tether constitutes the first Bitcoin-based fiat-pegged cryptocurrencies in existence today. Tether is based on the Bitcoin blockchain, the most secure and well-tested blockchain and public ledger in existence. Tethers are fully reserved in a one-to-one ratio, completely independent of market forces, pricing, or liquidity constraints. Tether has a simple and reliable Proof of Reserves implementation and undergoes regular professional audits. Our underlying banking relationships, compliance, and legal structure provide a secure foundation for us to be the custodian of reserve assets and issuer of tethers. Our team is composed of experienced and respected entrepreneurs from the Bitcoin ecosystem and beyond.

We are focused on arranging integrations with existing businesses in the cryptocurrency space. Business like exchanges, wallets, merchants, and others. We're already integrated with Bitfinex, HolyTransaction, Omni Wallet, Poloniex, C-CEX, and more to come. Please reach out to us to find out more.

Appendix

Audit Flaws: Exchanges and Wallets

Here is a summary of the current flaws found in technology-based¹⁴ exchange and wallet audits.

In the Merkle tree[6] approach users must manually report that their balances (user's leaf) have been correctly incorporated in the liability declaration of the exchange (the Merkle hash of the exchange's database of user balances). This proposed solution works if enough users verify that their account was included in the tree, and in a case where their account is not included this instance would be reported. One potential risk is that an exchange database owner could produce a hash that is not the true representation of

¹⁴ As opposed to hiring a professional auditor

the database at all; it hashes an incomplete database which would reduce its apparent liabilities to customers, making them appear solvent to a verifying party. Here are some scenarios where a fraudulent exchange would exclude accounts and :

- “Bitdust” Accounts: Inactive or low activity accounts would lower the chance that an uninterested user would check or report inconsistencies. In some cases these long-tail accounts could represent a significant percentage of the exchange’s liabilities.
- “Colluding Whales” Attack: There is evidence that large Bitcoin traders are operating on various exchanges and moving markets significantly. Such traders need to have capital reserves at the largest exchanges to quickly execute orders. Often, traders choose exchanges that they “trust”. In this way they can be assured that should a hack or liquidity issue arise, they have priority to get their money out. In this case, the exchange and trader could collude to remove the whales account balance from the database before it’s hashed.
- Key Rental Attack: To pass the audit, a malicious exchange could rent the private keys to bitcoins they do not own. This would make them appear solvent by increasing their assets without any acknowledgment that those funds were loaned to them. Likewise, they could “borrow” fiat currency to do the same.
- There are more attacks not discussed here.

Reaching Statistical Significance (reporting completeness): Even outside of these three attack vectors, a database that has been manipulated may never be detected if a sufficient number of users are not validating balances. The probability of getting 100% of the users to verify balances is likely zero, even with proper incentivization structure for users to verify their balances. Therefore, auditors would need statistical tools to make statements about the validity of an exchange’s database based on sampling frequency, size, and other properties.

Currently users have no way to receive compensation by legal means in case something goes wrong with the exchange. For example, when Mt.Gox closed operations, many users might not have independently recorded their account balances (prints screens, signed messages to themselves, etc) in a way that could conclusively prove to law enforcement that this exchange’s I.O.U’s actually existed. Such users are at the mercy of the exchange to somehow publish a record of that hash tree or original database.

The proposed structure in which these audits would be performed still contains some subtle but important flaws. In particular, the data reporting (hash tree) on the institution’s website gives no guarantee at all to

users, as a malicious exchange could publish different states/balances to different groups of users, or retroactively change the state. Thus it is fundamental to publish this data through a secure broadcast channel, e.g. the Bitcoin blockchain.

Privacy is a barrier to entry for the adoption of an automated/open auditing system. While some progress has been made towards better privacy there is no perfect solution yet. Further, to build up an accurate user verified liability space, these users will have to report account balances with the exchange and Bitcoin addresses. Some users likely would not report this information regardless of the incentive, therefore providing cryptographically secure privacy whilst obtaining the reporting goal is paramount.

Time Series: the Merkle tree hash is a single snapshot of the database at a single point in time. Not having a somewhat continuous time series of the database opens significant attack vectors. Additionally, a time series of user reported information would also be required for piecing together the history of any reported incidents of fraud.

Trusted Third Parties: All of the current exchange audits have relied on some “reputable” trusted third party to make some type of verification. In the Coinbase audit [7], that was Andreas Antonopoulos, in the Kraken audit [8], that was Stefan Thomas. If we absolutely must rely on a trusted third party then some audit standards and procedures should ensure this weakness is fortified.

Limitations of Existing Fiat-pegging Systems

Here's a list of some of the common drawbacks and limitations of existing fiat-pegging systems.

- The systems are based on closed-source software, running on private, centralized databases, fundamentally no different than Paypal or any other existing mass-market retail/institutional asset trading/transfer/storage system.
- Decentralized systems that rely on altcoin blockchains which haven't been stress-tested, developed, or reviewed as closely as other blockchains, like Bitcoin.
- Pegging processes that rely on hedging derivative meta-assets, efficient market theory, or collateralization of the underlying asset, wherein liquidity, transferability, security, and other issues can exist.

- Lack of transparency and audits for the custodian, either crypto, fiat, or relating to their own internal ledgers (same as closed source and centralised databases).
- Reliance on legacy banking systems and trusted third parties (bank account owners) as a transfer and settlement mechanism for reserve assets.

Market Risk Examples

In the collateralization method, market risk exists because the price of the asset being used as collateral can move in an adverse direction to the price of the asset it's backing/pegging. This would cause the total value of the collateral to become less than the total value of the issued asset and make the system insolvent. This risk is mitigated by the custodian closing the position before this happens; that is, when the collateral price equals the pegged asset price then the collateral is liquidated (sold on the open market) and the position is closed. A great approach, with merit, and used in many liquid markets across the traditional banking and financial markets. However, as we saw from the global financial crisis, situations can arise in which the acceleration of such events causes a "liquidity crunch" and thus the collateral is unable to be liquidated fast enough to meet trading obligations, subsequently creating losses. With the cryptocurrency markets being so small and volatile, this type of event is much more likely. Additionally, the overall approach suffers from other liquidity and pricing constraints since there must be a sufficient supply of users posting collateral for the creation of the pegged-assets to exist in the first place.

In the derivatives approach, the price of the asset is pegged through entering one of several derivatives strategies, such as: swap strategies, covered and naked options strategies, various futures and forwards strategies. Each strategy has their own strengths and weaknesses, the discussion of which we won't engage in here. To summarize, each of these pegging processes themselves have similar "market risk" characteristics as the aforementioned collateralization method. It should be noted that the two methods are not mutually exclusive and often paired in a specific trading, hedging, or risk management function at legacy system financial institutions.

Finally, understand that we believe some combination of the above approaches may become a secure, reliable, and generally risk-free process for backing/pegging assets; however, at this point in time, this is not a direction we feel is feasible to take to ensure liquidity and price stability. Further, we believe that a reserve-based approach will always be in existence and complement these other approaches as the entire industry grows. As advances in technology continue, we will evaluate and incorporate any benefits available while maintaining the guarantee of 100% redeemability.

Legal and Compliance

Tether Limited (“Tether”) is a limited company incorporated pursuant to the Hong Kong Companies Ordinance. It is wholly owned by Tether Holdings Limited, a BVI business company incorporated pursuant to the BVI Business Companies Act, 2004.

Tether is registered as a Money Services Business with the Financial Crimes Enforcement Network of the U.S. Department of the Treasury (MSB Registration Number 31000058542968). Tether is establishing a relationship with a U.S. financial institution for purposes of better servicing Tether users in the United States.

Tether is concluding a principal–agency agreement with RenRenBee Limited (“RenRenBee”). RenRenBee is licensed as a Money Services Operator by the Hong Kong Customs and Excise Department (Licence No. 13-09-01265). Pursuant to the agreement, RenRenBee will provide anti-money laundering compliance work and customer due diligence procedures as agent for Tether as principal.

Through these and other measures, Tether is undertaking customer due diligence, record-keeping, and reporting procedures consistent with U.S. law and with the Hong Kong Anti-Money Laundering and Counter-Terrorist Financing (Financial Institutions) Ordinance.

Tether Limited currently has accounts with Cathay Bank and Hwatai Bank in Taiwan, both of whom are aware and confident that Tether’s business model is acceptable.

These banks are satisfied with our processes and also satisfied that our business operates in accordance with Taiwan off-shore banking regulations, as all of the banks had been requested to check this with their own legal, compliance and head-office before opening accounts (also at our own request). It was our goal from the beginning to have a compliant operation and to provide the maximum level of comfort to our banking partners here. In addition these banks have and are working with other Bitcoin based businesses.

Glossary of Terms

Digital currency: As defined by http://en.wikipedia.org/wiki/Digital_currency

Cryptocurrency or decentralized digital currency: any type of cryptocurrency that is open-source, cryptographically secure, and uses a distributed ledger. See: <http://en.wikipedia.org/wiki/Cryptocurrency>

Real-world currency, or fiat currency, or national/sovereign currency: all types of currency that are not cryptocurrencies as defined above.

Cryptocurrency system: A collection of software and processes primarily created to enable the existence of a cryptocurrency.

Legacy financial system: any financial system that is not a cryptocurrency system.

Utility-backed digital tokens, a.k.a Dapps: A decentralized digital token whose value is derived from the usefulness of its application rather than just being a value transfer system.

Asset-backed/pegged cryptocurrency: Any cryptocurrency whose price is pegged to a real-world asset, i.e. its not a “utility-backed” cryptocurrency.

Tether(s): a single unit (or multiple units) of fiat-pegged cryptocurrency issued by Tether Limited

TetherUSD or tUSD: a single unit of crypto-USD issued by Tether Limited

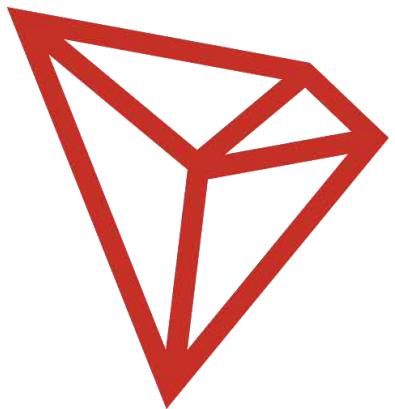
TUSD: collective amount of tUSD in circulation at any point in time.

Tether System: collectively refers to all process and technologies that enable tethers to exist

Proof of Reserves: The process by which the issuer of any asset-backed decentralized digital token, cryptographically/mathematically proves that all tokens that have been issued are fully reserved and backed by the underlying asset.

References

- [1] <https://www.thefinancialist.com/wp-content/uploads/2012/10/2012-GlobalWealthReport-.pdf>
- [2] <https://bitcoin.org/bitcoin.pdf>
- [3] http://www.deloitte.com/assets/Dcom-UnitedStates/Local%20Assets/Documents/FSI/us_fsi_BitcointheNewGoldRush_031814.pdf
- [4] <https://github.com/mastercoin-MSC/spec>
- [5] <http://unenumerated.blogspot.com/2005/12/bit-gold.html>
- [6] <https://iwilcox.me.uk/2014/proving-bitcoin-reserves>
- [7] <http://antonopoulos.com/2014/02/25/coinbase-review/>
- [8] <http://www.coindesk.com/krakens-audit-proves-holds-100-bitcoins-reserve/>



TRON

Advanced Decentralized Blockchain Platform

Whitepaper Version: 2.0

TRON Protocol Version: 3.2

TRON Foundation

December 10th, 2018, San Francisco
Page 78 of 414

1. Introduction	4
1.1 Vision	4
1.2 Background	4
1.3 History	5
1.4 Terminology	6
Address/Wallet	6
ABI	6
API	6
Asset	6
Bandwidth Points (BP)	6
Block	6
Block Reward	6
Block Header	6
Cold Wallet	7
DApp	7
gRPC	7
Hot Wallet	7
JDK	7
KhaosDB	7
LevelDB	7
Merkle Root	7
Public Testnet (Shasta)	8
RPC	8
Scalability	8
SUN	8
Throughput	8
Timestamp	8
TKC	8
TRC-10	8
TRX	8
2. Architecture	9
2.1 Core	10
2.2 Storage	10
2.2.1 Blockchain Storage	10
2.2.2 State Storage	10

2.3 Application	10
2.4 Protocol	11
2.4.1 Protocol Buffers	11
2.4.2 HTTP	11
2.5 TRON Virtual Machine (TVM)	11
2.6 Decentralized Exchange (DEX)	11
2.7 Implementation	12
3. Consensus	13
3.1 Delegated Proof of Stake (DPoS)	13
4. Account	16
4.1 Types	16
4.2 Creation	16
4.3 Structure	16
5. Block	18
5.1 Block Header	18
5.1.1 Raw Data	18
5.1.2 Witness Signature	19
5.1.3 Block ID	19
5.2 Transaction	19
5.2.1 Signing	19
5.2.2 Bandwidth Model	19
5.2.3 Fee	20
5.2.4 Transaction as Proof of Stake (TaPoS)	20
5.2.5 Transaction Confirmation	21
5.2.6 Structure	21
6. TRON Virtual Machine (TVM)	23
6.1 Introduction	23
6.2 Workflow	23
6.3 Performance	25
6.3.1 Lightweight Architecture	25
6.3.2 Robust	25
6.3.3 High Compatibility	25
6.3.4 Low Cost	25
7. Smart Contract	26
7.1 Introduction	26
7.2 Energy Model	26
7.3 Deployment	27

7.4 Trigger Function	27
7.5 TRON Solidity	27
8. Token	28
8.1 TRC-10 Token	28
8.2 TRC-20 Token	28
8.3 Beyond	29
9. Governance	30
9.1 Super Representative	30
9.1.1 General	30
9.1.2 Election	30
9.1.3 Reward	30
a. Vote Reward	30
b. Block Reward	31
c. Reward Calculation	31
9.2 Committee	32
9.2.1 General	32
9.2.2 Dynamic Network Parameters	32
9.2.3 Create Proposal	36
9.2.4 Vote Proposal	36
9.2.5 Cancel Proposal	36
9.3 Structure	36
10. DApp Development	37
10.1 APIs	37
10.2 Networks	37
10.3 Tools	37
10.4 Resources	37
11. Conclusion	39

1. Introduction

1.1 Vision

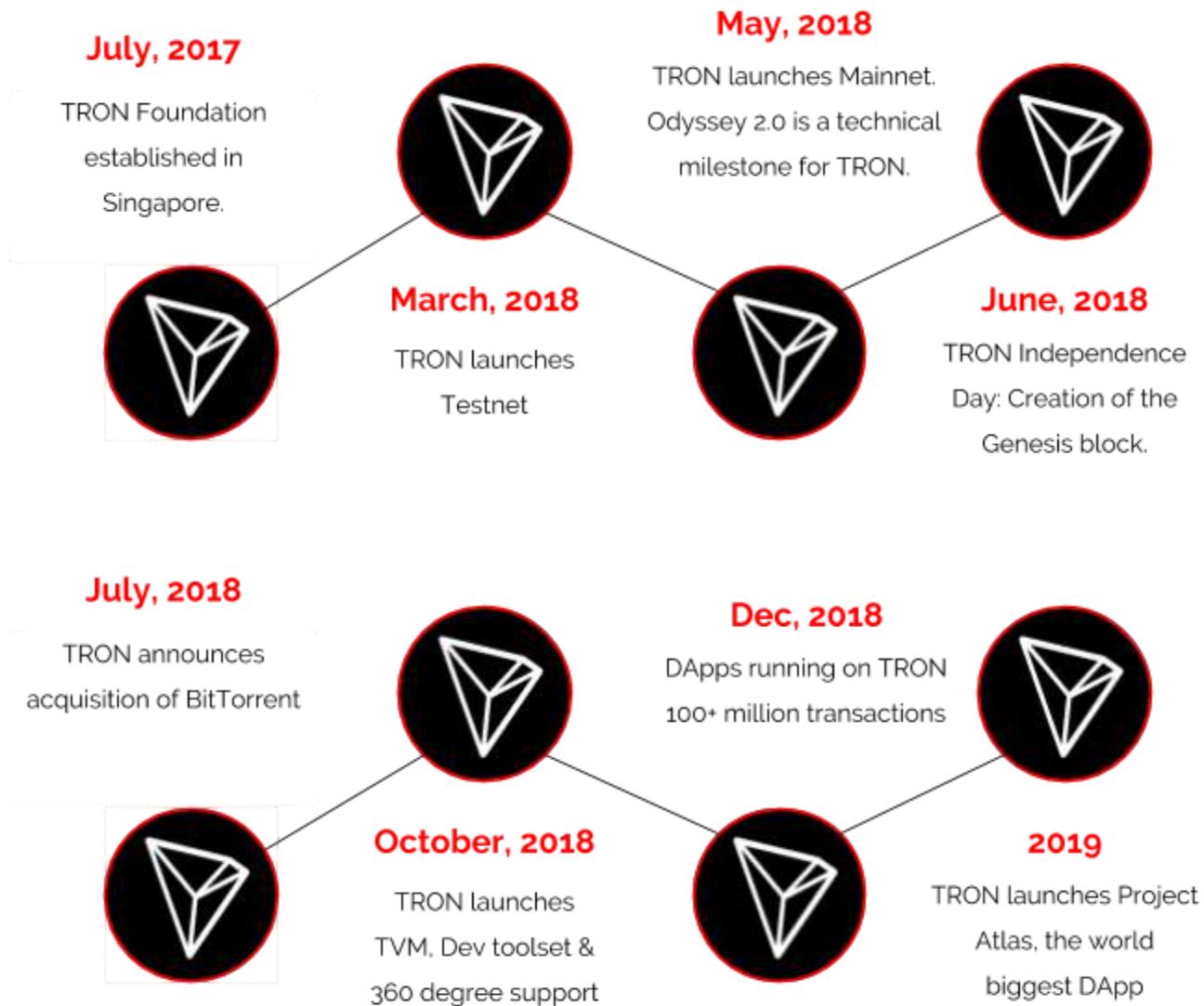
TRON is an ambitious project dedicated to the establishment of a truly decentralized Internet and its infrastructure. The TRON Protocol, one of the largest blockchain-based operating systems in the world, offers public blockchain support of high throughput, high scalability, and high availability for all Decentralized Applications (DApps) in the TRON ecosystem. The July 2018 acquisition of BitTorrent further cemented TRON's leadership in pursuing a decentralized ecosystem.

1.2 Background

The introduction of Bitcoin in 2009 revolutionized society's perception of the traditional financial system in the wake of the Great Recession (2007-2008). As centralized hedge funds and banks collapsed from speculation in opaque financial derivatives, blockchain technology provided a transparent universal ledger from which anybody could glean transaction information. The transactions were cryptographically secured using a Proof of Work (PoW) consensus mechanism, thus preventing double spend issues.

In late 2013, the Ethereum white paper proposed a network in which smart contracts and a Turing-complete Ethereum Virtual Machine (EVM) would allow developers to interact with the network through DApps. However, as transaction volumes in Bitcoin and Ethereum peaked in 2017, it was apparent from the low transaction throughput times and high transaction fees that cryptocurrencies like Bitcoin and Ethereum in their existing state were not scalable for widespread adoption. Thus, TRON was founded and envisioned as an innovative solution to these pressing scalability challenges.

1.3 History



The TRON Foundation was established in July 2017 in Singapore. In December 2017, TRON had launched its open source protocol. The Testnet, Blockchain Explorer, and Web Wallet were all launched by March 2018. TRON Mainnet launched shortly afterward in May 2018, marking the Odyssey 2.0 release as a technical milestone. In June 2018, TRON declared its independence with the creation of the Genesis block, along with the July 2018 acquisition of BitTorrent. In October 2018, TRON launched the TRON Virtual Machine (TVM), a complete developers' toolset, and 360 support system. The TRON roadmap involves combining BitTorrent's 100 million users with the TRON network via Project Atlas, as well as fostering the developer community to launch exciting new DApps on the TRON network¹.

¹ V1.0 is available at https://tron.network/static/doc/white_paper_v_1_0.pdf

1.4 Terminology

Address/Wallet

An address or wallet consisting of account credentials on the TRON network are generated by a key pair, which consists of a private key and a public key, the latter being derived from the former through an algorithm. The public key is usually used for session key encryption, signature verification, and encrypting data that could be decrypted by a corresponding private key.

ABI

An application binary interface (ABI) is an interface between two binary program modules; usually one of these modules is a library or an operating system facility, and the other is a user run program.

API

An application programming interface (API) is mainly used for user clients development. With API support, token issuance platforms can also be designed by developers themselves.

Asset

In TRON's documents, asset is the same as token, which is also denoted as TRC-10 token.

Bandwidth Points (BP)

To keep the network operating smoothly, TRON network transactions use BP as fuel. Each account gets 5000 free daily BP and more can be obtained by freezing TRX for BP. Both TRX and TRC-10 token transfers are normal transactions costing BP. Smart contract deployment and execution transactions consume both BP and Energy.

Block

Blocks contain the digital records of transactions. A complete block consists of the magic number, block size, block header, transaction counter, and transaction data.

Block Reward

Block production rewards are sent to a sub-account (address/wallet). Super Representatives can claim their rewards on Tronscan or through the API directly.

Block Header

A block header is part of a block. TRON block headers contain the previous block's hash, the Merkle root, timestamp, version, and witness address.

Cold Wallet

Cold wallet, also known as offline wallet, keeps the private key completely disconnected from any network. Cold wallets are usually installed on "cold" devices (e.g. computers or mobile phones staying offline) to ensure the security of TRX private key.

DApp

Decentralized Application is an App that operates without a centrally trusted party. An application that enables direct interaction/agreements/communication between end users and/or resources without a middleman.

gRPC

gRPC² (gRPC Remote Procedure Calls) is an open source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. Most common usage scenarios include connecting services in microservices style architecture and connecting mobile devices, and browser clients to backend services.

Hot Wallet

Hot wallet, also known as online wallet, allows user's private key to be used online, thus it could be susceptible to potential vulnerabilities or interception by malicious actors.

JDK

Java Development Kit is the Java SDK used for Java applications. It is the core of Java development, comprising the Java application environment (JVM+Java class library) and Java tools.

KhaosDB

TRON has a KhaosDB in the full-node memory that can store all the newly-forked chains generated within a certain period of time and supports witnesses to switch from their own active chain swiftly into a new main chain. See 2.2.2 State Storage for more details.

LevelDB

LevelDB was initially adopted with the primary goal to meet the requirements of fast R/W and rapid development. After launching the Mainnet, TRON upgraded its database to an entirely customized one catered to its very own needs. See 2.2.1 Blockchain Storage for more details.

Merkle Root

A Merkle root is the hash of all hashes of all transactions included as part of a block in a blockchain network. See 3.1 Delegated Proof of Stake (DPoS) for more details.

² <https://en.wikipedia.org/wiki/GRPC>

Public Testnet (Shasta)

A version of the network running in a single-node configuration. Developers can connect and test features without worrying about the economic loss. Testnet tokens have no value and anyone can request more from the public faucet.

RPC³

In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

Scalability

Scalability is a feature of the TRON Protocol. It is the capability of a system, network, or process to handle a growing amount of work or its potential to be enlarged to accommodate that growth.

SUN

SUN replaced drop as the smallest unit of TRX. 1 TRX = 1,000,000 SUN.

Throughput

High throughput is a feature of TRON Mainnet. It is measured in Transactions Per Second (TPS), namely the maximum transaction capacity in one second.

Timestamp

The approximate time of block production is recorded as Unix timestamp, which is the number of milliseconds that have elapsed since 00:00:00 01 Jan 1970 UTC.

TKC

Token configuration.

TRC-10

A standard of crypto token on TRON platform. Certain rules and interfaces are required to follow when holding an initial coin offering on TRON blockchain.

TRX

TRX stands for Tronix, which is the official cryptocurrency of TRON.

³ https://en.wikipedia.org/wiki/Remote_procedure_call

2. Architecture

TRON adopts a 3-layer architecture divided into Storage Layer, Core Layer, and Application Layer. The TRON protocol adheres to Google Protobuf, which intrinsically supports multi-language extension.

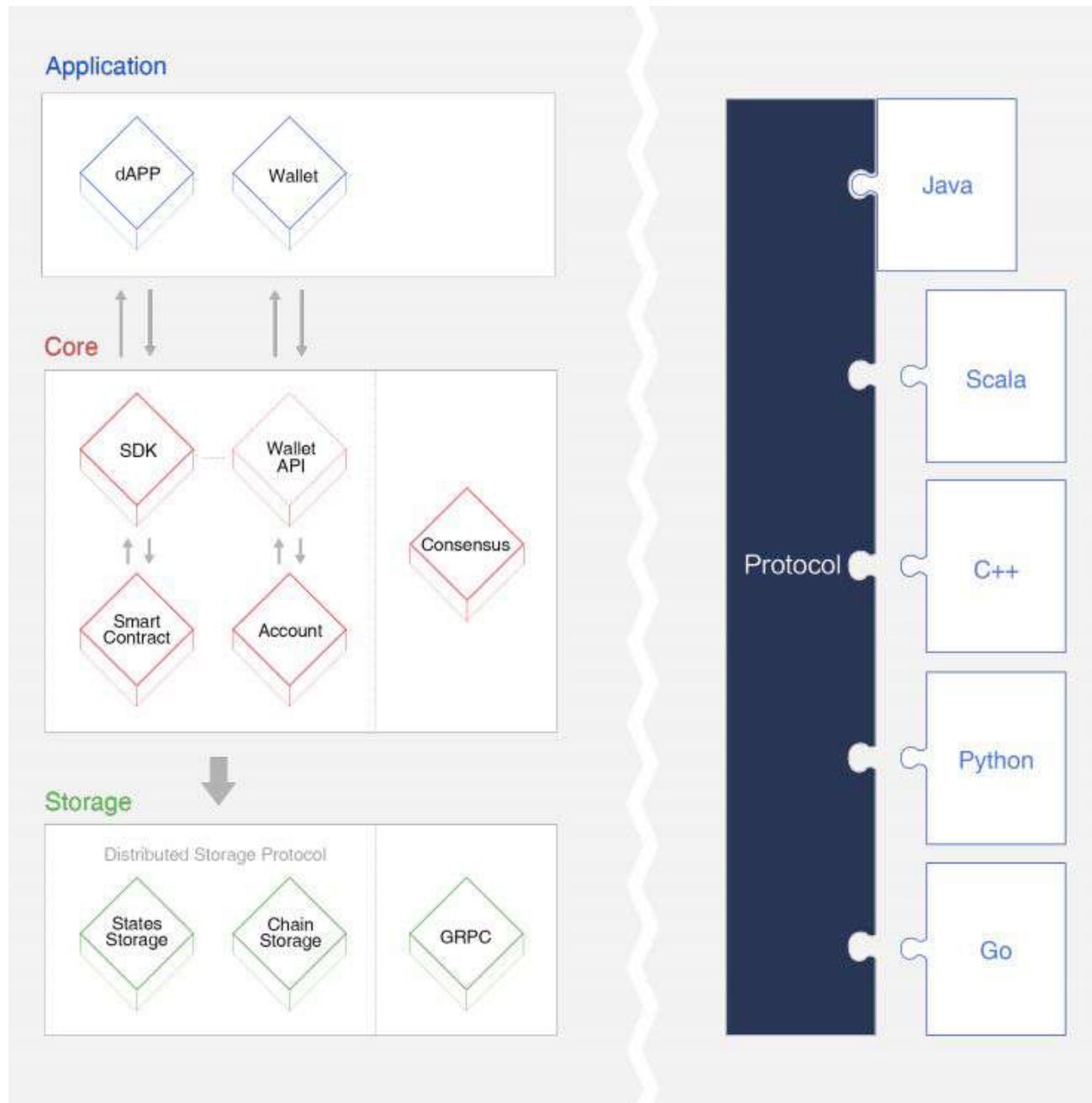


Figure 1: TRON 3-layer Architecture

2.1 Core

There are several modules in the core layer, including smart contracts, account management, and consensus. A stack-based virtual machine is implemented on TRON and an optimized instruction set is used. In order to better support DApp developers, Solidity⁴ was chosen as the smart contract language, followed by future support of other advanced languages. In addition, TRON's consensus mechanism is based on Delegated Proof of Stake (DPoS) and many innovations were made in order to meet its unique requirements.

2.2 Storage

TRON designed a unique distributed storage protocol consisting of Block Storage and State Storage. The notion of a graph database was introduced into the design of the storage layer to better meet the need for diversified data storage in the real world.

2.2.1 Blockchain Storage

TRON blockchain storage chooses to use LevelDB, which is developed by Google and proven successful with many companies and projects. It has high performance and supports arbitrary byte arrays as both keys and values, singular get, put and delete, batched put and delete, bi-directional iterators, and simple compression using the very fast Snappy algorithm.

2.2.2 State Storage

TRON has a KhaosDB in the full-node memory that can store all the newly forked chains generated within a certain period of time and supports witnesses to switch from their own active chain swiftly into a new main chain. It can also protect blockchain storage by making it more stable from being terminating abnormally in an intermediate state.

2.3 Application

Developers can create a diverse range of DApps and customized wallets on TRON. Since TRON enables smart contracts to be deployed and executed, the opportunities of utility applications are unlimited.

⁴ Solidity official documentation: <https://solidity.readthedocs.io/>

2.4 Protocol

TRON protocol adheres to Google Protocol Buffers⁵, which is a language-neutral, platform-neutral, and extensible way of serializing structured data for use in communications protocols, data storage, and more.

2.4.1 Protocol Buffers

Protocol Buffers (Protobuf) is a flexible, efficient, automated mechanism for serializing structured data, similar to JSON or XML, but much smaller, faster and simpler.

Protobuf (.proto) definitions can be used to generate code for C++, Java, C#, Python, Ruby, Golang, and Objective-C languages through the official code generators. Various third-party implementations are also available for many other languages. Protobuf eases development for clients by unifying the API definitions and also optimizing data transfers. Clients can take the API .proto from TRON's protocol repository and integrate through the automatically-generated code libraries.

As a comparison, Protocol Buffers is 3 to 10 times smaller and 20 to 100 times faster than XML, with less ambiguous syntax. Protobuf generates data access classes that are easier to use programmatically.

2.4.2 HTTP

TRON Protocol provides a RESTful HTTP API alternative to the Protobuf API. They share the same interface but the HTTP API can be readily used in javascript clients.

2.5 TRON Virtual Machine (TVM)

The TVM is a lightweight, Turing complete virtual machine developed for TRON's ecosystem. The TVM connects seamlessly with the existing development ecosystem to provide millions of global developers with a custom-built blockchain system that is efficient, convenient, stable, secure, and scalable.

2.6 Decentralized Exchange (DEX)

⁵ Google Protocol Buffers official documentation: <https://developers.google.com/protocol-buffers/>

The TRON network natively supports decentralized exchange functions. A decentralized exchange consists of multiple trading pairs. A trading pair (notation “Exchange”) is an Exchange Market between TRC-10 tokens, or between a TRC-10 token and TRX. Any account can create a trading pair between any tokens, even if the same pair already exists on the TRON network. Trading and price fluctuations of the trading pairs follow the Bancor Protocol⁶. The TRON network stipulates that the weights of the two tokens in all trading pairs are equal, so the ratio of their balances is the price between them. For example, consider a trading pair containing two tokens, ABC and DEF. ABC has a balance of 10 million and DEF has a balance of 1 million. Since their weights are equal, $10 \text{ ABC} = 1 \text{ DEF}$. This means that the ratio of ABC to DEF is 10 ABC per DEF.

2.7 Implementation

The TRON blockchain code is implemented in Java and was originally a fork from EthereumJ.

⁶ Bancor Protocol official website: <https://about.bancor.network/protocol/>

3. Consensus

3.1 Delegated Proof of Stake (DPoS)

The earliest consensus mechanism is the Proof of Work (PoW) consensus mechanism. This protocol is currently implemented in Bitcoin⁷ and Ethereum⁸. In PoW systems, transactions broadcast through the network are grouped together into nascent blocks for miner confirmation. The confirmation process involves hashing transactions using cryptographic hashing algorithms until a merkle root has been reached, creating a merkle tree:

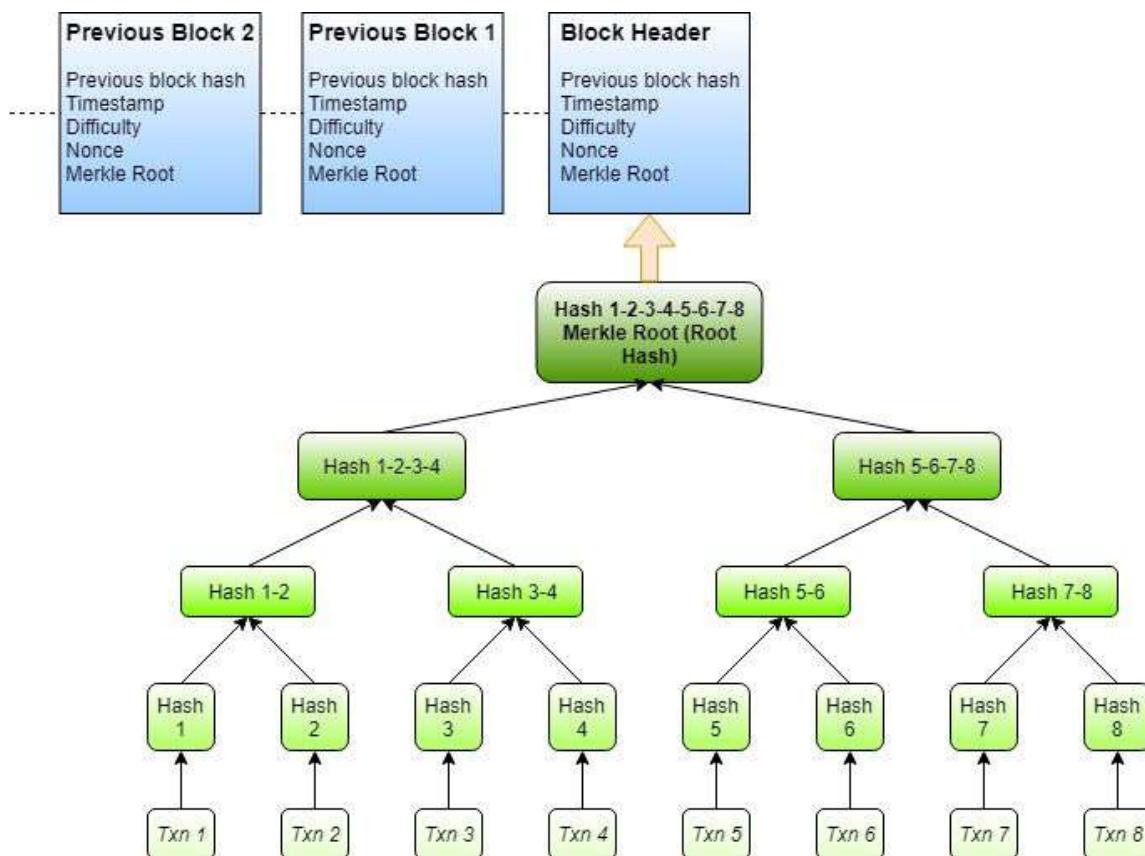


Figure 2: 8 TRX transactions are hashed into the merkle root. This merkle root is then included in the block header, which is attached to the previously confirmed blocks to form a blockchain. This allows for easy and transparent tracking of transactions, timestamps, and other related information.

⁷ Bitcoin whitepaper: <https://bitcoin.org/bitcoin.pdf>

⁸ Ethereum whitepaper: <https://github.com/ethereum/wiki/wiki/White-Paper>

Cryptographic hashing algorithms are useful in network attack prevention because they possess several properties⁹:

- **Input/Output length size** - The algorithm can pass in an input of any length in size, and outputs a fixed length hash value.
- **Efficiency** - The algorithm is relatively easy and fast to compute.
- **Preimage resistance** - For a given output z , it is impossible to find any input x such that $h(x) = z$. In other words, the hashing algorithm $h(x)$ is a one-way function in which only the output can be found, given an input. The reverse is not possible.
- **Collision resistance** - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$. In other words, the probability of finding two different inputs hashing to the same output is extremely low. This property also implies *second preimage resistance*.
- **Second preimage resistance** - Given x_1 , and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$. While this property is similar to *collision resistance*, the property differs in that it is saying an attacker with a given x , will find it computationally infeasible to find any x_2 hashing to the same output.
- **Deterministic** - maps each input to one and only one output.
- **Avalanche effect** - a small change in the input results in an entirely different output.

These properties give the cryptocurrency network its intrinsic value by ensuring attacks do not compromise the network. When miners confirm a block, they are rewarded tokens as a built-in incentive for network participation. However, as the global cryptocurrency market capitalization steadily increased, the miners became centralized and focused their computing resources on hoarding tokens as assets, rather than for network participation purposes. CPU miners gave way to GPUs, which in turn gave way to powerful ASICs. In one notable study, the total power consumption of Bitcoin mining has been estimated to be as high as 3 GW¹⁰, comparable to Ireland's power consumption. This same study projected total power consumption to reach 8 GW in the near future.

To solve the energy waste issue, the Proof of Stake (PoS) consensus mechanism was proposed by many new networks. In PoS networks, token holders lock their token balances to become block validators. The validators take turns proposing and voting on the next block. However, the problem with standard PoS is that validator influence correlates directly to the amount of tokens locked up. This results in parties hoarding large amounts of the network's base currency wielding undue influence in the network ecosystem.

The TRON consensus mechanism uses an innovative Delegated Proof of Stake system in which 27 Super Representatives (SRs) produce blocks for the network. Every 6 hours, TRX account holders who freeze their accounts can vote for a selection of SR candidates, with the top 27 candidates deemed the SRs. Voters may choose SRs based on criteria such as projects sponsored by SRs to

⁹ PAAR, C., PELZL, J., *Understanding Cryptography: A Textbook for Students and Practitioners*, 2010 ed. Springer-Verlag Berlin Heidelberg, 2010.

¹⁰ <https://www.sciencedirect.com/science/article/pii/S2542435118301776>

increase TRX adoption, and rewards distributed to voters. This allows for a more democratized and decentralized ecosystem. SRs' accounts are normal accounts, but their accumulation of votes allows them to produce blocks. With the low throughput rates of Bitcoin and Ethereum due to their PoW consensus mechanism and scalability issues, TRON's DPoS system offers an innovative mechanism resulting in 2000 TPS compared to Bitcoin's 3 TPS and Ethereum's 15 TPS.

The TRON protocol network generates one block every three seconds, with each block awarding 32 TRX to Super Representatives. A total of 336,384,000 TRX will be awarded annually to the 27 SRs. Each time an SR finishes block production, rewards are sent to a sub-account in the super-ledger. SRs can check, but not directly make use of these TRX tokens. A withdrawal can be made by each SR once every 24 hours, transferring the rewards from the sub-account to the specified SR account.

The three types of nodes on the TRON network are Witness Node, Full Node, and Solidity Node. Witness nodes are set up by SRs and are mainly responsible for block production and proposal creation/voting. Full nodes provide APIs and broadcast transactions and blocks. Solidity nodes sync blocks from other Full Nodes and also provide indexable APIs.

4. Account

4.1 Types

The three types of accounts in the TRON network are regular accounts, token accounts, and contract accounts.

1. Regular accounts are used for standard transactions.
2. Token accounts are used for storing TRC-10 tokens.
3. Contract accounts are smart contract accounts created by regular accounts and can be triggered by regular accounts as well.

4.2 Creation

There are three ways to create a TRON account:

1. Create a new account through API
2. Transfer TRX into a new account address
3. Transfer any TRC-10 token into a new account address

An offline key-pair consisting of an address (public key) and a private key, and not recorded by the TRON network, can also be generated. The user address generation algorithm consists of generating a key-pair and then extracting the public key (64-byte byte array representing x, y coordinates). Hash the public key using the SHA3-256 function (the SHA3 protocol adopted is KECCAK-256) and extract the last 20 bytes of the result. Add 41 to the beginning of the byte array and ensure the initial address length is 21 bytes. Hash the address twice using SHA3-256 function and take the first 4 bytes as verification code. Add the verification code to the end of the initial address and obtain the address in base58check format through base58 encoding. An encoded Mainnet address begins with T and is 34 bytes in length.

4.3 Structure

The three different account types are Normal, AssetIssue, and Contract. An Account contains 7 parameters:

1. **account_name**: the name for this account – e.g. BillsAccount.
2. **type**: what type of this account is – e.g. 0 (stands for type ‘Normal’).
3. **balance**: balance of this account – e.g. 4213312.

4. **vote**: received votes on this account – e.g. {("0x1b7w...9xj3",323), ("0x8djq...j12m",88),...,("0x82nd...mx6i",10001)}.
5. **asset**: other assets expected TRX in this account – e.g. {<"WishToken", 66666>, <"Dogie", 233>}.
6. **latest_operation_time**: the latest operation time of this account.

Protobuf data structure:

```
message Account {  
    message Vote {  
        bytes vote_address = 1;  
        int64 vote_count = 2;  
    }  
    bytes account_name = 1;  
    AccountType type = 2;  
    bytes address = 3;  
    int64 balance = 4;  
    repeated Vote votes = 5;  
    map<string, int64> asset = 6;  
    int64 latest_operation_time = 10;  
}
```

```
enum AccountType {  
    Normal = 0;  
    AssetIssue = 1;  
    Contract = 2;  
}
```

5. Block

A block typically contains a block header and several transactions.

Protobuf data structure:

```
message Block {  
    BlockHeader block_header = 1;  
    repeated Transaction transactions = 2;  
}
```

5.1 Block Header

A block header contains **raw_data**, **witness_signature**, and **blockID**.

Protobuf data structure:

```
message BlockHeader {  
    message raw {  
        int64 timestamp = 1;  
        bytes txTrieRoot = 2;  
        bytes parentHash = 3;  
        uint64 number = 4;  
        uint64 version = 5;  
        bytes witness_address = 6;  
    }  
    bytes witness_signature = 2;  
    bytes blockID = 3;  
}
```

5.1.1 Raw Data

Raw data is denoted as **raw_data** in Protobuf. It contains the raw data of a message, containing 6 parameters:

1. **timestamp**: timestamp of this message – e.g. 1543884429000.
2. **txTrieRoot**: the Merkle Tree's Root – e.g. 7dacsa...3ed.
3. **parentHash**: the hash of the last block – e.g. 7dacsa...3ed.
4. **number**: the block height – e.g. 4638708.
5. **version**: reserved – e.g. 5.

6. **witness_address**: the address of the witness packed in this block – e.g. 41928c...4d21.

5.1.2 Witness Signature

Witness signature is denoted as **witness_signature** in Protobuf, which is the signature for this block header from the witness node.

5.1.3 Block ID

Block ID is denoted as **blockID** in Protobuf. It contains the atomic identification of a block. A Block ID contains 2 parameters:

1. **hash**: the hash of block.
2. **number**: the hash and height of the block.

5.2 Transaction

5.2.1 Signing

TRON's transaction signing process follows a standard ECDSA cryptographic algorithm, with a SECP256K1 selection curve. A private key is a random number, and the public key is a point on the elliptic curve. The public key generation process consists of first generating a random number as a private key, and then multiplying the base point of the elliptic curve by the private key to obtain the public key. When a transaction occurs, the transaction raw data is first converted into byte format. The raw data then undergoes SHA-256 hashing. The private key corresponding to the contract address then signs the result of the SHA256 hash. The signature result is then added to the transaction.

5.2.2 Bandwidth Model

Ordinary transactions only consume bandwidth points, but smart contract operations consume both energy and bandwidth points. There are two types of bandwidth points available. Users can gain bandwidth points from freezing TRX, while 5000 free bandwidth points are also available daily.

When a TRX transaction is broadcast, it is transmitted and stored in the form of a byte array over the network. Bandwidth Points consumed by one transaction = number of transaction bytes multiplied by bandwidth points rate. For example, if the byte array length of a transaction is 200, then the transaction consumes 200 bandwidth points. However, if a TRX or token transfer results in the target account being created, then only the bandwidth points consumed to create the account will be deducted, and additional bandwidth points will not be deducted. In an account creation scenario, the network will first consume the bandwidth points that the transaction initiator gained

from freezing TRX. If this amount is insufficient, then the network consumes the transaction initiator's TRX.

In standard TRX transfer scenarios from one TRX account to another, the network first consumes the bandwidth points gained by the transaction initiator for freezing TRX. If that is insufficient, it then consumes from the free 5000 daily bandwidth points. If that is still not enough, then the network consumes the TRX of the transaction initiator. The amount is calculated by the number of bytes in the transaction multiplied by 10 SUN. Thus, for most TRX holders who may not necessarily freeze their TRX to participate in SR voting, the first step is automatically skipped (since TRX balance frozen = 0) and the 5000 daily free bandwidth powers the transaction.

For TRC-10 token transfers, the network first verifies whether the total free bandwidth points of the issued token asset are sufficient. If not, the bandwidth points obtained from freezing TRX are consumed. If there is still not enough bandwidth points, then it consumes the TRX of the transaction initiator.

5.2.3 Fee

TRON network generally does not charge fees for most transactions, however, due to system restrictions and fairness, bandwidth usage and transactions do take in certain fees.

Fee charges are broken down into the following categories:

1. Normal transactions cost bandwidth points. Users can use the free daily bandwidth points (5000) or freeze TRX to obtain more. When bandwidth points are not enough, TRX will be used directly from the sending account. The TRX needed is the number of bytes * 10 SUN.
2. Smart contracts cost energy (Section 6) but will also need bandwidth points for the transaction to be broadcasted and confirmed. The bandwidth cost is the same as above.
3. All query transactions are free. It doesn't cost energy or bandwidth.

TRON network also defines a set of fixed fees for the following transactions:

1. Creating a witness node: 9999 TRX
2. Issuing a TRC-10 token: 1024 TRX
3. Creating a new account: 0.1 TRX
4. Creating an exchange pair: 1024 TRX

5.2.4 Transaction as Proof of Stake (TaPoS)

TRON uses TaPoS to ensure the transactions all confirm the main blockchain, while making it difficult to forge counterfeit chains. In TaPoS, the networks require each transaction include part of the hash of a recent block header. This requirement prevents transactions from being replayed on forks not including the referenced block, and also signals the network that a particular user and their

stake are on a specific fork. This consensus mechanism protects the network against Denial of Service, 51%, selfish mining, and double spend attacks.

5.2.5 Transaction Confirmation

A transaction is included in a future block after being broadcast to the network. After 19 blocks are mined on TRON (including its own block), the transaction is confirmed. Each block is produced by one of the top 27 Super Representatives in a round robin fashion. Each block takes ~3 seconds to be mined on the blockchain. Time may slightly vary for each Super Representative due to network conditions and machine configurations. In general, a transaction is considered fully confirmed after ~1 minute.

5.2.6 Structure

Transaction APIs consist of the following functions:

```
message Transaction {
    message Contract {
        enum ContractType {
            AccountCreateContract = 0; // Create account/wallet
            TransferContract = 1; // Transfer TRX
            TransferAssetContract = 2; // Transfer TRC10 token
            VoteWitnessContract = 4; // Vote for Super Representative (SR)
            WitnessCreateContract = 5; // Create a new SR account
            AssetIssueContract = 6; // Create a new TRC10 token
            WitnessUpdateContract = 8; // Update SR information
            ParticipateAssetIssueContract = 9; // Purchase TRC10 token
            AccountUpdateContract = 10; // Update account/wallet information
            FreezeBalanceContract = 11; // Freeze TRX for bandwidth or energy
            UnfreezeBalanceContract = 12; // Unfreeze TRX
            WithdrawBalanceContract = 13; // Withdraw SR rewards, once per day
            UnfreezeAssetContract = 14; // Unfreeze TRC10 token
            UpdateAssetContract = 15; // Update a TRC10 token's information
            ProposalCreateContract = 16; // Create a new network proposal by any SR
            ProposalApproveContract = 17; // SR votes yes for a network proposal
            ProposalDeleteContract = 18; // Delete a network proposal by owner
            CreateSmartContract = 30; // Deploy a new smart contract
            TriggerSmartContract = 31; // Call a function on a smart contract
            GetContract = 32; // Get an existing smart contract
            UpdateSettingContract = 33; // Update a smart contract's parameters
            ExchangeCreateContract = 41; // Create a token trading pair on DEX
            ExchangeInjectContract = 42; // Inject funding into a trading pair
        }
    }
}
```

```
ExchangeWithdrawContract = 43; // Withdraw funding from a trading pair
ExchangeTransactionContract = 44; // Perform token trading
UpdateEnergyLimitContract = 45; // Update origin_energy_limit on a
smart contract
}
}
}
```

6. TRON Virtual Machine (TVM)

6.1 Introduction

TRON Virtual Machine (TVM) is a lightweight, Turing complete virtual machine developed for the TRON's ecosystem. Its goal is to provide a custom-built blockchain system that is efficient, convenient, stable, secure and scalable.

TVM initially forked from EVM¹¹ and can connect seamlessly with the existing solidity smart contract development ecosystem. Based on that, TVM additionally supports DPoS consensus.

TVM employs the concept of Energy. Different from the Gas mechanism on EVM, operations of transactions and smart contracts on TVM are free, with no TRX consumed. Technically, executable computation capacity on TVM is not restricted by total holding amount of tokens.

6.2 Workflow

The compiler first translates the Solidity smart contract into bytecode readable and executable on the TVM. The TVM then processes data through opcode, which is equivalent to operating the logic of a stack-based finite state machine. Finally, the TVM accesses blockchain data and invokes External Data Interface through the Interoperation layer.

¹¹ EVM: Ethereum Virtual Machine (<https://github.com/ethereum/ethereumj>)

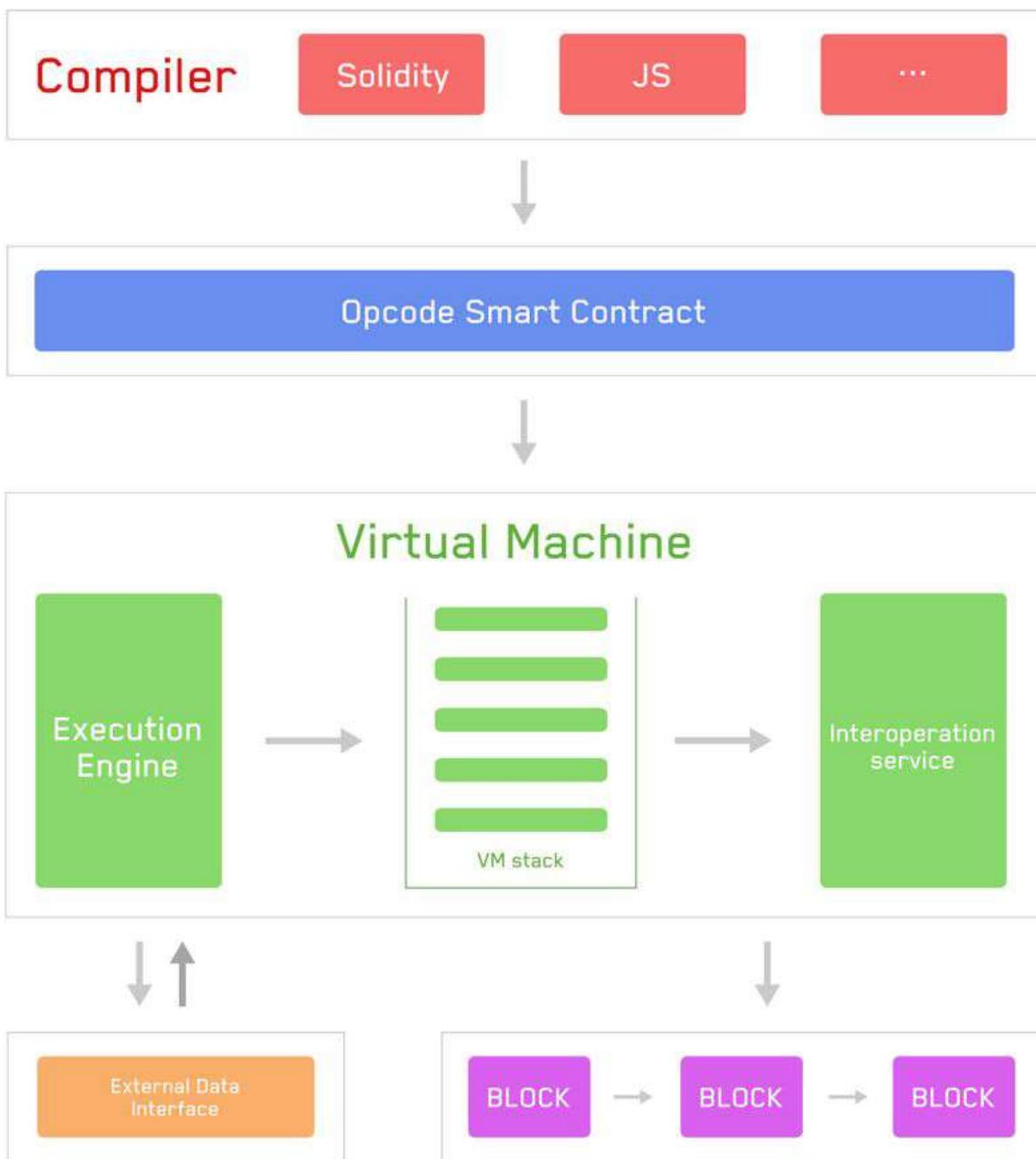


Figure 3: TVM Workflow

6.3 Performance

6.3.1 Lightweight Architecture

TVM adopts a lightweight architecture with the aim of reducing resource consumption to guarantee system performance.

6.3.2 Robust

TRX transfers and smart contract execution cost bandwidth points only, instead of TRX, which exempts TRON from being attacked. Bandwidth consumption is predictable and static since each computational step cost is fixed.

6.3.3 High Compatibility

TVM is compatible with EVM and will be compatible with more mainstream VMs in the future. Thereby, all smart contracts on EVM are executable on TVM.

6.3.4 Low Cost

Due to TVM's bandwidth setup, development costs are reduced and developers can focus on the logic development of their contract code. TVM also offers all-in-one interfaces for contract deployment, triggering and viewing to offer the convenience for developers.

7. Smart Contract

7.1 Introduction

A smart contract is a protocol that digitally verifies contract negotiation. They define the rules and penalties related to an agreement and also automatically enforce those obligations. The smart contract code facilitates, verifies, and enforces the negotiation or performance of an agreement or transaction. From a tokenization perspective, smart contracts also facilitate automatic funds transfers between participating parties should certain criteria be met.

TRON smart contracts are written in the Solidity language. Once written and tested, they can be compiled into bytecode, then deployed onto the TRON network for the TRON Virtual Machine. Once deployed, smart contracts can be queried via their contract addresses. The contract Application Binary Interface (ABI) shows the contract's call functions and is used for interacting with the network.

7.2 Energy Model

The maximum energy limit for deploying and triggering a smart contract is a function of several variables:

- Dynamic energy from freezing 1 TRX is $50,000,000,000 \text{ (Total Energy Limit)} / \text{ (Total Energy Weight)}$
- Energy limit is the daily account energy limit from freezing TRX
- Remaining daily account energy from freezing TRX is calculated as Energy Limit - Energy Used
- Fee limit in TRX is set in smart contract deploy/trigger call
- Remaining usable TRX in the account
- Energy per TRX if purchased directly ($10 \text{ SUN} = 1 \text{ Energy}$) = 100,000, SRs can vote on adjustment

There are two consumption scenarios to calculate for maximum energy limit for deployment and trigger. The logic can be expressed as follows:

```
const R = Dynamic Energy Limit
const F = Daily account energy from freezing TRX
const E = Remaining daily account energy from freezing TRX
const L = Fee limit in TRX set in deploy/trigger call
const T = Remaining usable TRX in account
```

```

const C = Energy per TRX if purchased directly

// Calculate M, defined as maximum energy limit for deployment/trigger of
smart contract
if F > L*R
    let M = min(E+T*C, L*R)
else
    let M = E+T*C

```

7.3 Deployment

When a TRON solidity smart contract is compiled, the TRON Virtual Machine reads the compiled bytecode. The bytecode consists of a section for code deployment, contract code, and the Auxdata. The Auxdata is the source code's cryptographic fingerprint, used for verification. The deployment bytecode runs the constructor function and sets up the initial storage variables. The deployment code also calculates the contract code and returns it to the TVM. The ABI is a JSON file that describes a TRON smart contract's functions. This file defines the function names, their payability, the function return values, and their state mutability.

7.4 Trigger Function

Once the TRON smart contracts are deployed, their functions can be triggered individually either via TronStudio or through API calls. State-changing functions require Energy while read-only functions execute without Energy.

7.5 TRON Solidity

TRON Solidity is a fork from Ethereum's Solidity language. TRON modifies the original project to support TRX and SUN units (1 TRX = 1,000,000 SUN). The rest of the language syntax is compatible with Solidity ^0.4.24. Thus the Tron Virtual Machine (TVM) is almost 100% compatible with EVM instructions.

8. Token

8.1 TRC-10 Token

In the TRON network, each account can issue tokens at the expense of 1024 TRX. To issue tokens, the issuer needs to specify a token name, the total capitalization, the exchange rate to TRX, circulation duration, description, website, maximum bandwidth consumption per account, total bandwidth consumption, and the amount of token frozen. Each token issuance can also configure each account's maximum daily token transfer Bandwidth Points, the entire network's maximum daily token transfer Bandwidth Points, total token supply, locking duration in days, and the total amount of tokens locked.

8.2 TRC-20 Token

TRC-20 is a technical standard used for smart contracts implementing tokens supported by the TRON Virtual Machine. It is fully compatible with ERC-20.

The interface is as follows:

```
contract TRC20Interface {  
    function totalSupply() public constant returns (uint);  
    function balanceOf(address tokenOwner) public constant returns (uint  
balance);  
    function allowance(address tokenOwner, address spender) public constant  
returns (uint remaining);  
    function transfer(address to, uint tokens) public returns (bool success);  
    function approve(address spender, uint tokens) public returns (bool  
success);  
    function transferFrom(address from, address to, uint tokens) public  
returns (bool success);  
  
    event Transfer(address indexed from, address indexed to, uint tokens);  
    event Approval(address indexed tokenOwner, address indexed spender, uint  
tokens);  
}
```

From a developer's perspective, there are several differences between TRC-10 and TRC-20. Some of the key differences are that TRC-10 tokens are accessible by APIs and smart contracts while TRC-20 tokens allow for interface customization but are only accessible within smart contracts.

From a cost perspective, TRC-10 tokens have transaction fees that are 1000 times lower than TRC-20, but carry bandwidth costs for API transfers and deposits. Transfers and deposits in smart contracts for TRC-10 tokens cost both bandwidth and energy.

8.3 Beyond

Since TRON uses the same Solidity version as Ethereum, more token standards could be readily ported to TRON.

9. Governance

9.1 Super Representative

9.1.1 General

Every account in the TRON network can apply and have the opportunity to become a Super Representative (denoted as SR). Everyone can vote for SR candidates. The top 27 candidates with the most votes will become SRs with the right and obligation to generate blocks. The votes are counted every 6 hours and the SRs will change accordingly.

To prevent malicious attacks, there is a cost to becoming an SR candidate. When applying, 9999 TRX will be burned from the applicant's account. Once successful, such account can join the SR election.

9.1.2 Election

TRON Power (denoted as TP) is needed to vote and the amount of TP depends on the voter's frozen assets (TRX).

TP is calculated in the following way:

$$1 \text{ TP} = 1 \text{ TRX frozen to get bandwidth}$$

Every account in the TRON network has the right to vote for their own SRs.

After the release (unfreeze, available after 3 days), users won't have any frozen assets and lose all TP accordingly. As a result, all votes become invalid for the ongoing and future voting round unless TRX is frozen again to vote.

Note that the TRON network only records the most recent vote, which means that every new vote will negate all previous votes.

9.1.3 Reward

a. Vote Reward

Also known as Candidate Reward, which the top 127 candidates updated once every round (6 hours) will share 115,200 TRX as mined. The reward will be split in accordance with the vote weight each candidate receives. Each year, the total reward for candidates will be 168,192,000 TRX.

Total vote reward per round

Why 115,200 TRX every round?

$$115,200 \text{ TRX} = \text{total vote reward per round (VR/round)}$$

$$VR/\text{round} = 16 \text{ TRX/block} \times 20 \text{ blocks/min} \times 60 \text{ mins/hr} \times 6 \text{ hrs/round}$$

Notice: this is set by WITNESS_STANDBY_ALLOWANCE = 115,200 TRX. See dynamic network parameters.

Total vote reward per year

Why 168,192,000 TRX every year?

$$168,192,000 \text{ TRX} = \text{total vote reward per year (VR/year)}$$

$$VR/\text{year} = 115,200 \text{ TRX/round} \times 4 \text{ rounds/day} \times 365 \text{ days/year}$$

b. Block Reward

Also known as Super Representative Reward, which the top 27 candidates (SRs) who are elected every round (6 hours) will share roughly 230,400 TRX as mined. The reward will be split evenly between the 27 SRs (minus the total reward blocks missed due to network error). A total of 336,384,000 TRX will be awarded annually to the 27 SRs.

Total block reward per round

Why 230,400 TRX every round?

$$230,400 \text{ TRX} = \text{total block reward per round (BR/round)}$$

$$BR/\text{round} = 32 \text{ TRX/bloc} \times 20 \text{ blocks/min} \times 60 \text{ mins/hr} \times 6 \text{ hrs/round}$$

Notice: the unit block reward is set by WITNESS_PAY_PER_BLOCK = 32 TRX. See dynamic network parameters.

Total block reward per year

Why 336,384,000 TRX every year?

$$336,384,000 \text{ TRX} = \text{total block reward per year (BR/year)}$$

$$BR/\text{year} = 230,400 \text{ TRX/round} \times 4 \text{ rounds/day} \times 365 \text{ days/year}$$

January 1, 2021

There will be no inflation on the TRON network before January 1, 2021, and the TRON Foundation will award all block rewards and candidate rewards prior to that date.

c. Reward Calculation

SR reward calculation

total reward = vote reward (VR) + block reward (BR)

VR = total VR × $\frac{\text{votes SR candidate received}}{\text{total votes}}$

BR = $\frac{\text{total BR}}{27}$ - block missed × 32

Note: the reward is calculated per SR per round (6 hours)

Rank 28 to rank 127 SR candidate reward calculation

total reward = vote reward (VR)

VR = total VR × $\frac{\text{votes SR candidate received}}{\text{total votes}}$

Note: the reward is calculated per SR candidate per round (6 hours)

9.2 Committee

9.2.1 General

The committee is used to modify TRON dynamic network parameters, such as block generation rewards, transaction fees, etc. The committee consists of the 27 SRs in the current round. Each SR has the right to propose and vote on proposals. When a proposal receives 19 votes or more, it is approved and the new network parameters will be applied in the next maintenance period (3 days).

9.2.2 Dynamic Network Parameters

0. MAINTENANCE_TIME_INTERVAL

a. Description

Modify the maintenance interval time in ms. Known as the SR vote interval time per round.

b. Example

[6 * 3600 * 1000] ms - which is 6 hours.

c. Range

[3 * 27* 1000, 24 * 3600 * 1000] ms

1. ACCOUNT_UPGRADE_COST

a. Description

Modify the cost of applying for SR account.

b. Example

[9,999,000,000] SUN - which is 9,999 TRX.

c. Range

[0,100 000 000 000 000 000] SUN

2. CREATE_ACCOUNT_FEE

a. Description

Modify the account creation fee.

- b. Example
[100,000] SUN - which is 1 TRX.
 - c. Range
[0,100 000 000 000 000 000] SUN
3. TRANSACTION_FEE
- a. Description
Modify the amount of fee used to gain extra bandwidth.
 - b. Example
[10] SUN/byte.
 - c. Range
[0,100 000 000 000 000 000] SUN/byte
4. ASSET_ISSUE_FEE
- a. Description
Modify asset issuance fee.
 - b. Example
[1024,000,000] SUN - which is 1024 TRX.
 - c. Range
[0,100 000 000 000 000 000] SUN
5. WITNESS_PAY_PER_BLOCK
- a. Description
Modify SR block generation reward. Known as unit block reward.
 - b. Example
[32,000,000] SUN - which is 32 TRX.
 - c. Range
[0,100 000 000 000 000 000] SUN
6. WITNESS_STANDBY_ALLOWANCE
- a. Description
Modify the rewards given to the top 127 SR candidates. Known as total vote reward per round.
 - b. Example
[115,200,000,000] SUN - which is 115,200 TRX.
 - c. Range
[0,100 000 000 000 000 000] SUN
7. CREATE_NEW_ACCOUNT_FEE_IN_SYSTEM_CONTRACT
- a. Description
Modify the cost of account creation. Combine dynamic network parameters #8 to get total account creation cost:
 $CREATE_NEW_ACCOUNT_FEE_IN_SYSTEM_CONTRACT \times CREATE_NEW_ACCOUNT_BANDWIDTH_RATE$
 - b. Example
[0] SUN.
 - c. Range
[0,100 000 000 000 000 000] SUN
8. CREATE_NEW_ACCOUNT_BANDWIDTH_RATE

a. Description

Modify the cost of account creation. Combine dynamic network parameters #7 to get total account creation cost:

CREATE_NEW_ACCOUNT_FEE_IN_SYSTEM_CONTRACT × CREATE_NEW_ACCOUNT_BANDWIDTH_RATE

b. Example

[1].

c. Range

[0,100,000,000,000,000,000]

9. ALLOW_CREATION_OF_CONTRACTS

a. Description

To turn on Tron Virtual Machine (TVM).

b. Example

True - set to activate and effect since 10/10/2018 23:47 UTC.

c. Range

True/False

10. REMOVE_THE_POWER_OF_THE_GR

a. Description

Remove the initial GR genesis votes

b. Example

True - effected at 11/4/2018 08:46 UTC.

c. Range

True/False - Notice: cannot set back to False from True.

11. ENERGY_FEE

a. Description

Modify the fee of 1 energy.

b. Example

20 SUN.

c. Range

[0,100 000 000 000 000 000] SUN

12. EXCHANGE_CREATE_FEE

a. Description

Modify the cost of trading pair creation. Known as the cost of creating a trade order.

b. Example

[1,024,000,000] SUN - which is 1024 TRX.

c. Range

[0,100 000 000 000 000 000] SUN

13. MAX_CPU_TIME_OF_ONE_TX

a. Description

Modify the maximum execution time of one transaction. Known as the timeout limit of one transaction.

b. Example

50 ms.

c. Range

[0, 1000] ms

14. ALLOW_UPDATE_ACCOUNT_NAME

a. Description

Modify the option to let an account update their account name.

b. Example

False - which is available to propose from java-tron Odyssey v3.2.

c. Range

True/False - Notice: cannot set back to False from True.

15. ALLOW_SAME_TOKEN_NAME

a. Description

Modify the validation of allowing different token have a duplicate name.

b. Example

False - which is available to propose from java-tron Odyssey v3.2.

c. Range

True/False - Notice: cannot set back to False from True.

16. ALLOW_DELEGATE_RESOURCE

a. Description

Modify the validation of allowing to issue token with a duplicate name, so the **tokenId** of the token, in long integer data type, would be the only atomic identification of a token.

b. Example

False - which is available to propose from java-tron Odyssey v3.2.

c. Range

True/False - Notice: cannot set back to False from True.

17. TOTAL_ENERGY_LIMIT

a. Description

Modify the whole network total energy limit.

b. Example

[50,000,000,000,000,000] SUN - which is 50,000,000,000 TRX.

c. Range

[0,100,000,000,000,000,000] SUN

18. ALLOW_TVM_TRANSFER_TRC10

a. Description

Allow TRC-10 token transfer within smart contracts.

ALLOW_UPDATE_ACCOUNT_NAME, ALLOW_SAME_TOKEN_NAME, ALLOW_DELEGATE_RESOURCE proposals must all be approved before proposing this parameter change.

b. Example

False - which is available to propose from java-tron Odyssey v3.2.

c. Range

True/False - Notice: cannot set back to False from True.

9.2.3 Create Proposal

Only the SR accounts have the rights to propose a change in dynamic network parameters.

9.2.4 Vote Proposal

Only committee members (SRs) can vote for a proposal and the member who does not vote in time will be considered as a disagree. The proposal is active for 3 days after it is created. The vote can be changed or retrieved during the 3-days voting window. Once the period ends, the proposal will either succeed (19+ votes) or fail (and end).

9.2.5 Cancel Proposal

The proposer can cancel the proposal before it becomes effective.

9.3 Structure

SRs are the witnesses of newly generated blocks. A witness contains 8 parameters:

1. **address**: the address of this witness – e.g. 0xu82h...7237.
2. **voteCount**: number of received votes on this witness – e.g. 234234.
3. **pubKey**: the public key for this witness – e.g. 0xu82h...7237.
4. **url**: the url for this witness – e.g. <https://www.noonetrust.com>.
5. **totalProduced**: the number of blocks this witness produced – e.g. 2434.
6. **totalMissed**: the number of blocks this witness missed – e.g. 7.
7. **latestBlockNum**: the latest height of block – e.g. 4522.
8. **isjobs**: a boolean flag.

Protobuf data structure:

```
message Witness{  
    bytes address = 1;  
    int64 voteCount = 2;  
    bytes pubKey = 3;  
    string url = 4;  
    int64 totalProduced = 5;  
    int64 totalMissed = 6;  
    int64 latestBlockNum = 7;  
    bool isJobs = 8;  
}
```

10. DApp Development

10.1 APIs

The TRON network offers a wide selection of over 60+ HTTP API gateways for interacting with the network via Full and Solidity Nodes. Additionally, TronWeb is a comprehensive JavaScript library containing API functions that enable developers to deploy smart contracts, change the blockchain state, query blockchain and contract information, trade on the DEX, and much more. These API gateways can be directed towards a local privatenet, the Shasta testnet, or the TRON Mainnet.

10.2 Networks

TRON has both a Shasta testnet as well as a Mainnet. Developers may connect to the networks by deploying nodes, interacting via TronStudio, or using APIs via the TronGrid service. The TronGrid service consists of load balanced node clusters hosted on AWS servers worldwide. As DApp development scales up and API call volumes increase, TronGrid successfully fields the increase in API traffic.

10.3 Tools

TRON offers a suite of development tools for enabling developers to create innovative DApps. TronBox is a framework that allows developers to test and deploy smart contracts via the TronWeb API. TronGrid is a load balanced and hosted API service that allows developers to access the TRON network without having to run their own node. TronGrid offers access to both the Shasta testnet as well as the TRON Mainnet. TronStudio is a comprehensive Integrated Development Environment (IDE) that enables developers to compile, deploy, and debug their Solidity smart contracts. TronStudio contains an internal full node that creates a private local environment for smart contract testing prior to deployment. The TronWeb API library connects developers to the network via a wide selection of HTTP API calls wrapped in JavaScript.

10.4 Resources

The TRON Developer Hub is a comprehensive API documentation¹² site tailored towards developers wishing to build on the TRON network. The Developer Hub provides a high-level conceptual understanding of TRON and walks users through the details of interacting with the

¹² Developer Hub: <https://developers.tron.network/>

network. The guides walk developers through node setup, deployment and interaction with smart contracts, API interaction and implementation, building sample DApps, and using each of the developer tools. Additionally, developer community channels are available through Discord¹³.

¹³ Discord: <https://discordapp.com/invite/GsRgsTD>

11. Conclusion

TRON is a scalable blockchain solution that has employed innovative methods for tackling challenges faced by legacy blockchain networks. Having reached over 2M transactions per day, with over 700K TRX accounts, and surpassing 2000 TPS, TRON has enabled the community in creating a decentralized and democratized network.

The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus

DAVID MAZIÈRES, Stellar Development Foundation

This paper introduces a new model for consensus called federated Byzantine agreement (FBA). FBA achieves robustness through quorum slices—individual trust decisions made by each node that together determine system-level quorums. Slices bind the system together much the way individual networks’ peering and transit decisions now unify the Internet.

We also present the Stellar Consensus Protocol (SCP), a construction for FBA. Like all Byzantine agreement protocols, SCP makes no assumptions about the rational behavior of attackers. Unlike prior Byzantine agreement models, which presuppose a unanimously accepted membership list, SCP enjoys open membership that promotes organic network growth. Compared to decentralized proof-of-work and proof-of-stake schemes, SCP has modest computing and financial requirements, lowering the barrier to entry and potentially opening up financial systems to new participants.

CCS Concepts: •**Security and privacy** → **Distributed systems security; Security protocols;**

Additional Key Words and Phrases: Byzantine fault tolerance, asynchronous systems

1. INTRODUCTION

Financial infrastructure is currently a mess of closed systems. Gaps between these systems mean that transaction costs are high [Provost 2013] and money moves slowly across political and geographic boundaries [Banning-Lover 2015; CGAP 2008]. This friction has curtailed the growth of financial services, leaving billions of people underserved financially [Demirguc-Kunt et al. 2015].

To solve these problems, we need financial infrastructure that supports the kind of organic growth and innovation we’ve seen from the Internet, yet still ensures the integrity of financial transactions. Historically, we have relied on high barriers to entry to ensure integrity. We trust established financial institutions and do our best to regulate them. But this exclusivity conflicts with the goal of organic growth. Growth demands new, innovative participants, who may possess only modest financial and computing resources.

We need a worldwide financial network open to anyone, so that new organizations can join and extend financial access to unserved communities. The challenge for such a network is ensuring participants record transactions correctly. With a low barrier to entry, users won’t trust providers to police themselves. With worldwide reach, providers won’t all trust a single entity to operate the network. A compelling alternative is a decentralized system in which participants together ensure integrity by agreeing on the validity of one another’s transactions. Such agreement hinges on a mechanism for worldwide consensus.

This paper presents federated Byzantine agreement (FBA), a model suitable for worldwide consensus. In FBA, each participant knows of others it considers important. It waits for the vast majority of those others to agree on any transaction before considering the transaction settled. In turn, those important participants do not agree to the transaction until the participants *they* consider important agree as well, and so on. Eventually, enough of the network accepts a transaction that it becomes infeasible for an attacker to roll it back. Only then do any participants consider the transaction settled. FBA’s consensus can ensure the integrity of a financial network. Its decentralized control can spur organic growth.

This paper further presents the Stellar consensus protocol (SCP), a construction for FBA. We prove that SCP’s safety is optimal for an asynchronous protocol, in that it guarantees agreement under any node-failure scenario that admits such a guarantee.

Draft of February 25, 2016

We also show that SCP is free from *blocked* states—in which consensus is no longer possible—unless participant failures make it impossible to satisfy trust dependencies. SCP is the first provably safe consensus mechanism to enjoy four key properties simultaneously:

- **Decentralized control.** Anyone is able to participate and no central authority dictates whose approval is required for consensus.
- **Low latency.** In practice, nodes can reach consensus at timescales humans expect for web or payment transactions—i.e., a few seconds at most.
- **Flexible trust.** Users have the freedom to trust any combination of parties they see fit. For example, a small non-profit may play a key role in keeping much larger institutions honest.
- **Asymptotic security.** Safety rests on digital signatures and hash families whose parameters can realistically be tuned to protect against adversaries with unimaginably vast computing power.

SCP has applications beyond financial markets for ensuring organizations perform important functions honestly. An example is certificate authorities (CAs), who literally hold the keys to the web. Experience shows that CAs sign incorrect certificates that get used in the wild [Microsoft 2013; Langley 2015]. Several proposals address this problem through certificate transparency [Kim et al. 2013; Laurie et al. 2013; Basin et al. 2014; Melara et al. 2014]. Certificate transparency allows users to examine the history of certificates issued for any given entity and detect attempts by CAs to change an entity’s public key without the endorsement of the previous key. SCP holds the potential to strengthen the indelible certificate history at the core of certificate transparency. Demanding global consensus on certificate history among a decentralized group of auditors would make it harder to backpedal and override previously issued certificates.

The next section discusses previous approaches to consensus. Section 3 defines federated Byzantine agreement (FBA) and lays out notions of safety and liveness applicable in the FBA model. Section 4 discusses optimal failure resilience in an FBA system, thereby establishing the security goals for SCP. Section 5 develops federated voting, a key building block of the SCP protocol. Section 6 presents SCP itself, proving safety and freedom from blocked states. Section 7 discusses limitations of SCP. Finally, Section 8 summarizes results. For readers less familiar with mathematical notation, Appendix A defines some symbols used throughout the paper.

2. RELATED WORK

Figure 1 summarizes how SCP differs from previous consensus mechanisms. The most famous decentralized consensus mechanism is the proof-of-work scheme advanced by Bitcoin [Nakamoto 2008]. Bitcoin takes a two-pronged approach to consensus. First, it provides incentives for rational actors to behave well. Second, it settles transactions through a proof-of-work [Dwork and Naor 1992] algorithm designed to protect against ill-behaved actors who do not possess the majority of the system’s computing power. Bitcoin has overwhelmingly demonstrated the appeal of decentralized consensus [Bonneau et al. 2015].

Proof of work has limitations, however. First, it wastes resources: by one estimate from 2014, Bitcoin might consume as much electric power as the entire country of Ireland [O’Dwyer and Malone 2014]. Second, secure transaction settlement suffers from expected latencies in the minutes or tens of minutes [Karame et al. 2012]. Finally, in contrast to traditional cryptographic protocols, proof of work offers no asymptotic security. Given non-rational attackers—or ones with extrinsic incentives to sabotage

mechanism	decentralized control	low latency	flexible trust	asymptotic security
proof of work	✓			
proof of stake	✓	maybe		maybe
Byzantine agreement		✓	✓	✓
Tendermint	✓	✓		✓
SCP (this work)	✓	✓	✓	✓

Fig. 1. Properties of different consensus mechanisms

consensus—small computational advantages can invalidate the security assumption, allowing history to be re-written in so-called “51% attacks.” Worse, attackers initially controlling less than 50% of computation can game the system to provide disproportionate rewards for those who join them [Eyal and Sirer 2013], thereby potentially gaining majority control. As the leading digital currency backed by the most computational power, Bitcoin enjoys a measure of protection against 51% attacks. Smaller systems have fallen victim [crazyearner 2013; Bradbury 2013], however, posing a problem for any proof-of-work system not built on the Bitcoin block chain.

An alternative to proof of work is proof of stake [King and Nadal 2012], in which consensus depends on parties that have posted collateral. Like proof of work, rewards encourage rational participants to obey the protocol; some designs additionally penalize bad behavior [Buterin 2014; Davarpanah et al. 2015]. Proof of stake opens the possibility of so-called “nothing at stake” attacks, in which parties that previously posted collateral but later cashed it in and spent the money can go back and rewrite history from a point where they still had stake. To mitigate such attacks, systems effectively combine proof of stake with proof of work—scaling down the required work in proportion to stake—or delay refunding collateral long enough for some other (sometimes informal) consensus mechanism to establish an irreversible checkpoint.

Still another approach to consensus is Byzantine agreement [Pease et al. 1980; Lamport et al. 1982], the best known variant of which is PBFT [Castro and Liskov 1999]. Byzantine agreement ensures consensus despite arbitrary (including non-rational) behavior on the part of some fraction of participants. This approach has two appealing properties. First, consensus can be fast and efficient. Second, trust is entirely decoupled from resource ownership, which makes it possible for a small non-profit to help keep more powerful organizations, such as banks or CAs, honest. Complicating matters, however, all parties must agree on the exact list of participants. Moreover, attackers must be prevented from joining multiple times and exceeding the system’s failure tolerance, a so-called Sybil attack [Douceur 2002]. BFT-CUP [Alchieri et al. 2008] accommodates unknown participants, but still presupposes a Sybil-proof centralized admission-control mechanism.

Generally, membership in Byzantine agreement systems is set by a central authority or closed negotiation. Prior attempts to decentralize admission have given up some of the benefits. One approach, taken by Ripple, is to publish a “starter” membership list that participants can edit for themselves, hoping people’s edits are either inconsequential or reproduced by an overwhelming fraction of participants. Unfortunately, because divergent lists invalidate safety guarantees [Schwartz et al. 2014], users are reluctant to edit the list in practice and a great deal of power ends up concentrated in the maintainer of the starter list. Another approach, taken by Tendermint [Kwon 2014], is to base membership on proof of stake. However, doing so once again ties trust to resource

ownership. SCP is the first Byzantine agreement protocol to give each participant maximum freedom in choosing which combinations of other participants to trust.

3. FEDERATED BYZANTINE AGREEMENT SYSTEMS

This section introduces the federated Byzantine agreement (FBA) model. Like non-federated Byzantine agreement, FBA addresses the problem of updating replicated state, such as a transaction ledger or certificate tree. By agreeing on what updates to apply, nodes avoid contradictory, irreconcilable states. We identify each update by a unique *slot* from which inter-update dependencies can be inferred. For instance, slots may be consecutively numbered positions in a sequentially applied log.

An FBA system runs a *consensus protocol* that ensures nodes agree on slot contents. A node v can safely apply update x in slot i when it has safely applied updates in all slots upon which i depends and, additionally, it believes all correctly functioning nodes will eventually agree on x for slot i . At this point, we say v has *externalized* x for slot i . The outside world may react to externalized values in irreversible ways, so a node cannot later change its mind about them.

A challenge for FBA is that malicious parties can join many times and outnumber honest nodes. Hence, traditional majority-based quorums do not work. Instead, FBA determines quorums in a decentralized way, by each node selecting what we call quorum slices. The next subsection defines quorums based on slices. The following subsection provides some examples and discussion. Finally, we define the key properties of safety and liveness that a consensus protocol should hope to achieve.

3.1. Quorum slices

In a consensus protocol, nodes exchange messages asserting statements about slots. We assume such assertions cannot be forged, which can be guaranteed if nodes are named by public key and they digitally sign messages. When a node hears a sufficient set of nodes assert a statement, it assumes no functioning node will ever contradict that statement. We call such a sufficient set a *quorum slice*, or, more concisely, just a *slice*. To permit progress in the face of node failures, a node may have multiple slices, any one of which is sufficient to convince it of a statement. At a high level, then, an FBA system consists of a loose confederation of nodes each of which has chosen one or more slices. More formally:

Definition (FBAS). A federated Byzantine agreement system, or *FBAS*, is a pair $\langle \mathbf{V}, \mathbf{Q} \rangle$ comprising a set of nodes \mathbf{V} and a quorum function $\mathbf{Q} : \mathbf{V} \rightarrow 2^{2^{\mathbf{V}}} \setminus \{\emptyset\}$ specifying one or more quorum slices for each node, where a node belongs to all of its own quorum slices—i.e., $\forall v \in \mathbf{V}, \forall q \in \mathbf{Q}(v), v \in q$. (Note 2^X denotes the powerset of X .)

Definition (quorum). A set of nodes $U \subseteq \mathbf{V}$ in FBAS $\langle \mathbf{V}, \mathbf{Q} \rangle$ is a *quorum* iff $U \neq \emptyset$ and U contains a slice for each member—i.e., $\forall v \in U, \exists q \in \mathbf{Q}(v)$ such that $q \subseteq U$.

A quorum is a set of nodes sufficient to reach agreement. A quorum slice is the subset of a quorum convincing one particular node of agreement. A quorum slice may be smaller than a quorum. Consider the four-node system in Figure 2, where each node has a single slice and arrows point to the other members of that slice. Node v_1 's slice $\{v_1, v_2, v_3\}$ is sufficient to convince v_1 of a statement. But v_2 's and v_3 's slices include v_4 , meaning neither v_2 nor v_3 can assert a statement without v_4 's agreement. Hence, no agreement is possible without v_4 's participation, and the only quorum including v_1 is the set of all nodes $\{v_1, v_2, v_3, v_4\}$.

Traditional, non-federated Byzantine agreement requires all nodes to accept the same slices, meaning $\forall v_1, v_2, \mathbf{Q}(v_1) = \mathbf{Q}(v_2)$. Because every member accepts every slice, traditional systems do not distinguish between slices and quorums. The downside is

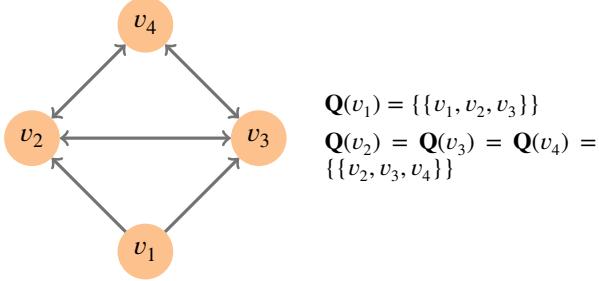
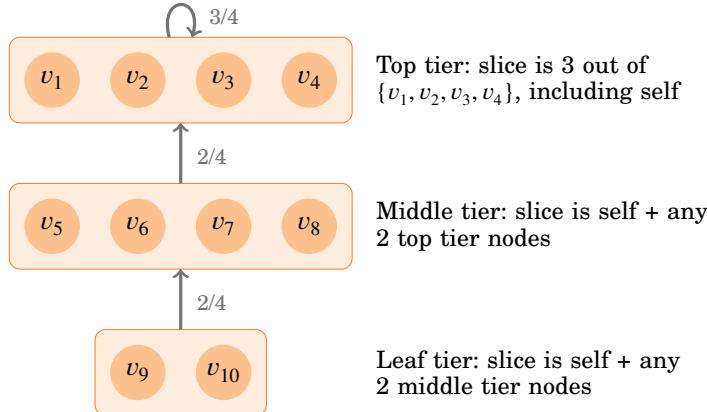
Fig. 2. v_1 's quorum slice is not a quorum without v_4 .

Fig. 3. Tiered quorum structure example

that membership and quorums must somehow be pre-ordained, precluding open membership and decentralized control. A traditional system, such as PBFT [Castro and Liskov 1999], typically has $3f + 1$ nodes, any $2f + 1$ of which constitute a quorum. Here f is the maximum number of Byzantine failures—meaning nodes acting arbitrarily—the system can survive.

FBA, introduced by this paper, generalizes Byzantine agreement to accommodate a greater range of settings. FBA's key innovation is enabling each node v to chose its own quorum slice set $Q(v)$. System-wide quorums thus arise from individual decisions made by each node. Nodes may select slices based on arbitrary criteria such as reputation or financial arrangements. In some settings, no individual node may have complete knowledge of all nodes in the system, yet consensus should still be possible.

3.2. Examples and discussion

Figure 3 shows an example of a tiered system in which different nodes have different slice sets, something possible only with FBA. A top tier, comprising v_1, \dots, v_4 , is structured like a PBFT system with $f = 1$, meaning it can tolerate one Byzantine failure so long as the other three nodes are reachable and well-behaved. Nodes v_5, \dots, v_8 constitute a middle tier and depend not on each other, but rather on the top tier. Only two top tier nodes are required to form a slice for a middle tier node. (The top tier assumes at most one Byzantine failure, so two top tier nodes cannot both fail unless the whole system has failed.) Nodes v_9 and v_{10} are in a leaf tier for which a slice consists of any

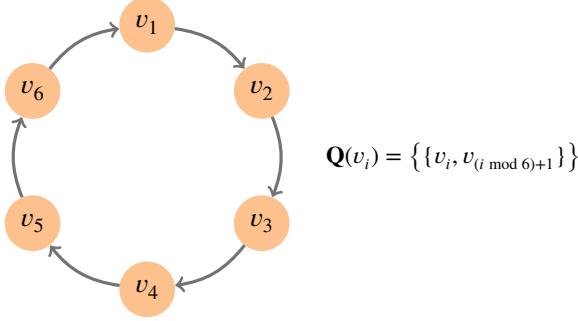


Fig. 4. Cyclic quorum structure example

two middle tier nodes. Note that v_9 and v_{10} may pick disjoint slices such as $\{v_5, v_6\}$ and $\{v_7, v_8\}$; nonetheless, both will indirectly depend on the top tier.

In practice, the top tier could consist of anywhere from four to dozens of widely known and trusted financial institutions. As the size of the top tier grows, there may not be exact agreement on its membership, but there will be significant overlap between most parties' notions of top tier. Additionally, one can imagine multiple middle tiers, for instance one for each country or geographic region.

This tiered structure resembles inter-domain network routing. The Internet today is held together by individual peering and transit relationships between pairs of networks. No central authority dictates or arbitrates these arrangements. Yet these pairwise relationships have sufficed to create a notion of *de facto* tier one ISPs [Norton 2010]. Though Internet reachability does suffer from firewalls, *transitive* reachability is nearly complete—e.g., a firewall might block The New York Times, but if it allows Google, and Google can reach The New York Times, then The New York Times is transitively reachable. Transitive reachability may be of limited utility for web sites, but it is crucial for consensus; the equivalent example would be Google accepting statements only if The New York Times does.

If we think of quorum slices as analogous to network reachability and quorums as analogous to transitive reachability, then the Internet's near complete transitive reachability suggests we can likewise ensure worldwide consensus with FBA. In many ways, consensus is an easier problem than inter-domain routing. While transit consumes resources and costs money, slice inclusion merely requires checking digital signatures. Hence, FBA nodes can err on the side of inclusiveness, constructing conservative slices with greater interdependence and redundancy than typically seen in peering and transit arrangements.

Another example not possible with centralized consensus is cyclic dependency structures, such as the one depicted in Figure 4. Such a cycle is unlikely to arise intentionally, but when individual nodes choose their own slices, it is possible for the overall system to end up embedding dependency cycles. The bigger point is that, compared to traditional Byzantine agreement, an FBA protocol must cope with a far wider variety of quorum structures.

3.3. Safety and liveness

We categorize nodes as either *well-behaved* or *ill-behaved*. A well-behaved node chooses sensible quorum slices (discussed further in Section 4.1) and obeys the protocol, including eventually responding to all requests. An ill-behaved node does not. Ill-behaved nodes suffer Byzantine failure, meaning they behave arbitrarily. For instance, an ill-

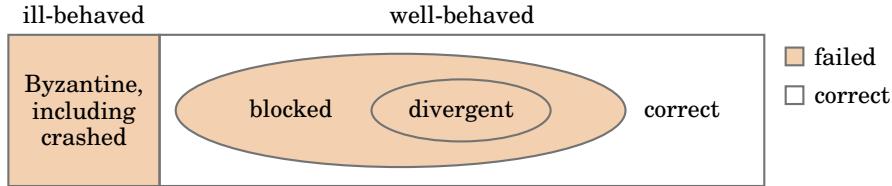


Fig. 5. Venn diagram of node failures

behaved node may be compromised, its owner may have maliciously modified the software, or it may have crashed.

The goal of Byzantine agreement is to ensure that well-behaved nodes externalize the same values despite the presence of such ill-behaved nodes. There are two parts to this goal. First, we would like to prevent nodes from diverging and externalizing different values for the same slot. Second, we would like to ensure nodes can actually externalize values, as opposed to getting blocked in some dead-end state from which consensus is no longer possible. We introduce the following two terms for these properties:

Definition (safety). A set of nodes in an FBAS enjoy *safety* if no two of them ever externalize different values for the same slot.

Definition (liveness). A node in an FBAS enjoys *liveness* if it can externalize new values without the participation of any failed (including ill-behaved) nodes.

We call well-behaved nodes that enjoy both safety and liveness *correct*. Nodes that are not correct have *failed*. All ill-behaved nodes have failed, but a well-behaved node can fail, too, by waiting indefinitely for messages from ill-behaved nodes, or, worse, by having its state poisoned by incorrect messages from ill-behaved nodes.

Figure 5 illustrates the possible kinds of node failure. To the left are Byzantine failures, meaning the ill-behaved nodes. To the right are two kinds of well-behaved but failed nodes. Nodes that lack liveness are termed *blocked*, while those that lack safety are termed *divergent*. An attack violating safety is strictly more powerful than one violating only liveness, so we classify divergent nodes as a subset of blocked ones.

Our definition of liveness is weak in that it says a node *can* externalize new values, not that it *will*. Hence, it admits a state of *perpetual preemption* in which consensus remains forever possible, yet the network continually thwarts it by delaying or re-ordering critical messages in just the wrong way. Perpetual preemption is inevitable in a purely asynchronous, deterministic system that survives node failure [Fischer et al. 1985]. Fortunately, preemption is transient. It does not indicate node failure, because the system can recover at any time. Protocols can mitigate the problem through randomness [Ben-Or 1983; Bracha and Toueg 1985] or through realistic assumptions about message latency [Dwork et al. 1988]. Latency assumptions are more practical when one would like to limit execution time or avoid the trusted dealers often required by more efficient Randomized algorithms [?]. Of course, only termination and not safety should depend upon message timing.

4. OPTIMAL RESILIENCE

Whether or not nodes enjoy safety and liveness depends on several factors: what quorum slices they have chosen, which nodes are ill-behaved, and of course the concrete consensus protocol and network behavior. As is common for asynchronous systems, we assume the network eventually delivers messages between well-behaved nodes, but can otherwise arbitrarily delay or reorder messages.

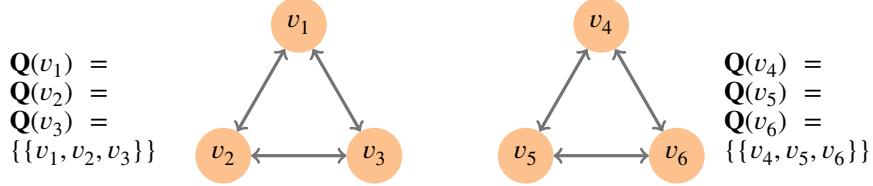
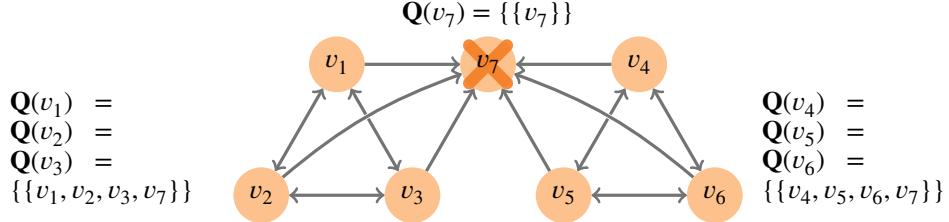


Fig. 6. FBAS lacking quorum intersection

Fig. 7. Ill-behaved node v_7 can undermine quorum intersection.

This section answers the following question: given a specific $\langle V, Q \rangle$ and particular subset of V that is ill-behaved, what are the best safety and liveness that any federated Byzantine agreement protocol can guarantee regardless of the network? We first discuss quorum intersection, a property without which safety is impossible to guarantee. We then introduce a notion of dispensable sets—sets of failed nodes in spite of which it is possible to guarantee both safety and liveness.

4.1. Quorum intersection

A protocol can guarantee agreement only if the quorum slices represented by function Q satisfy a validity property we call quorum intersection.

Definition (quorum intersection). An FBAS enjoys *quorum intersection* iff any two of its quorums share a node—i.e., for all quorums U_1 and U_2 , $U_1 \cap U_2 \neq \emptyset$.

Figure 6 illustrates a system lacking quorum intersection, where Q permits two quorums, $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$, that do not intersect. Disjoint quorums can independently agree on contradictory statements, undermining system-wide agreement. When many quorums exist, quorum intersection fails if any two do not intersect. For example, the set of all nodes $\{v_1, \dots, v_6\}$ in Figure 6 is a quorum that intersects the other two, but the system still lacks quorum intersection because the other two do not intersect each other.

No protocol can guarantee safety in the absence of quorum intersection, since such a configuration can operate as two different FBAS systems that do not exchange any messages. However, even with quorum intersection, safety may be impossible to guarantee in the presence of ill-behaved nodes. Compare Figure 6, in which there are two disjoint quorums, to Figure 7, in which two quorums intersect at a single node v_7 , and v_7 is ill-behaved. If v_7 makes inconsistent statements to the left and right quorums, the effect is equivalent to disjoint quorums.

In fact, since ill-behaved nodes contribute nothing to safety, no protocol can guarantee safety without the well-behaved nodes enjoying quorum intersection on their own. After all, in a worst-case scenario for safety, ill-behaved nodes can just always make any possible (contradictory) statement that completes a quorum. Two quorums overlapping only at ill-behaved nodes will again be able to operate like two different FBAS

systems thanks to the duplicity of the ill-behaved nodes. In short, FBAS $\langle V, Q \rangle$ can survive Byzantine failure by a set of nodes $B \subseteq V$ iff $\langle V, Q \rangle$ enjoys quorum intersection after deleting the nodes in B from V and from all slices in Q . More formally:

Definition (delete). If $\langle V, Q \rangle$ is an FBAS and $B \subseteq V$ is a set of nodes, then to *delete* B from $\langle V, Q \rangle$, written $\langle V, Q \rangle^B$, means to compute the modified FBAS $\langle V \setminus B, Q^B \rangle$ where $Q^B(v) = \{ q \setminus B \mid q \in Q(v) \}$.

It is the responsibility of each node v to ensure $Q(v)$ does not violate quorum intersection. One way to do so is to pick conservative slices that lead to large quorums. Of course, a malicious v may intentionally pick $Q(v)$ to violate quorum intersection. But a malicious v can also lie about the value of $Q(v)$ or ignore $Q(v)$ to make arbitrary assertions. In short, $Q(v)$'s value is not meaningful when v is ill-behaved. This is why the necessary property for safety—quorum intersection of well-behaved nodes after deleting ill-behaved nodes—is unaffected by the slices of ill-behaved nodes.

Suppose Figure 6 evolved from a three-node FBAS v_1, v_2, v_3 with quorum intersection to a six-node FBAS without. When v_4, v_5, v_6 join, they maliciously choose slices that violate quorum intersection and no protocol can guarantee safety for V . Fortunately, deleting the bad nodes to yield $\langle V, Q \rangle^{\{v_4, v_5, v_6\}}$ restores quorum intersection, meaning at least $\{v_1, v_2, v_3\}$ can enjoy safety. Note that deletion is conceptual, for the sake of describing optimal safety. A protocol should guarantee safety for v_1, v_2, v_3 without their needing to know that v_4, v_5, v_6 are ill-behaved.

4.2. Dispensable sets (DSets)

We capture the fault tolerance of nodes' slice selections through the notion of a *dispensable set* or *DSet*. Informally, the safety and liveness of nodes outside a DSet can be guaranteed regardless of the behavior of nodes inside the DSet. Put another way, in an optimally resilient FBAS, if a single DSet encompasses every ill-behaved node, it also contains every failed node, and conversely all nodes outside the DSet are correct. As an example, in a centralized PBFT system with $3f + 1$ nodes and quorum size $2f + 1$, any f or fewer nodes constitute a DSet. Since PBFT in fact survives up to f Byzantine failures, its robustness is optimal.

In the less regular example of Figure 3, $\{v_1\}$ is a DSet, since one top tier node can fail without affecting the rest of the system. $\{v_9\}$ is also a DSet because no other node depends on v_9 for correctness. $\{v_6, \dots, v_{10}\}$ is a DSet, because neither v_5 nor the top tier depend on any of those five nodes. $\{v_5, v_6\}$ is *not* a DSet, as it is a slice for v_9 and v_{10} and hence, if entirely malicious, can lie to v_9 and v_{10} and convince them of assertions inconsistent with each other or the rest of the system.

To prevent a misbehaving DSet from affecting the correctness of other nodes, two properties must hold. For safety, deleting the DSet cannot undermine quorum intersection. For liveness, the DSet cannot deny other nodes a functioning quorum. This leads to the following definition:

Definition (DSet). Let $\langle V, Q \rangle$ be an FBAS and $B \subseteq V$ be a set of nodes. We say B is a dispensable set, or *DSet*, iff:

- (1) (*quorum intersection despite B*) $\langle V, Q \rangle^B$ enjoys quorum intersection, and
- (2) (*quorum availability despite B*) Either $V \setminus B$ is a quorum in $\langle V, Q \rangle$ or $B = V$.

Quorum availability despite B protects against nodes in B refusing to answer requests and blocking other nodes' progress. Quorum intersection despite B protects against the opposite—nodes in B making contradictory assertions that enable other nodes to externalize inconsistent values for the same slot. Nodes must balance the two threats in slice selection. All else equal, bigger slices lead to bigger quorums with

well-behaved / ill-behaved	Local property of nodes, independent of other nodes (except for the validity of slice selection).
intact / befouled	Property of nodes given their quorum slices and a particular set of ill-behaved nodes. Befouled nodes are ill-behaved or depend, possibly indirectly, on too many ill-behaved nodes.
correct / failed	Property of nodes given their quorum slices, a concrete protocol, and actual network behavior. The goal of a consensus protocol is to guarantee correctness for all intact nodes.

Fig. 8. Key properties of FBAS nodes

greater overlap, meaning fewer failed node sets B will undermine quorum intersection when deleted. On the other hand, bigger slices are more likely to contain failed nodes, endangering quorum availability.

The smallest DSet containing all ill-behaved nodes may encompass well-behaved nodes as well, reflecting the fact that a sufficiently large set of ill-behaved nodes can cause well-behaved nodes to fail. For instance, in Figure 3, the smallest DSet containing v_5 and v_6 is $\{v_5, v_6, v_9, v_{10}\}$. The set of all nodes, V , is always a DSet, as an FBAS $\langle V, Q \rangle$ vacuously enjoys quorum intersection despite V and, by special case, also enjoys quorum availability despite V . The motivation for the special case is that given sufficiently many ill-behaved nodes, V may be the smallest DSet to contain all ill-behaved ones, indicating a scenario under which no protocol can guarantee anything better than complete system failure.

The DSets in an FBAS are determined *a priori* by the quorum function Q . Which nodes are well- and ill-behaved depends on runtime behavior, such as machines getting compromised. The DSets we care about are those that encompass all ill-behaved nodes, as they help us distinguish nodes that should be guaranteed correct from ones for which such a guarantee is impossible. To this end, we introduce the following terms:

Definition (intact). A node v in an FBAS is *intact* iff there exists a DSet B containing all ill-behaved nodes and such that $v \notin B$.

Definition (befouled). A node v in an FBAS is *befouled* iff it is not intact.

A befouled node v is surrounded by enough failed nodes to block its progress or poison its state, even if v itself is well-behaved. No FBAS can guarantee the correctness of a befouled node. However, an optimal FBAS guarantees that every intact node remains correct. Figure 8 summarizes the key properties of nodes. The following theorems facilitate analysis by showing that the set of befouled nodes is always a DSet in an FBAS with quorum intersection.

THEOREM 1. *Let U be a quorum in FBAS $\langle V, Q \rangle$, let $B \subseteq V$ be a set of nodes, and let $U' = U \setminus B$. If $U' \neq \emptyset$ then U' is a quorum in $\langle V, Q \rangle^B$.*

PROOF. Because U is a quorum, every node $v \in U$ has a $q \in Q(v)$ such that $q \subseteq U$. Since $U' \subseteq U$, it follows that every $v \in U'$ has a $q \in Q(v)$ such that $q \setminus B \subseteq U'$. Rewriting with deletion notation yields $\forall v \in U', \exists q \in Q^B(v)$ such that $q \subseteq U'$, which, because $U' \subseteq V \setminus B$, means that U' is a quorum in $\langle V, Q \rangle^B$. \square

THEOREM 2. *If B_1 and B_2 are DSets in an FBAS $\langle V, Q \rangle$ enjoying quorum intersection, then $B = B_1 \cap B_2$ is a DSet, too.*

PROOF. Let $U_1 = V \setminus B_1$ and $U_2 = V \setminus B_2$. If $U_1 = \emptyset$, then $B_1 = V$ and $B = B_2$ (a DSet), so we are done. Similarly, if $U_2 = \emptyset$, then $B = B_1$, and we are done. Otherwise, note

that by quorum availability despite DSets B_1 and B_2 , U_1 and U_2 are quorums in $\langle V, Q \rangle$. It follows from the definition that the union of two quorums is also a quorum. Hence $V \setminus B = U_1 \cup U_2$ is a quorum and we have quorum availability despite B .

We must now show quorum intersection despite B . Let U_a and U_b be any two quorums in $\langle V, Q \rangle^B$. Let $U = U_1 \cap U_2 = U_2 \setminus B_1$. By quorum intersection of $\langle V, Q \rangle$, $U = U_1 \cap U_2 \neq \emptyset$. But then by Theorem 1, $U = U_2 \setminus B_1$ must be a quorum in $\langle V, Q \rangle^{B_1}$. Now consider that $U_a \setminus B_1$ and $U_a \setminus B_2$ cannot both be empty, or else $U_a \setminus B = U_a$ would be. Hence, by Theorem 1, either $U_a \setminus B_1$ is a quorum in $(\langle V, Q \rangle^B)^{B_1} = \langle V, Q \rangle^{B_1}$, or $U_a \setminus B_2$ is a quorum in $(\langle V, Q \rangle^B)^{B_2} = \langle V, Q \rangle^{B_2}$, or both. In the former case, note that if $U_a \setminus B_1$ is a quorum in $\langle V, Q \rangle^{B_1}$, then by quorum intersection of $\langle V, Q \rangle^{B_1}$, $(U_a \setminus B_1) \cap U \neq \emptyset$; since $(U_a \setminus B_1) \cap U = (U_a \setminus B_1) \setminus B_2$, it follows that $U_a \setminus B_2 \neq \emptyset$, making $U_a \setminus B_2$ a quorum in $\langle V, Q \rangle^{B_2}$. By a similar argument, $U_b \setminus B_2$ must be a quorum in $\langle V, Q \rangle^{B_2}$. But then quorum intersection despite B_2 tells us that $(U_a \setminus B_2) \cap (U_b \setminus B_2) \neq \emptyset$, which is only possible if $U_a \cap U_b \neq \emptyset$. \square

THEOREM 3. *In an FBAS with quorum intersection, the set of befouled nodes is a DSet.*

PROOF. Let B_{\min} be the intersection of every DSet that contains all ill-behaved nodes. It follows from the definition of *intact* that a node v is intact iff $v \notin B_{\min}$. Thus, B_{\min} is precisely the set of befouled nodes. By Theorem 2, DSets are closed under intersection, so B_{\min} is also a DSet. \square

5. FEDERATED VOTING

This section develops a federated voting technique that FBAS nodes can use to agree on a statement. At a high level, the process for agreeing on some statement a involves nodes exchanging two sets of messages. First, nodes *vote* for a . Then, if the vote was successful, nodes *confirm* a , effectively holding a second vote on the fact that the first vote succeeded.

From each node's perspective, the two rounds of messages divide agreement on a statement a into three phases: unknown, accepted, and confirmed. (This pattern dates back to three-phase commit [Skeen and Stonebraker 1983].) Initially, a 's status is completely *unknown* to a node v — a could end up true, false, or even *stuck* in a permanently indeterminate state. If the first vote succeeds, v may come to *accept* a . No two intact nodes ever accept contradictory statements, so if v is intact and accepts a , then a cannot be false.

For two reasons, however, v accepting a does not suffice for v to act on a . First, the fact that v accepted a does not mean all intact nodes can; a could be stuck for other nodes. Second, if v is befouled, then accepting a means nothing— a may be false at intact nodes. Yet even if v is befouled—which v does not know—the system may still enjoy quorum intersection of well-behaved nodes, in which case, for optimal safety, v needs greater assurance of a . Holding a second vote addresses both problems. If the second vote succeeds, v moves to the *confirmed* phase in which it can finally deem a true and act on it.

The next few subsections detail the federated voting process. Because voting does not rule out the possibility of stuck statements, Section 5.6 discusses how to cope with them. Section 6 will turn federated voting into a consensus protocol that avoids the possibility of stuck slots for intact nodes.

5.1. Voting with open membership

A correct node in a Byzantine agreement system acts on a statement a only when it knows that other correct nodes will never agree to statements contradicting a . Most protocols employ voting for this purpose. Well-behaved nodes vote for a statement a only if it is valid. Well-behaved nodes also never change their votes. Hence, in centralized Byzantine agreement, it is safe to accept a if a quorum comprising a majority of well-behaved nodes has voted for it. We say a statement is *ratified* once it has received the necessary votes.

In a federated setting, we must adapt voting to accommodate open membership. One difference is that a quorum no longer corresponds to a majority of well-behaved nodes. However, the majority requirement primarily serves to ensure quorum intersection of well-behaved nodes, which Section 4.1 already adapted to FBA. Another implication of open membership is that nodes must discover what constitutes a quorum as part of the voting process. To implement quorum discovery, a protocol should specify $Q(v)$ in all messages from v .

Definition (vote). A node v votes for an (abstract) statement a iff

- (1) v asserts a is valid and consistent with all statements v has accepted, and
- (2) v asserts it has never voted against a —i.e., voted for a statement that contradicts a —and v promises never to vote against a in the future.

Definition (ratify). A quorum U_a ratifies a statement a iff every member of U_a votes for a . A node v ratifies a iff v is a member of a quorum U_a that ratifies a .

THEOREM 4. *Two contradictory statements a and \bar{a} cannot both be ratified in an FBAS that enjoys quorum intersection and contains no ill-behaved nodes.*

PROOF. By contradiction. Suppose quorum U_1 ratifies a and quorum U_2 ratifies \bar{a} . By quorum intersection, $\exists v \in U_1 \cap U_2$. Such a v must have illegally voted for both a and \bar{a} , violating the assumption of no ill-behaved nodes. \square

THEOREM 5. *Let $\langle V, Q \rangle$ be an FBAS enjoying quorum intersection despite B , and suppose B contains all ill-behaved nodes. Let v_1 and v_2 be two nodes not in B . Let a and \bar{a} be contradictory statements. If v_1 ratifies a then v_2 cannot ratify \bar{a} .*

PROOF. By contradiction. Suppose v_1 ratifies a and v_2 ratifies \bar{a} . By definition, there must exist a quorum U_1 containing v_1 that ratified a and quorum U_2 containing v_2 that ratified \bar{a} . By Theorem 1, since $U_1 \setminus B \neq \emptyset$ and $U_2 \setminus B \neq \emptyset$, both must be quorums in $\langle V, Q \rangle^B$, meaning they ratified a and \bar{a} respectively in $\langle V, Q \rangle^B$. But $\langle V, Q \rangle^B$ enjoys quorum intersection and has no ill-behaved nodes, so Theorem 4 tell us a and \bar{a} cannot both be ratified. \square

THEOREM 6. *Two intact nodes in an FBAS with quorum intersection cannot ratify contradictory statements.*

PROOF. Let B be the set of befouled nodes. By Theorem 3, B is a DSet. By the definition of DSet, $\langle V, Q \rangle$ enjoys quorum intersection despite B . By Theorem 5, two nodes not in B cannot ratify contradictory statements. \square

5.2. Blocking sets

In centralized consensus, liveness is an all-or-nothing property of the system. Either a unanimously well-behaved quorum exists, or else ill-behaved nodes can prevent the rest of the system from accepting new statements. In FBA, by contrast, liveness may differ across nodes. For instance, in the tiered quorum example of Figure 3, if middle

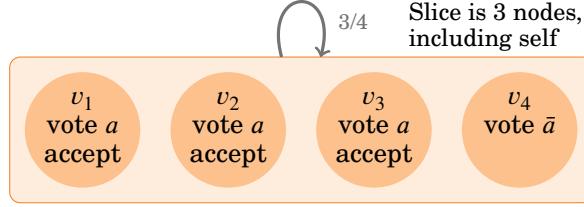


Fig. 9. v_4 voted for \bar{a} , which contradicts ratified statement a .

tier nodes v_6, v_7, v_8 crash, the leaf tier will be blocked while the top tier and node v_5 will continue to enjoy liveness.

An FBA protocol can guarantee liveness to a node v only if $Q(v)$ contains at least one quorum slice comprising only correct nodes. A set B of failed nodes can violate this property if B contains at least one member of each of v 's slices. We term such a set B v -blocking, because it has the power to block progress by v .

Definition (v -blocking). Let $v \in V$ be a node in FBAS $\langle V, Q \rangle$. A set $B \subseteq V$ is v -blocking iff it overlaps every one of v 's slices—i.e., $\forall q \in Q(v), q \cap B \neq \emptyset$.

THEOREM 7. Let $B \subseteq V$ be a set of nodes in FBAS $\langle V, Q \rangle$. $\langle V, Q \rangle$ enjoys quorum availability despite B iff B is not v -blocking for any $v \in V \setminus B$.

PROOF. “ $\forall v \in V \setminus B, B$ is not v -blocking” is equivalent to “ $\forall v \in V \setminus B, \exists q \in Q(v)$ such that $q \subseteq V \setminus B$.” By the definition of *quorum*, the latter holds iff $V \setminus B$ is a quorum or $B = V$, the exact definition of *quorum availability despite B* . \square

As a corollary, the DSet of befouled nodes is not v -blocking for any intact v .

5.3. Accepting statements

When an intact node v learns that it has ratified a statement, Theorem 6 tells v that other intact nodes will not ratify contradictory statements. This condition is sufficient for v to accept a , but we cannot make it necessary. Ratifying a statement requires voting for it, and some nodes may have voted for contradictory statements. In Figure 9, for example, v_4 votes for \bar{a} before learning that the other three nodes ratified the contradictory statement a . Though v_4 cannot now vote for a , we would still like it to accept a to be consistent with the other nodes.

A key insight is that if a node v is intact, then no v -blocking set B can consist entirely of befouled nodes. Now suppose B is a v -blocking set and every member of B claims to accept statement a . If v is intact, at least one member of B must be, too. The intact member will not lie about accepting a ; hence, a is true and v can accept it. Of course, if v is befouled, then a might not be true. But a befouled node can accept anything and vacuously not affect the correctness of intact nodes.

Definition (accept). An FBAS node v accepts a statement a iff it has never accepted a statement contradicting a and it determines that either

- (1) There exists a quorum U such that $v \in U$ and each member of U either voted for a or claims to accept a , or
- (2) Each member of a v -blocking set claims to accept a .

Though a well-behaved node cannot vote for contradictory statements, condition 2 above allows a node to *vote* for one statement and later *accept* a contradictory one.

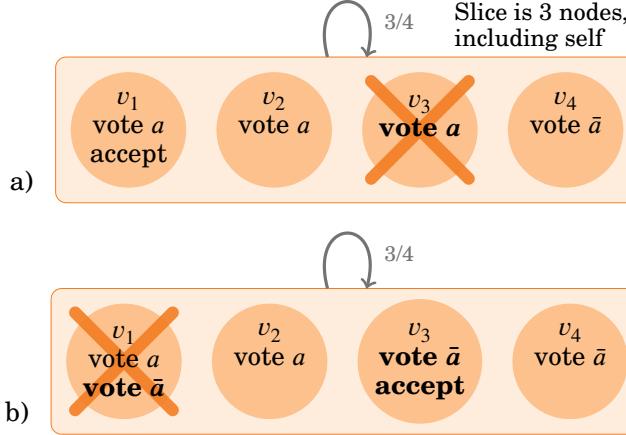


Fig. 10. Scenarios indistinguishable to v_2 when v_2 does not see bold messages

THEOREM 8. *Two intact nodes in an FBAS that enjoys quorum intersection cannot accept contradictory statements.*

PROOF. Let $\langle V, Q \rangle$ be an FBAS with quorum intersection and let B be its DSet of befouled nodes (which exists by Theorem 3). Suppose an intact node accepts statement a . Let v be the first intact node to accept a . At the point v accepts a , only befouled nodes in B can claim to accept it. Since by the corollary to Theorem 7, B cannot be v -blocking, it must be that v accepted a through condition 1. Thus, v identified a quorum U such that every node claimed to vote for or accept a , and since v is the first intact node to accept a , it must mean all nodes in $U \setminus B$ voted for a . In other words, v ratified a in $\langle V, Q \rangle^B$. Generalizing, any statement accepted by an intact node in $\langle V, Q \rangle$ must be ratified in $\langle V, Q \rangle^B$. Because B is a DSet, $\langle V, Q \rangle^B$ enjoys quorum intersection. Because additionally B contains all ill-behaved nodes, Theorem 4 rules out ratification of contradictory statements. \square

5.4. Accepting is not enough

Unfortunately, for nodes to assume the truth of accepted statements would yield sub-optimal safety and liveness guarantees in a federated consensus protocol. We discuss the issues with safety and liveness in turn. To provide some context, we then explain why these issues are thornier in FBA than in centralized Byzantine agreement.

5.4.1. Safety. Consider an FBAS $\langle V, Q \rangle$ in which the only quorum is unanimous consent—i.e., $\forall v, Q(v) = \{V\}$. This ought to be a conservative choice for safety—don’t do anything unless everyone agrees. Yet since every node is v -blocking for every v , any node can single-handedly convince any other node to accept arbitrary statements.

The problem is that accepted statements are only safe among intact nodes. But as discussed in Section 4.1, the only condition necessary to guarantee safety is quorum intersection of well-behaved nodes, which might hold even in the case that some well-behaved nodes are befouled. In particular, when $Q(v) = \{V\}$, the only DSets are \emptyset and V , meaning any node failure befouls the whole system. By contrast, quorum intersection holds despite every $B \subseteq V$.

5.4.2. Liveness. Another limitation of accepted statements is that other intact nodes may be unable to accept them. This possibility makes reliance on accepted statements

problematic for liveness. If a node proceeds to act on a statement because it accepted the statement, other nodes could be unable to proceed in a similar fashion.

Consider Figure 10a, in which node v_3 crashes after helping v_1 ratify and accept statement a . Though v_1 accepts a , v_2 and v_4 cannot. In particular, from v_2 's perspective, the situation depicted is indistinguishable from Figure 10b, in which v_3 voted for \bar{a} and is well-behaved but slow to respond, while v_1 is ill-behaved and sent v_3 a vote for \bar{a} (thereby causing v_3 to accept \bar{a}) while illegally also sending v_2 a vote for a .

To support a protocol-level notion of liveness in cases like Figure 10a, v_1 needs a way to ensure every other intact node can eventually accept a before v_1 acts on a . Once this is the case, it makes sense to say the system agrees on a .

Definition (agree). An FBAS $\langle V, Q \rangle$ *agrees* on a statement a iff, regardless of what subsequently transpires, once sufficient messages are delivered and processed, every intact node will accept a .

5.4.3. Comparison to centralized voting. To understand why the above issues arise in federated voting, consider a centralized Byzantine agreement system of N nodes with quorum size T . Such a system enjoys quorum availability with $f_L = N - T$ or fewer node failures. Since any two quorums share at least $2T - N$ nodes, quorum intersection of well-behaved nodes holds up to $f_S = 2T - N - 1$ Byzantine failures.

Centralized Byzantine agreement systems typically set $N = 3f + 1$ and $T = 2f + 1$ to yield $f_L = f_S = f$, the equilibrium point at which safety and liveness have the same fault tolerance. If safety is more important than liveness, some protocols increase T so that $f_S > f_L$ [Li and Mazières 2007]. In FBA, because quorums arise organically, systems are unlikely to find themselves at equilibrium, making it far more important to protect safety in the absence of liveness.

Now consider a centralized system in which, because of node failure and contradictory votes, some node v cannot ratify statement a that was ratified by other nodes. If v hears $f_S + 1$ nodes claim a was ratified, v knows that either one of them is well-behaved or all safety guarantees have collapsed. Either way, v can act on a with no loss of safety. The FBA equivalent would be to hear from a set B where B , if deleted, undermines quorum intersection of well-behaved nodes. Identifying such a B is hard for three reasons: one, quorums are discovered dynamically; two, ill-behaved nodes may lie about slices; and three, v does not know which nodes are well-behaved. Instead, we defined federated voting to accept a when a v -blocking set does. The v -blocking property has the advantage of being easily checkable, but is equivalent to hearing from $f_L + 1$ nodes in a centralized system when we really want $f_S + 1$.

To guarantee agreement among all well-behaved nodes in a centralized system, one merely needs $f_L + f_S + 1$ nodes to acknowledge that a statement was ratified. If more than f_L of them fail, we do not expect liveness anyway. If f_L or fewer fail, then we know $f_S + 1$ nodes remain willing to attest to ratification, which will in turn convince all other well-behaved nodes. The reliance on f_S has no easy analogue in the FBA model. Interestingly, however, $f_L + f_S + 1 = T$, the quorum size, suggesting a similar approach might work with a more complex justification.

Put another way, at some point nodes need to believe a statement strongly enough to depend on its truth for safety. A centralized system offers two ways to reach this point for a statement a : ratify a first-hand, or reason backwards from $f_S + 1$ nodes claiming a was ratified, figuring safety is hopeless if they have all lied. FBA lacks the latter approach; the only tool it has for safety among well-behaved nodes is first-hand ratification. Since nodes still need a way to overcome votes against ratified statements, we introduced a notion of accepting, but it provides a weaker consistency guarantee limited to intact nodes.

5.5. Statement confirmation

Both limitations of accepted statements stem from complications when a set of intact nodes S votes against a statement a that is nonetheless ratified. Particularly in light of FBA's non-uniform quorums, S may prevent some intact node from ever ratifying v . To provide v a means of accepting a despite votes against it, the definition of *accept* has a second criterion based on v -blocking sets. But the second criterion is weaker than ratification, offering no guarantees to befouled nodes that enjoy quorum intersection.

Now suppose a statement a has the property that no intact node ever votes against it. Then we have no need to accept a and can instead insist that nodes directly ratify a before acting on it. We call such statements *irrefutable*.

Definition (irrefutable). A statement a is *irrefutable* in an FBAS if no intact node can ever vote against it.

Theorem 8 tells us that two intact nodes cannot accept contradictory statements. Thus, while some intact nodes may vote against a statement a that was accepted by an intact node, the statement “an intact node accepted a ” is irrefutable. This suggests holding a second vote to ratify the fact that an intact node accepted a .

Definition (confirm). A quorum U_a in an FBAS *confirms* a statement a iff $\forall v \in U_a, v$ claims to accept a . A node *confirms* a iff it is in such a quorum.

Nodes express that they have accepted statement a by stating “*accept(a)*,” an abbreviation of the statement, “An intact node accepted a .” To confirm a means to ratify *accept(a)*. A well-behaved node v can vote for *accept(a)* only after accepting a , as v cannot assume any particular other nodes are intact. If v itself is befouled, *accept(a)* might be false, in which case voting for it may cost v liveness, but a befouled node has no guarantee of liveness anyway.

The next theorem shows that nodes can rely on confirmed statements without losing optimal safety. Theorem 11 then shows that confirmed statements meet the definition of *agreement* from Section 5.4.2, meaning nodes can rely on confirmed statements without endangering the liveness of intact nodes.

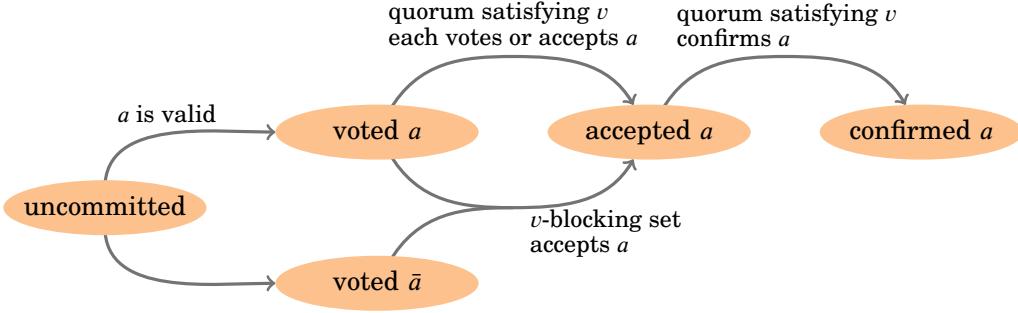
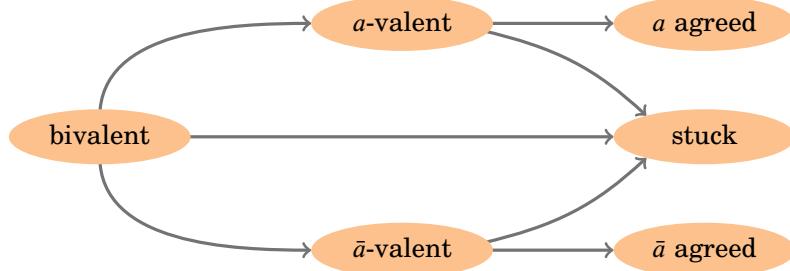
THEOREM 9. Let $\langle V, Q \rangle$ be an FBAS enjoying quorum intersection despite B , and suppose B contains all ill-behaved nodes. Let v_1 and v_2 be two nodes not in B . Let a and \bar{a} be contradictory statements. If v_1 confirms a , then v_2 cannot confirm \bar{a} .

PROOF. First note that *accept(a)* contradicts *accept(̄a)*—no well-behaved node can vote for both. Note further that v_1 must ratify *accept(a)* to confirm a . By Theorem 5, v_2 cannot ratify *accept(̄a)* and hence cannot confirm \bar{a} . \square

THEOREM 10. Let B be the set of befouled nodes in an FBAS $\langle V, Q \rangle$ with quorum intersection. Let U be a quorum containing an intact node ($U \not\subseteq B$), and let S be any set such that $U \subseteq S \subseteq V$. Let $S^+ = S \setminus B$ be the set of intact nodes in S , and let $S^- = (V \setminus S) \setminus B$ be the set of intact nodes not in S . Either $S^- = \emptyset$, or $\exists v \in S^-$ such that S^+ is v -blocking.

PROOF. If S^+ is v -blocking for some $v \in S^-$, then we are done. Otherwise, we must show $S^- = \emptyset$. If S^+ is not v -blocking for any $v \in S^-$, then, by Theorem 7, either $S^- = \emptyset$ or S^- is a quorum in $\langle V, Q \rangle^B$. In the former case we are done, while in the latter we get a contradiction: By Theorem 1, $U \setminus B$ is a quorum in $\langle V, Q \rangle^B$. Since B is a DSet (by Theorem 3), $\langle V, Q \rangle^B$ must enjoy quorum intersection, meaning $S^- \cap (U \setminus B) \neq \emptyset$. This is impossible, since $(U \setminus B) \subseteq S$ and $S^- \cap S = \emptyset$. \square

THEOREM 11. If an intact node in an FBAS $\langle V, Q \rangle$ with quorum intersection confirms a statement a , then, whatever subsequently transpires, once sufficient messages are delivered and processed, every intact node will accept and confirm a .

Fig. 11. Possible states of an accepted statement a at a single node v Fig. 12. Possible system-wide status of a statement a

PROOF. Let B be the DSet of befouled nodes and let $U \not\subseteq B$ be the quorum through which an intact node confirmed a . Let nodes in $U \setminus B$ broadcast $\text{accept}(a)$. By definition, any node v , regardless of how it has voted, accepts a after receiving $\text{accept}(a)$ from a v -blocking set. Hence, these messages may convince additional nodes to accept a . Let these additional nodes in turn broadcast $\text{accept}(a)$ until a point is reached at which, regardless of future communication, no further intact nodes can ever accept a . At this point let S be the set of nodes that claim to accept a (where $U \subseteq S$), let S^+ be the set of intact nodes in S , and let S^- be the set of intact nodes not in S . S^+ cannot be v -blocking for any node in S^- , or else more nodes could come to accept a . By Theorem 10, then, $S^- = \emptyset$, meaning every intact node has accepted a . \square

Figure 11 summarizes the paths an intact node v can take to confirm a . Given no knowledge, v might vote for either a or the contradictory \bar{a} . If v votes for \bar{a} , it cannot later vote for a , but can nonetheless accept a if a v -blocking set accepts it. A subsequent quorum of confirmation messages allows v to confirm a , which by Theorem 11 means the system agrees on a .

5.6. Liveness and neutralization

The main challenge of distributed consensus, whether centralized or not, is that a statement can get stuck in a permanently indeterminate state before the system reaches agreement on it. Hence, a protocol must not attempt to ratify externalized values directly. Should the statement “The value of slot i is x ” get stuck, the system will be forever unable to agree on slot i , losing liveness. The solution is to craft the statements in votes carefully. It must be possible to break a stuck statement’s hold on the question we really care about, namely slot contents. We call the process of obsoleting a stuck statement *neutralization*.

Local state	System-wide status of a
uncommitted	unknown (any)
voted a	unknown (any)
voted \bar{a}	unknown (any)
accepted a	stuck, a -valent, or a agreed
confirmed a	a agreed

Fig. 13. What an intact node knows about the status of statement a

More concretely, Figure 12 depicts the potential status a statement a can have system-wide. Initially, the system is *bivalent*, by which we mean there is one sequence of possible events through which all intact nodes will accept a , and another sequence through which all intact nodes will *reject* a (i.e., accept a statement \bar{a} contradicting a). At some point, one of these two outcomes may cease to be possible. If no intact node can ever reject a , we say the system is *a -valent*; conversely, if no intact node can ever accept a , we say the system is *\bar{a} -valent*.

At the time an FBAS transitions from bivalent to a -valent, there is a possible outcome in which all intact nodes accept a . However, this might not remain the case. Consider a PBFT-like four-node system $\{v_1, \dots, v_4\}$ in which any three nodes constitute a quorum. If v_1 and v_2 vote for a , the system becomes a -valent; no three nodes can ratify a contradictory statement. However, if v_3 and v_4 subsequently vote for \bar{a} contradicting a , it also becomes impossible to ratify a . In this case, a 's state is permanently indeterminate, or *stuck*.

As seen in Figure 10a, even once an intact node accepts a , the system may still fail to reach system-wide agreement on a . However, by Theorem 11, once an intact node confirms a , all intact nodes can eventually come to accept it; hence the system has agreed upon a . Figure 13 summarizes what intact nodes know about the global state of a statement from their own local state.

To preserve the possibility of consensus, a protocol must ensure that every statement is either irrefutable, and hence cannot get stuck, or neutralizable, and hence cannot block progress if stuck. There are two popular approaches to crafting neutralizable statements: the *view-based* approach, pioneered by viewstamped replication [Oki and Liskov 1988] and favored by PBFT [Castro and Liskov 1999]; and the *ballot-based* approach, invented by Paxos [Lamport 1998]. The ballot-based approach may be harder to understand [Ongaro and Ousterhout 2014]. Compounding confusion, people often call viewstamped replication “Paxos” or assert that the two algorithms are the same when they are not [van Renesse et al. 2014].

View-based protocols associate the *slots* in votes with monotonically increasing view numbers. Should consensus get stuck on the i th slot in view n , nodes recover by agreeing that view n had fewer than i meaningful slots and moving to a higher view number. Ballot-based protocols associate the *values* in votes with monotonically increasing ballot numbers. Should a ballot get stuck, nodes retry the same slot with a higher ballot, taking care never to select values that would contradict prior stuck ballots.

This work takes a ballot-based approach, as doing so makes it easier to do away with the notion of a distinguished primary node or leader. For example, leader behavior can be emulated [Lamport 2011b].

6. SCP: A FEDERATED BYZANTINE AGREEMENT PROTOCOL

This section presents the Stellar Consensus Protocol, SCP. At a high level, SCP consists of two sub-protocols: a nomination protocol and a ballot protocol. The nomination

protocol produces candidate values for a slot. If run long enough, it eventually produces the same set of candidate values at every intact node, which means nodes can combine the candidate values in a deterministic way to produce a single composite value for the slot. There are two huge caveats, however. First, nodes have no way of knowing when the nomination protocol has reached the point of convergence. Second, even after convergence, ill-behaved nodes may be able to reset the nomination process a finite number of times.

When nodes guess that the nomination protocol has converged, they execute the ballot protocol, which employs federated voting to commit and abort ballots associated with composite values. When intact nodes agree to commit a ballot, the value associated with the ballot will be externalized for the slot in question. When they agree to abort a ballot, the ballot's value becomes irrelevant. If a ballot gets stuck in a state where one or more intact nodes cannot commit or abort it, then nodes try again with a higher ballot; they associate the new ballot with the same value as the stuck one in case any node believes the stuck ballot was committed. Intuitively, safety results from ensuring that all stuck and committed ballots are associated with the same value. Liveness follows from the fact that a stuck ballot can be neutralized by moving to a higher ballot.

The remainder of this section presents the nomination and ballot protocols. Each is described first in terms of conceptual statements, then as a concrete protocol with messages representing sets of conceptual statements. Finally, Section 6.3 shows the correctness of the protocol. SCP treats each slot completely independently and can be viewed as many separate instances of a single-slot consensus protocol (akin to the “single-decree synod” in Paxos [Lamport 1998]). Concepts such as candidate values and ballots must always be interpreted in the context of a particular slot even if much of the discussion leaves the slot implicit.

6.1. Nomination protocol

Because slots need only be partially ordered, some applications of SCP will have only one plausible ballot per slot. For example, in certificate transparency, each CA may have its own series of slots and sign exactly one certificate tree per slot. However, other applications admit many plausible values per slot, in which case it is helpful to narrow down the possible input values. Our strategy is to begin with a synchronous nomination protocol that achieves consensus under certain timing assumptions, and feed the output of the nomination protocol into an asynchronous ballot protocol whose safety does not depend on timing [Lamport 2011a]. Such an initial synchronous phase is sometimes called a *conciliator* [Aspnes 2010].

The nomination protocol works by converging on a set of candidate values for a slot. Nodes then deterministically combine these candidates into a single *composite* value for the slot. Exactly how to combine values depends on the application. By way of example, the Stellar network uses SCP to choose a set of transactions and a ledger timestamp for each slot. To combine candidate values, Stellar takes the union of their transaction sets and the maximum of their timestamps. (Values with invalid timestamps will not receive enough nominations to become candidates.) Other possible approaches include combining sets by intersection or simply picking the candidate value with the highest hash.

Nodes produce a candidate value x through federated voting on the statement *nominate x*.

Definition (candidate). A node v considers a value x to be a *candidate* when v has confirmed the statement *nominate x*—i.e., v has ratified *accept(nominate x)*.

So long as node v has no candidate values, v may vote in favor of *nominate* x for any value x that passes application-level validity checks (such as timestamps not being in the future). In fact, v should generally re-nominate any values that it sees other nodes nominate, with some rate-limiting discussed below to avoid an explosion of candidates. As soon as v has a candidate value, however, it must cease voting to *nominate* x for any new values x . It should still continue to accept *nominate* statements for new values (when accepted by a v -blocking set) and confirm new *nominate* statements as prescribed by the federated voting procedure.

The nomination protocol enjoys several properties when a system has intact nodes (meaning it has avoided complete failure). Specifically, for each slot:

- (1) Intact nodes can produce at least one candidate value.
- (2) At some point, the set of possible candidate values stops growing.
- (3) If any intact node considers x to be a candidate value, then eventually every intact node will consider x to be a candidate value.

Now consider how the nomination protocol achieves its three properties. Property 1 is achieved because *nominate* statements are irrefutable. Nodes never vote against nominating a particular value, and until the first candidate value is confirmed, intact nodes can vote to nominate any value. So long as any value x passes application-level validity checks, intact nodes can vote for and confirm *nominate* x . Property 2 is ensured because once each intact node confirms at least one candidate value—which will happen in a finite amount of time—no intact nodes will vote to nominate any new values. Hence, the only values that can become candidates are those that already have votes from intact nodes. Property 3 is a direct consequence of Theorem 11.

The nomination process will be more efficient if fewer combinations of values are in play. Hence, we assign nodes a temporary priority and have each node, when possible, nominate the same values as a higher-priority node. More concretely, let H be a cryptographic hash function whose range can be interpreted as a set of integers $\{0, \dots, h_{\max} - 1\}$. (H might be SHA-256 [National Institute of Standards and Technology 2012], in which case $h_{\max} = 2^{256}$.) Let $G_i(m) = H(i, x_{i-1}, m)$ be a slot-specific hash function for slot i , where x_{i-1} is the value chosen for the slot preceding i (or the sorted set of values of all immediate dependencies of slot i when slots are governed by a partial order). Given a slot i and a *round number* n , each node v computes a set of *neighbors* and a *priority* for each neighbor as follows:

$$\begin{aligned} \text{weight}(v, v') &= \frac{\left| \{ q \mid q \in \mathbf{Q}(v) \wedge v' \in q \} \right|}{|\mathbf{Q}(v)|} \\ \text{neighbors}(v, n) &= \{ v' \mid G_i(\mathbf{N}, n, v') < h_{\max} \cdot \text{weight}(v, v') \} \\ \text{priority}(n, v') &= G_i(\mathbf{P}, n, v') \end{aligned}$$

\mathbf{N} and \mathbf{P} are constants to produce two different hash functions. The function $\text{weight}(v, v')$ returns the fraction of slices in $\mathbf{Q}(v)$ containing v' . By using weight as the probability over n that v' appears in $\text{neighbors}(v, n)$, we also reduce the chance that nodes without a lot of trust will dominate a round.

Each node v should initially find a node $v_0 \in \text{neighbors}(v, 0)$ that maximizes $\text{priority}(0, v_0)$ among nodes it can communicate with, then vote to *nominate* the same values as v_0 . Only if $v = v_0$ should v introduce a new value to nominate. v should use timeouts to decide on new *nominate* statements to vote for. After n timeouts, v should

Variable	Meaning
X	The set of values node v has voted to nominate
Y	The set of values node v has accepted as nominated
Z	The set of values that node v considers candidate values
N	The set of the latest NOMINATE message received from each node

Fig. 14. Nomination state maintained by node v for each slotNOMINATE $v \ i \ X \ Y \ D$

This is a message from node v nominating values for slot i . D is v 's quorum slice $\mathbf{Q}(v)$ or a collision-resistant hash of $\mathbf{Q}(v)$. X and Y are from v 's state. The concrete message encodes the following conceptual messages:

- { $\text{nominate } x \mid x \in X$ } (votes to nominate each value in X)
- { $\text{accept}(\text{nominate } x) \mid x \in Y$ } (votes to confirm nominations in Y)

Fig. 15. Message in nomination protocol

find a node $v_n \in \text{neighbors}(v, n)$ maximizing $\text{priority}(n, v_n)$ and vote to nominate everything v_n has voted to nominate.

THEOREM 12. *Eventually, all intact nodes will have the same composite value.*

PROOF. The theorem follows from the three properties of the nomination protocol. Each intact node will only ever vote to nominate a finite number of ballots. In the absence of action by ill-behaved nodes, intact nodes will converge on the same set of candidate values, call it Z . To forestall this convergence, ill-behaved nodes may introduce new candidate values, which for a period may be candidates at some but not all intact nodes. Such values will need to have garnered votes from well-behaved nodes, however, which limits them to a finite set. Eventually, ill-behaved nodes will either stop perturbing the system or run out of new candidate values to inject, in which case intact nodes will converge on Z . \square

6.1.1. Concrete nomination protocol. Figure 14 lists the nomination protocol state a node v must maintain for each slot. X is the set of values x for which v has voted $\text{nominate } x$, Y is the set of values for which v has accepted $\text{nominate } x$, and Z is the set of candidate values—i.e., all values for which a quorum including v has stated $\text{accept}(\text{nominate } x)$. Finally, v maintains N , the latest concrete message from each node. (Technically, X , Y , and Z can all be recomputed from N , but it is convenient to be able to reference them directly.) All four fields are initialized to the empty set. Note that all three of X , Y , and Z are growing over time—nodes never remove a value from these sets.

Figure 15 shows the concrete message that constitutes the nomination protocol. Because X and Y grow monotonically over time, it is possible to determine which of multiple NOMINATE messages from the same node is the latest, independent of network delivery order, so long as D does not change mid-nomination (or D has to be versioned). Only one remote procedure call (RPC) is needed for nomination—the argument is the sender's latest NOMINATE message and the return value is the receiver's. If D or the nominated values are cryptographic hashes, a second RPC should permit retrieval of uncached hash preimages as needed.

Because nodes cannot tell when the nomination protocol is complete anyway, SCP must cope with different composite values at different nodes. As an optimization, then,

nodes can attempt to predict the final composite value before they even have a candidate value. To do this, the composite value can be taken as $\text{combine}(Z)$ when $Z \neq \emptyset$, otherwise $\text{combine}(Y)$ when $Y \neq \emptyset$, otherwise $\text{combine}(X)$ when $X \neq \emptyset$. This means the highest-priority node can optimistically initiate balloting at the same time as nomination, piggybacking its first ballot message PREPARE (described below) on its first NOMINATE message.

6.2. Ballot protocol

Once nodes have a composite value, they engage in the ballot protocol, though nomination may continue to update the composite value in parallel. A ballot b is a pair of the form $b = \langle n, x \rangle$, where $x \neq \perp$ is a value and b is a referendum on externalizing x for the slot in question. The value $n \geq 1$ is a counter to ensure higher ballot numbers are always available. We use C-like notation $b.n$ and $b.x$ to denote the counter and value fields of ballot b , so that $b = \langle b.n, b.x \rangle$. Ballots are totally ordered, with $b.n$ more significant than $b.x$. For convenience, a special invalid null ballot $\mathbf{0} = \langle 0, \perp \rangle$ is less than all other ballots, and a special counter value ∞ is greater than all other counters.

We speak of committing and aborting a ballot b as a shorthand for using federated voting to agree on the statements *commit b* and *abort b*, respectively. For a given ballot, *commit* and *abort* are contradictory, so a well-behaved node may vote for at most one of them. In the notation of Section 5, the opposite of *commit b* would be “ $\overline{\text{commit } b}$,” but *abort b* is a more intuitive notation.

Because at most one value can be chosen for a given slot, all committed and stuck ballots must contain the same value. Roughly speaking, this means *commit* statements are invalid if they conflict with lower-numbered unaborted ballots.

Definition (compatible). Two ballots b_1 and b_2 are *compatible*, written $b_1 \sim b_2$, iff $b_1.x = b_2.x$ and *incompatible*, written $b_1 \not\sim b_2$, iff $b_1.x \neq b_2.x$. We also write $b_1 \lesssim b_2$ or $b_2 \gtrsim b_1$ iff $b_1 \leq b_2$ (or equivalently $b_2 \geq b_1$) and $b_1 \sim b_2$. Similarly, $b_1 \not\lesssim b_2$ or $b_2 \not\gtrsim b_1$ means $b_1 \leq b_2$ (or equivalently $b_2 \geq b_1$) and $b_1 \not\sim b_2$.

Definition (prepared). A ballot b is *prepared* iff every statement in the following set is true: $\{ \text{abort } b_{\text{old}} \mid b_{\text{old}} \not\lesssim b \}$.

More precisely, then, *commit b* is valid to vote for only if b is confirmed prepared, which nodes ensure through federated voting on the corresponding *abort* statements. It is convenient to vote on these statements en masse, so wherever we write “ b is prepared,” the surrounding context applies to the whole set of *abort* statements. In particular, a node votes, accepts, or confirms that b is prepared iff it votes for, accepts, or confirms, respectively, all of these *aborts*.

To commit a ballot b and externalize its value $b.x$, SCP nodes first accept and confirm b is prepared, then accept and confirm *commit b*. Before the first intact node votes for *commit b*, the prepare step, through federated voting, ensures all intact nodes can eventually confirm b is prepared. When an intact node v accepts *commit b*, it means $b.x$ will eventually be chosen. However, as discussed in Section 5.4.1, v must confirm *commit* before acting on it in case v is befouled.

6.2.1. Concrete ballot protocol. Figure 16 illustrates the per-slot state maintained by each node. A node v stores: its current phase φ ; its current ballot b ; the two most recent incompatible ballots it has prepared (p, p'); the lowest (c) and highest (h) ballot, if any, it has voted to *commit* and for which it has not subsequently accepted an *abort* (or for which it has accepted or confirmed a *commit* in later phases); a next value z to try if the current ballot fails; and the latest message received from each node (M). Ballots b , p , p' , and h are non-decreasing within a phase. In addition, if $c \neq \mathbf{0}$ —meaning v may

Variable	Meaning
φ	Current phase: one of PREPARE, CONFIRM, or EXTERNALIZE
b	Current ballot that node v is attempting to prepare and commit ($b \neq \mathbf{0}$)
p', p	The two highest ballots accepted as prepared such that $p' \not\lesssim p$, where $p' = \mathbf{0}$ or $p = p' = \mathbf{0}$ if there are no such ballots
c, h	In PREPARE: h is the highest ballot confirmed as prepared, or $\mathbf{0}$ if none; if $c \neq \mathbf{0}$, then c is lowest and h the highest ballot for which v has voted <i>commit</i> and not accepted <i>abort</i> . In CONFIRM: lowest, highest ballot for which v accepted <i>commit</i> In EXTERNALIZE: lowest, highest ballot for which v confirmed <i>commit</i> Invariant: if $c \neq \mathbf{0}$, then $c \lesssim h \lesssim b$.
z	Value to use in next ballot. If $h = \mathbf{0}$, then z is the composite value (see Section 6.1); otherwise, $z = h.x$.
M	Set of the latest ballot message seen from each node

Fig. 16. Ballot state maintained by each node v for each slotPREPARE $v i b p p' c.n h.n D$

This is a message from node v about slot i . D specifies $\mathbf{Q}(v)$. The other fields reflect v 's state. Values $c.x$ and $h.x$ are elided as $c.x = h.x = b.x$ when $c.n \neq 0$. This concrete message encodes a host of conceptual statements, as follows:

- $\{ \text{abort } b' \vee \text{accept}(\text{abort } b') \mid b' \not\lesssim b \}$ (a vote to prepare b)
- $\{ \text{accept}(\text{abort } b') \mid b' \not\lesssim p \}$ (a vote to confirm p is prepared)
- $\{ \text{accept}(\text{abort } b') \mid b' \not\lesssim p' \}$ (a vote to confirm p' is prepared)
- $\{ \text{commit } b' \mid c.n \neq 0 \wedge c \lesssim b' \lesssim h \}$ (a vote to commit c, \dots, h if $c \neq \mathbf{0}$)

CONFIRM $v i b p.n c.n h.n D$

Sent by v to try to externalize $b.x$ for slot i after accepting a *commit*. Implies $p.x = c.x = h.x = b.x$ in v 's state. For convenience, we also say $p' = \mathbf{0}$ (p' is irrelevant after accepting *commit*). D specifies $\mathbf{Q}(v)$ as above. Encodes:

- Everything implied by PREPARE $v i \langle \infty, b.x \rangle p \mathbf{0} c.n \infty D$
- $\{ \text{accept}(\text{commit } b') \mid c \lesssim b' \lesssim h \}$ (a vote to confirm *commit* c, \dots, h)

EXTERNALIZE $v i x c.n h.n D$

After v confirms *commit* $\langle c.n, x \rangle$ for slot i and externalizes value x , this message helps other nodes externalize x . Implies $c = \langle c.n, x \rangle$ and $h = \langle h.n, x \rangle$. For convenience, we also say $b = p = h = \langle \infty, x \rangle$, and $p' = \mathbf{0}$. Encodes:

- Everything implied by CONFIRM $v i \langle \infty, x \rangle \infty c.n \infty D$
- Everything implied by CONFIRM $v i \langle \infty, x \rangle \infty c.n h.n \{\{v\}\}$

Fig. 17. Messages in SCP's ballot protocol

have participated in ratifying *commit* c —code must ensure $c \lesssim h \lesssim b$. This invariant guarantees a node can always legally vote to prepare its current ballot b .

Figure 17 shows the three ballot protocol messages, with φ determining which one of the three a node can send. Ballot messages may overlap with nomination messages, so that, when $h = \mathbf{0}$, a node may update z in response to a NOMINATE message. Note

that “ $a \vee \text{accept}(a)$ ” is what each node must assert for a quorum to accept a under condition 1 of the definition of accept .

For convenience, when comparing state across nodes, we will identify fields belonging to particular nodes with subscripts. If v is a node, then we write b_v, p_v, p'_v, \dots to denote the values of b, p, p', \dots in node v 's state as described in Figure 16. Similarly, we let v_m denote message m 's sender, and b_m, p_m, p'_m, \dots denote the corresponding values of b, p, p', \dots in v_m 's state as implied by m .

Each node initializes its ballot state for a slot by setting $\varphi \leftarrow \text{PREPARE}$, $z \leftarrow \perp$, $b \leftarrow \langle 0, z \rangle$, $M \leftarrow \emptyset$, and all other fields (p, p', c, h) to the invalid ballot $\mathbf{0}$. While $z = \perp$, a node can receive but not send ballot messages. Once $z \neq \perp$, if $b.n = 0$, a node reinitializes $b \leftarrow \langle 1, z \rangle$ to start sending messages. Nodes then repeatedly exchange messages with peers, sending whichever ballot message is indicated by φ . Upon adding a newly received message m to M_v , a node v updates its state as follows:

- (1) If $\varphi = \text{PREPARE}$ and m lets v accept new ballots as prepared, update p and p' . Afterwards, if either $p \not\geq h$ or $p' \not\geq h$, then set $c \leftarrow \mathbf{0}$.
- (2) If $\varphi = \text{PREPARE}$ and m lets v confirm new higher ballots prepared, then raise h to the highest such ballot and set $z \leftarrow h.x$.
- (3) If $\varphi = \text{PREPARE}$, $c = \mathbf{0}$, $b \leq h$, and neither $p \not\geq h$ nor $p' \not\geq h$, then set c to the lowest ballot satisfying $b \leq c \leq h$.
- (4) If $\varphi = \text{PREPARE}$ and v accepts commit for one or more ballots, set c to the lowest such ballot, then set h to the highest ballot such that v accepts all $\{ \text{commit } b' \mid c \leq b' \leq h \}$, and set $\varphi \leftarrow \text{CONFIRM}$. Also set $z \leftarrow h.x$ after updating h , and unless $h \leq b$, set $b \leftarrow h$.
- (5) If $\varphi = \text{CONFIRM}$ and the received message lets v accept new ballots prepared, raise p to the highest accepted prepared ballot such that $p \sim c$.
- (6) If $\varphi = \text{CONFIRM}$ and v accepts more commit messages or raises b , then let h' be the highest ballot such that v accepts all $\{ \text{commit } b' \mid b \leq b' \leq h' \}$ (if any). If there exists such an h' and $h' > h$, then set $h \leftarrow h'$, and, if necessary, raise c to the lowest ballot such that v accepts all $\{ \text{commit } b' \mid c \leq b' \leq h \}$.
- (7) If $\varphi = \text{CONFIRM}$ and v confirms $\text{commit } c'$ for any c' , set c and h to the lowest and highest such ballots, set $\varphi \leftarrow \text{EXTERNALIZE}$, externalize $c.x$, and terminate.
- (8) If $\varphi \in \{\text{PREPARE}, \text{CONFIRM}\}$ and $b < h$, then set $b \leftarrow h$.
- (9) If $\varphi \in \{\text{PREPARE}, \text{CONFIRM}\}$ and $\exists S \subseteq M_v$ such that the set of senders $\{ v_{m'} \mid m' \in S \}$ is v -blocking and $\forall m' \in S, b_{m'}.n > b_v.n$, then set $b \leftarrow \langle n, z \rangle$, where n is the lowest counter for which no such S exists. Repeat the previous steps after updating b .

While $c = \mathbf{0}$, the above protocol implements federated voting to confirm b is prepared. Once $c \neq \mathbf{0}$, the protocol implements federated voting on $\text{commit } c'$ for every $c \leq c' \leq h$. For the CONFIRM phase, once a well-behaved node accepts $\text{commit } c$, the node never accepts, and hence never attempts to confirm, $\text{commit } c'$ for any $c' \neq c$. Once a commit is confirmed, the value of its ballot is safe to externalize assuming quorum intersection.

All messages sent by a particular node are totally ordered by $\langle \varphi, b, p, p', h \rangle$, with φ the most significant and h the least significant field. The values of these fields can be determined from messages, as described in Figure 17. All PREPARE messages precede all CONFIRM messages, which in turn precede the single EXTERNALIZE message for a given slot. The ordering makes it possible to ensure M contains only the latest ballot

from each node without relying on timing to order the messages, since the network may re-order messages.

A few details of the protocol merit explanation. The statements implied by PREPARE of the form “*abort b' v accept(abort b')*” do not specify whether v is voting for or confirming *abort b'*. The distinction is unimportant for the definition of *accept*. Glossing over the distinction allows v to forget about old ballots it voted to *commit* (and hence cannot vote to *abort*), so long as it accepted an *abort* message for them. Indeed, the only time v modifies c when $c \neq \mathbf{0}$ is to set it back to $\mathbf{0}$ after accepting *abort* for every ballot it is voting to *commit* in step 1 on the preceding page. Conversely, the only time v modifies c when $c = \mathbf{0}$ is to set it to a value $c \geq b$ in step 3. Because nodes never vote *abort c* for any $c \geq b$, no past *abort* votes can conflict with *commit c*.

Theorem 11 requires that nodes rebroadcast what they have accepted. It follows from the definition of *prepare* that the two highest incompatible ballots a node has accepted as prepared subsume all ballots the node has accepted as prepared. Hence, including p and p' in every message ensures that nodes converge on h —a confirmed prepared ballot. Note further that the ballots a node *accepts* as prepared must be a superset of the ballots the node *confirms* as prepared; hence, step 2 can never set h such that $h \sim c \neq \mathbf{0}$, as step 1 will set $c \leftarrow \mathbf{0}$ if the new h is incompatible with the old c .

At the time v sends an EXTERNALIZE message, it has accepted $\{\text{commit } b' \mid b' \gtrsim c\}$. More importantly, however, it has confirmed $\{\text{commit } b' \mid c \lesssim b' \lesssim h\}$. v can assert its acceptance of confirmed statements without regard to $Q(v)$, because it has already checked that one of its slices unanimously agrees; this explains the appearance of $\{\{v\}\}$ in place of D for the second implicit CONFIRM message in the description of EXTERNALIZE. Eliminating D allows a single static EXTERNALIZE message to help other nodes catch up arbitrarily far in the future, even if quorum slices have changed significantly in the meantime.

Only one RPC is needed to exchange ballot messages. The argument is the sender's latest message and the return value is the receiver's latest message. As with NOMINATE, if D or the values x in ballots are cryptographic hashes, then a separate RPC is needed to retrieve uncached hash preimages.

6.2.2. Timeouts and ballot updates. If all intact nodes start with the same ballot b , then steps 1 to 9 on the previous page are sufficient to confirm *commit b* and externalize value $b.x$. Unfortunately, if the ballot protocol starts before the nomination protocol has converged, nodes may start off with different values for z . If a ballot fails, or takes long enough that it may fail because of unresponsive nodes, then nodes must time out and try again with a higher ballot. For this reason, nodes employ a timer as follows:

- (a) A node v with $\varphi_v \neq \text{EXTERNALIZE}$ arms a timer whenever $\exists S \subseteq M_v$ such that the set of senders $U = \{v_m \mid m \in S\}$ is a quorum, $v \in U$, and $\forall m \in S, b_m.n \geq b_v.n$.
- (b) If the timer fires, v updates its ballot by setting $b_v \leftarrow \langle b_v.n + 1, z_v \rangle$.

Different nodes may start ballots at different times. However, condition (a) delays setting a timer at a node v that has gotten ahead of a quorum. Conversely, step 9 on the preceding page allows nodes that have fallen too far behind to catch up without waiting for timers. Taken together, these rules ensure that given long enough timers, intact nodes will spend time together on the same ballot; moreover, this time will grow proportionally to the timer duration. To ensure timeouts are long enough without predicting latencies, an implementation can increase the timeout as a function of $b.n$.

6.3. Correctness

An SCP node cannot vote to confirm *commit b* until it has voted to confirm *abort* for all lower-numbered incompatible ballots. Because a well-behaved node cannot accept (and

hence vote to confirm) contradictory statements, this means that for a given $\langle V, Q \rangle$, Theorem 5 ensures a set S of well-behaved nodes cannot externalize contradictory values so long as S enjoys quorum intersection despite $V \setminus S$. This safety holds if V and Q change only between slots, but what if they change mid-slot (for instance, in reaction to node crashes)?

To reason about safety under reconfiguration, we join all old and new quorum slice sets, reflecting the fact that nodes may make decisions based on a combination of messages from different configuration eras. To be very conservative, we might require quorum intersection of the aggregation of the present configuration with every past configuration. However, we can relax this slightly by separating nodes that have sent illegal messages from those that have merely crashed.

THEOREM 13. *Let $\langle V_1, Q_1 \rangle, \dots, \langle V_k, Q_k \rangle$ be the set of configurations an FBAS has experienced during agreement on a single slot. Let $V = V_1 \cup \dots \cup V_k$ and $Q(v) = \{ q \mid \exists j \text{ such that } v \in V_j \wedge q \in Q_j(v) \}$. Let $B \subseteq V$ be a set such that B contains all ill-behaved nodes that have sent illegal messages, though $V \setminus B$ may still contain crashed (unresponsive) nodes. Suppose nodes v_1 and v_2 are well-behaved, v_1 externalizes x_1 for the slot, and v_2 externalizes x_2 . If $\langle V, Q \rangle^B$ enjoys quorum intersection, then $x_1 = x_2$.*

PROOF. For v_1 to externalize x_1 , it must have ratified $\text{accept}(\text{commit } \langle n_1, x_1 \rangle)$ in collaboration with a pseudo-quorum $U_1 \subseteq V$. We say pseudo-quorum because U_1 might not be a quorum in $\langle V_j, Q_j \rangle$ for any particular j , as ratification may have involved messages spanning multiple configurations. Nonetheless, for ratification to succeed $\forall v \in U_1, \exists j, \exists q \in Q_j(v)$ such that $q \subseteq U_1$. It follows from the construction of Q that $q \in Q(v)$. Hence U_1 is a quorum in $\langle V, Q \rangle$. By a similar argument a pseudo-quorum U_2 must have ratified $\text{accept}(\text{commit } \langle n_2, x_2 \rangle)$, and U_2 must be a quorum in $\langle V, Q \rangle$. By quorum intersection of $\langle V, Q \rangle^B$, there must exist some $v \in V \setminus B$ such that $v \in U_1 \cap U_2$. By assumption, such a $v \notin B$ could not claim to accept incompatible ballots. Since v confirmed accepting commit for ballots with both x_1 and x_2 , it must be that $x_1 = x_2$. \square

For liveness of a node v , we care about several things when an FBAS has undergone a series of reconfigurations $\langle V_1, Q_1 \rangle, \dots, \langle V_k, Q_k \rangle$ within a single slot. First, the safety prerequisites of Theorem 13 must hold for v and the set of nodes v cares about, since violating safety undermines liveness and Theorem 11 requires quorum intersection. Second, the set of ill-behaved nodes in the latest state, $\langle V_k, Q_k \rangle$, must not be v -blocking, as this could deny v a quorum and prevent it from ratifying statements. Finally, v 's state must never have been poisoned by a v -blocking set falsely claiming to accept a statement.

To summarize, then, if B is the set of nodes that have sent illegal messages, we consider a node v to be *cumulatively intact* when the following conditions hold:

- (1) v is intact in the latest configuration $\langle V_k, Q_k \rangle$,
- (2) The aggregation of the present and all past configurations has quorum intersection despite B (i.e., the prerequisite for Theorem 13 holds), and
- (3) B is not v -blocking in $\langle V_j, Q_j \rangle$ for any $1 \leq j \leq k$.

The next few theorems show that ill-behaved nodes cannot drive intact nodes into dead-end stuck states:

THEOREM 14. *In an FBAS with quorum intersection, if no intact node is in the EXTERNALIZE phase and an intact node with ballot $\langle n, x \rangle$ arms its timer as described in Section 6.2.2, then, given sufficient communication, every intact node v can set $b_v \geq n$ before any timer fires.*

PROOF. Let $S = \{v \mid b_v \geq n\}$ be the set of nodes with counters at least n . By assumption, S contains an intact node. Furthermore, because that intact node armed its timer, S must also encompass a quorum. Let S^+ be the intact subset of S , and S^- be the set of intact nodes not in S . By Theorem 10, either $S^- = \emptyset$ (in which case the theorem is trivial), or S^+ is v -blocking for some $v \in S$. By step 9 on page 24, v will adjust its ballot so $b_v.n \geq n$. At this point, repeat the argument with $S \leftarrow S \cup \{v\}$ until such point as $S^- = \emptyset$. \square

THEOREM 15. *Given long enough timeouts, if an intact node has reached the CONFIRM phase with $b.x = x$, then eventually all intact nodes will terminate.*

PROOF. If an intact node has reached the EXTERNALIZE phase, it has confirmed *commit* c for some ballot c . By Theorem 11, all intact nodes will confirm *commit* c , after which they will terminate in step 7 on page 24.

Otherwise, an intact node in the CONFIRM phase has accepted *commit* c where $c = \langle n, x \rangle$. Beforehand, an intact node confirmed c was prepared. By Theorem 11, all intact nodes will eventually have $h \geq c$. Moreover, by Theorem 8, no intact node v can accept *abort* c , so no intact node can accept as prepared any ballot p such that $p \not\geq c$. Hence, after sufficient communication, every intact node will permanently have $h \geq c$. The intact node or nodes with the lowest b will, by Theorem 14, raise their ballots until such point as all intact nodes with armed timers have the same ballot counter. Since they also have identical $z = h.x = x$, they will all have the same ballot. If they cannot complete the protocol because one or more intact nodes have higher ballots, the nodes with higher numbered ballots will not have timers set. Hence, the nodes with lower-numbered ballots will after a timeout set set $b \leftarrow \langle b.n + 1, x \rangle$ until eventually all intact nodes are on the same ballot and can complete the protocol. \square

THEOREM 16. *Regardless of past ill-behavior, given long enough timeouts and periods in which ill-behaved nodes do not send new messages, intact nodes running SCP will terminate.*

PROOF. By Theorem 12, all intact nodes will eventually have identical sets Z of candidate values. Assume this point has passed and every intact node v has the same composite value $z = \text{combine}(Z)$. If no intact node ever confirms any ballot b prepared without $b.x = z$, then after at most one timeout, all new ballots of intact nodes will have value z and, given a sufficient timeout, complete the protocol. By Theorem 15, nodes will also complete if any intact node has progressed beyond the PREPARE phase.

The remaining case is that an intact node has $h \neq \mathbf{0}$ and all intact nodes have $\varphi = \text{PREPARE}$. By Theorem 14, when the intact node or nodes with the highest $b.n$ arm their timers, if timers are long enough, other nodes will catch up. Moreover, by Theorem 11, if timers are long enough, nodes will converge on the value of h (the highest confirmed prepared ballot) before the next timeout, at which point all intact nodes will raise b to the same value and complete the protocol. \square

Theorem 16 assures us there are no dead-end states in SCP. However, a set of ill-behaved nodes with very good timing could perpetually preempt an SCP system by delaying messages so that some fraction of intact nodes update h right before timers fire and the remaining update it after, preventing intact nodes from converging on the next ballot. Nodes can recover from such an attack by removing ill-behaved nodes from their slices.

An alternative would be to add randomness to the protocol, for instance changing step 2 on page 24 to update z with probability $1/2$ (or even with probability proportional to the fraction of the timer remaining). Such an approach would terminate with

probability 1, but in worse expected running time for the common case that most or all nodes are well-behaved or fail-stop.

7. LIMITATIONS

SCP can only guarantee safety when nodes choose adequate quorum slices. Section 3.2 discusses why we can reasonably expect them to do so. Nonetheless, when security depends upon a user-configurable parameter, there is always the possibility people will set it wrong.

Even when people set quorum slices correctly and SCP guarantees safety, safety alone does not rule out other security issues that may arise in a federated system. For example, in a financial market, widely trusted nodes could leverage their position in the network to gain information with which to engage in front running or other unethical activities.

Byzantine nodes may attempt to filter transactions on the input side of SCP while otherwise producing the correct output. If well-behaved nodes accept all transactions, the *combine* function takes the union of transactions, and there are intact nodes, then such filtering will eventually fail to block victim transactions with probability 1, but may nonetheless impose delays.

Though SCP's safety is optimal, its performance and communication latency are not. In the common case that nodes have not previously voted to commit ballots incompatible with the current one, it is possible to reduce the number of communication rounds by one. An earlier version of SCP did so, but the protocol description was more complex. First, it required nodes to cache and retransmit signed messages previously sent by failed nodes. Second, it was no longer possible to gloss over the distinction between votes and confirmations of *abort* statements in PREPARE messages, so nodes had to send around potentially unbounded lists of exceptions to their *abort* votes.

SCP can suffer perpetual preemption as discussed in Section 6.3. An open question is whether, without randomness, a different protocol could guarantee termination assuming bounded communication latency but tolerating Byzantine nodes that continuously inject bad messages at exactly the point where timeouts fire. Such a protocol is not ruled out by the FLP impossibility result [Fischer et al. 1985]. However, the two main techniques to guarantee termination assuming synchrony do not directly apply in the FBA model: PBFT [Castro and Liskov 1999] chooses a leader in round-robin fashion, which is not directly applicable when nodes do not agree on membership. (Possibly something along the lines of priority in Section 6.1 could be adapted.) The Byzantine Generals protocol [Lamport et al. 1982] relays messages so as to compensate for ill-behaved nodes saying different things to different honest nodes, an approach that cannot help when nodes depend on distinct ill-behaved nodes in their slices. Still another possibility might be to leverage both randomness and synchrony to terminate with probability 1, but in shorter expected time than Ben Or-style randomized protocols [Ben-Or 1983] that make no synchrony assumptions. Public coin techniques [?] that speed up randomized centralized Byzantine agreement protocols appear to be difficult to adapt to the federated model, barring some cryptographic breakthrough in federated threshold signatures.

Unfortunately, changing slices mid-slot to accommodate failed nodes is problematic for liveness if a well-behaved node v has ever experienced a wholly malicious and colluding v -blocking set. The good news is that Theorem 13 guarantees safety to any set S of well-behaved nodes enjoying quorum intersection despite $V \setminus S$, even when S has befouled members. The bad news is that updating \mathbf{Q} may be insufficient to unblock nodes if well-behaved nodes were tricked into voting to confirm a bad *commit* message. In such a situation, nodes must disavow past votes, which they can do only by rejoining the system under a new node names. There may exist a way to automate such

recovery, such as having other nodes recognize reincarnated nodes and automatically update their slices.

The FBA model requires continuity of participants over time. Should all nodes simultaneously and permanently leave, restarting consensus would require central coordination or human-level agreement. By contrast, a proof-of-work system such as Bitcoin could undergo sudden complete turnover yet continue to operate with little human intervention. On the other hand, if nodes do return, an FBAS can recover from an arbitrarily long outage, while a proof-of-work scheme would face the possibility of an attacker working on a fork during the outage.

An intriguing possibility is to leverage SCP to mediate tussles [Clark et al. 2005] by voting on changes to configuration parameters or upgrades to an application protocol. One way to do this is to nominate special messages that update parameters. Candidate values could then consist of both a set of values and a set of parameter updates. A big limitation of this approach is that a set of malicious nodes large enough to deny the system a quorum but not large enough to undermine safety could nonetheless trigger configuration changes by lying and putting configuration changes in Y that were never ratified. It remains an open question how to vote on parameter changes in a way that requires the consent of a full quorum but also never jeopardizes liveness.

8. SUMMARY

Byzantine agreement has long enabled distributed systems to achieve consensus with efficiency, standard cryptographic security, and flexibility in designating trusted participants. More recently, Bitcoin introduced the revolutionary notion of decentralized consensus, leading to many new systems and research challenges. This paper introduces federated Byzantine agreement (FBA), a model for achieving decentralized consensus while preserving the traditional benefits of Byzantine agreement. The key distinction between FBA and prior Byzantine agreement systems is that FBA forms quorums from participants' individual trust decisions, allowing an organic growth model similar to that of the Internet. The Stellar Consensus Protocol (SCP) is a construction for FBA that achieves optimal safety against ill-behaved participants.

Acknowledgments

Jed McCaleb inspired this work and provided feedback, terminology suggestions, and help thinking through numerous conjectures. Jessica Collier collaborated on writing the paper. Stan Polu created the first implementation of SCP and provided invaluable corrections, suggestions, simplifications, and feedback in the process. Jelle van den Hooff provided the key idea to restructure the paper around quorum intersection and federated voting, as well as other crucial suggestions for terminology, organization, and presentation. Nicolas Barry found several bugs in the paper as he implemented the protocol, as well as identifying necessary clarifications. Ken Birman, Bekki Bolt-house, Joseph Bonneau, Mike Hamburg, Graydon Hoare, Joyce Kim, Tim Makarios, Mark Moir, Robert Morris, Lucas Ryan, and Katherine Tom slogged through drafts of the paper, identifying errors and sources of confusion as well as providing helpful suggestions. Eva Gantz provided helpful motivation and references. Winnie Lim provided guidance on figures. The reddit community and Tahoe-LAFS group pointed out a censorship weakness in an earlier version of SCP, leading to the improved nomination protocol. Finally, the author would like to thank the whole Stellar team for their support, feedback, and encouragement.

Disclaimer

Professor Mazières's contribution to this publication was as a paid consultant, and was not part of his Stanford University duties or responsibilities.

REFERENCES

- Eduardo A. Alchieri, Alysson Neves Bessani, Joni Silva Fraga, and Fabíola Greve. 2008. Byzantine Consensus with Unknown Participants. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*. 22–40.
- James Aspnes. 2010. A Modular Approach to Shared-memory Consensus, with Applications to the Probabilistic-write Model. In *Proceedings of the 29th Symposium on Principles of Distributed Computing*. 460–467.
- Rachel Banning-Lover. 2015. Boatfuls of cash: how do you get money into fragile states? (February 2015). <http://www.theguardian.com/global-development-professionals-network/2015/feb/19/boatfuls-of-cash-how-do-you-get-money-into-fragile-states>.
- David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Paweł Szalachowski. 2014. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 382–393.
- Michael Ben-Or. 1983. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *Proceedings of the 2nd Symposium on Principles of Distributed Computing*. 27–30.
- Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*.
- Gabriel Bracha and Sam Toueg. 1985. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM* 32, 4 (Oct. 1985), 824–840.
- Danny Bradbury. 2013. Feathercoin hit by massive attack. (June 2013). <http://www.coindesk.com/feathercoin-hit-by-massive-attack/>.
- Vitalik Buterin. 2014. Slasher: A Punitive Proof-of-Stake Algorithm. (January 2014). <https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/>.
- Miguel Castro and Barbara Liskov. 1999. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*. 173–186.
- CGAP. 2008. Making Money Transfers Work for Microfinance Institutions. (March 2008). <http://www.cgap.org/sites/default/files/CGAP-Technical-Guide-Making-Money-Transfers-Work-for-Microfinance-Institutions-A-Technical-Guide-to-Developing-and-Delivering-Money-Transfers-Mar-2008.pdf>.
- David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. 2005. Tussle in Cyberspace: Defining Tomorrow's Internet. *IEEE/ACM Transactions on Networking* 13, 3 (June 2005), 462–475.
- crazyearner. 2013. TERRACOIN ATTACK OVER 1.2TH ATTACK CONFIRMD [sic]. (July 2013). <https://bitcointalk.org/index.php?topic=261986.0>.
- Kourosh Davarpanah, Dan Kaufman, and Ophelie Pubellier. 2015. NeuCoin: the First Secure, Cost-efficient and Decentralized Cryptocurrency. (March 2015). <http://www.neucoin.org/en/whitepaper/download>.
- Asli Demirguc-Kunt, Leora Klapper, Dorothe Singer, and Peter Van Oudheusden. 2015. *The Global Findex Database 2014 Measuring Financial Inclusion Around the World*. Policy Research Working Paper 7255. World Bank. http://www-wds.worldbank.org/external/default/WDSContentServer/WDSP/IB/2015/04/15/090224b082dca3aa/1_0/Rendered/PDF/TheGlobalFinInclusionaroundtheWorld.pdf.
- John R. Douceur. 2002. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. 251–260.
- Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. 1988. Consensus in the Presence of Partial Synchrony. *Journal of the ACM* 35, 2 (April 1988), 288–323.
- Cynthia Dwork and Moni Naor. 1992. Pricing via Processing or Combatting Junk Mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. 139–147.
- Ittay Eyal and Emin Gün Sirer. 2013. Majority is not Enough: Bitcoin Mining is Vulnerable. (November 2013). <http://arxiv.org/abs/1311.0243>.
- Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM* 32, 2 (April 1985), 374–382.
- Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 906–917.
- Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. 2013. Accountable Key Infrastructure (AKI): A Proposal for a Public-key Validation Infrastructure. In *Proceedings of the 22nd International Conference on World Wide Web*. 679–690.

- Sunny King and Scott Nadal. 2012. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. (August 2012). <http://peercoin.net/assets/paper/peercoin-paper.pdf>.
- Jae Kwon. 2014. Tendermint: Consensus without Mining. (2014). <http://tendermint.com/docs/tendermint.pdf>.
- Leslie Lamport. 1998. The Part-Time Parliament. 16, 2 (May 1998), 133–169.
- Leslie Lamport. 2011a. Brief Announcement: Leaderless Byzantine Paxos. In *Proceedings of the 25th International Conference on Distributed Computing*. 141–142.
- Leslie Lamport. 2011b. Byzantizing Paxos by Refinement. In *Proceedings of the 25th International Conference on Distributed Computing*. 211–224.
- Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (July 1982), 382–401.
- Adam Langley. 2015. Maintaining digital certificate security. (March 2015). <http://googleonlinesecurity.blogspot.com/2015/03/maintaining-digital-certificate-security.html>.
- Ben Laurie, Adam Langley, and Emilia Kasper. 2013. *Certificate Transparency*. RFC 6962. Internet Engineering Task Force (IETF). <http://tools.ietf.org/html/rfc6962>.
- Jinyuan Li and David Mazières. 2007. Beyond One-third Faulty Replicas in Byzantine Fault Tolerant Systems. In *Proceedings of the 4th Symposium on Networked Systems Design and Implementation*. 131–144.
- Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Michael J. Freedman, and Edward W. Felten. 2014. CONIKS: A Privacy-Preserving Consistent Key Service for Secure End-to-End Communication. Cryptology ePrint Archive, Report 2014/1004. (December 2014). <http://eprint.iacr.org/2014/1004>.
- Microsoft. 2013. Fraudulent Digital Certificates Could Allow Spoofing. Microsoft Security Advisory 2798897. (January 2013). <https://technet.microsoft.com/en-us/library/security/2798897.aspx>.
- Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). <http://bitcoin.org/bitcoin.pdf>.
- National Institute of Standards and Technology. 2012. *Secure Hash Standard (SHS)*. Federal Information Processing Standards Publication 180-4. <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.
- William B. Norton. 2010. The Art of Peering: The Peering Playbook. (August 2010). <http://drpeering.net/white-papers/Art-Of-Peering-The-Peering-Playbook.html>.
- Karl J. O'Dwyer and David Malone. 2014. Bitcoin Mining and its Energy Footprint. In *Irish Signals and Systems Conference*. Limerick, Ireland, 280–285.
- Brian M. Oki and Barbara H. Liskov. 1988. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. In *Proceedings of the 7th Symposium on Principles of Distributed Computing*. 8–17.
- Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *2014 USENIX Annual Technical Conference*. 305–319.
- Marshall Pease, Robert Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *Journal of the ACM* 27, 2 (April 1980), 228–234.
- Claire Provost. 2013. Why do Africans pay the most to send money home? (January 2013). <http://www.theguardian.com/global-development/2013/jan/30/africans-pay-most-send-money>.
- David Schwartz, Noah Youngs, and Arthur Britto. 2014. The Ripple Protocol Consensus Algorithm. (2014). https://ripple.com/files/ripple_consensus_whitepaper.pdf.
- Dale Skeen and Michael Stonebraker. 1983. A Formal Model of Crash Recovery in a Distributed System. *IEEE Transactions on Software Engineering* 9, 3 (May 1983), 219–228.
- Robbert van Renesse, Nicolas Schiper, and Fred B. Schneider. 2014. Vive la Différence: Paxos vs. Viewstamped Replication vs. Zab. *IEEE Transactions on Dependable and Secure Computing* (September 2014).

A. GLOSSARY OF NOTATION

Notation	Name	Definition
iff		An abbreviation of “if and only if”
$f : A \rightarrow B$	function	Function f maps each element of set A to a result in set B .
$f(x)$	application	The result of calculating function f on argument x
\bar{a}	complement	An overbar connotes the opposite, i.e., \bar{a} is the opposite of a .
$\langle a_1, \dots, a_n \rangle$	tuple	A structure (compound value) with field values a_1, \dots, a_n
$A \wedge B$	logical and	Both A and B are true.
$A \vee B$	logical or	At least one, possibly both, of A and B are true.
$\exists e, C(e)$	there exists	There is at least one value e for which condition $C(e)$ is true.
$\forall e, C(e)$	for all	$C(e)$ is true of every value e .
$\{a, b, \dots\}$	set	A set containing the listed elements (a, b, \dots)
$\{e \mid C(e)\}$	set-builder	The set of all elements e for which $C(e)$ is true
\emptyset	empty set	The set containing no elements
$ S $	cardinality	The number of elements in set S
$e \in S$	element of	Element e is a member of set S .
$A \subseteq B$	subset	Every member of set A is also a member of set B .
$A \subsetneq B$	strict subset	$A \subseteq B$ and $A \neq B$.
2^A	powerset	The set of sets containing every possible combination of members of A , i.e., $2^A = \{B \mid B \subseteq A\}$
$A \cup B$	union	The set containing all elements that are members of A or members of B , i.e., $A \cup B = \{e \mid e \in A \vee e \in B\}$
$A \cap B$	intersection	The set containing all elements that are members of both A and B , i.e., $A \cap B = \{e \mid e \in A \wedge e \in B\}$
$A \setminus B$	set difference	The set containing every element of A that is not a member of B , i.e., $A \setminus B = \{e \mid e \in A \wedge e \notin B\}$
/	not	Negates a symbol’s meaning. E.g., $e \notin A$ means $e \in A$ is false, while $\exists e, C(e)$ means no e exists such that $C(e)$ is true.

Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol

Aggelos Kiayias* Alexander Russell† Bernardo David‡ Roman Oliynykov§

August 21, 2017

Abstract

We present “Ouroboros”, the first blockchain protocol based on *proof of stake* with rigorous security guarantees. We establish security properties for the protocol comparable to those achieved by the bitcoin blockchain protocol. As the protocol provides a “proof of stake” blockchain discipline, it offers qualitative efficiency advantages over blockchains based on proof of physical resources (e.g., proof of work). We also present a novel reward mechanism for incentivizing Proof of Stake protocols and we prove that, given this mechanism, honest behavior is an approximate Nash equilibrium, thus neutralizing attacks such as selfish mining. We also present initial evidence of the practicality of our protocol in real world settings by providing experimental results on transaction confirmation and processing.

1 Introduction

A primary consideration regarding the operation of blockchain protocols based on proof of work (PoW)—such as bitcoin [30]—is the energy required for their execution. At the time of this writing, generating a single block on the bitcoin blockchain requires a number of hashing operations exceeding 2^{60} , which results in striking energy demands. Indeed, early calculations indicated that the energy requirements of the protocol were comparable to that of a small country [32].

This state of affairs has motivated the investigation of alternative blockchain protocols that would obviate the need for proof of work by substituting it with another, more energy efficient, mechanism that can provide similar guarantees. It is important to point out that the proof of work mechanism of bitcoin facilitates a type of randomized “leader election” process that elects one of the miners to issue the next block. Furthermore, provided that all miners follow the protocol, this selection is performed in a randomized fashion proportionally to the computational power of each miner. (Deviations from the protocol may distort this proportionality as exemplified by “selfish mining” strategies [21, 38].)

A natural alternative mechanism relies on the notion of “proof of stake” (PoS). Rather than miners investing computational resources in order to participate in the leader election process, they instead run a process that randomly selects one of them proportionally to the *stake* that each possesses according to the current blockchain ledger.

*University of Edinburgh and IOHK. akiayias@inf.ed.ac.uk. Work partly performed while at the National and Kapodistrian University of Athens, supported by ERC project CODAMODA #259152. Work partly supported by H2020 Project #653497, PANORAMIX.

†University of Connecticut. acr@cse.uconn.edu.

‡Aarhus University and IOHK, bernardo.david@iohk.io. Work partly supported by European Research Council Starting Grant 279447.

§IOHK, roman.olynykov@iohk.io.

In effect, this yields a self-referential blockchain discipline: maintaining the blockchain relies on the stakeholders themselves and assigns work to them (as well as rewards) based on the amount of stake that each possesses as reported in the ledger. Aside from this, the discipline should make no further “artificial” computational demands on the stakeholders. In some sense, this sounds ideal; however, realizing such a proof-of-stake protocol appears to involve a number of definitional, technical, and analytic challenges.

Previous work. The concept of PoS has been discussed extensively in the bitcoin forum.¹ Proof-of-stake based blockchain design has been more formally studied by Bentov et al., both in conjunction with PoW [5] as well as the sole mechanism for a blockchain protocol [4]. Although Bentov et al. showed that their protocols are secure against some classes of attacks, they do not provide a formal model for analysing PoS based protocols or security proofs relying on precise definitions. Heuristic proof-of-stake based blockchain protocols have been proposed (and implemented) for a number of cryptocurrencies.² Being based on heuristic security arguments, these cryptocurrencies have been frequently found to be deficient from the point of view of security. See [4] for a discussion of various attacks.

It is also interesting to contrast a PoS-based blockchain protocol with a classical consensus blockchain that relies on a *fixed* set of authorities (see, e.g., [17]). What distinguishes a PoS-based blockchain from those which assume static authorities is that stake changes over time and hence the trust assumption evolves with the system.

Another alternative to PoW is the concept of *proof of space* [2, 20], which has been specifically investigated in the context of blockchain protocols [33]. In a proof of space setting, a “prover” wishes to demonstrate the utilization of space (storage / memory); as in the case of a PoW, this utilizes a physical resource but can be less energy demanding over time. A related concept is *proof of space-time* (PoST) [28]. In all these cases, however, an expensive physical resource (either storage or computational power) is necessary.

The PoS Design challenge. A fundamental problem for PoS-based blockchain protocols is to simulate the leader election process. In order to achieve a fair randomized election among stakeholders, entropy must be introduced into the system, and mechanisms to introduce entropy may be prone to manipulation by the adversary. For instance, an adversary controlling a set of stakeholders may attempt to simulate the protocol execution trying different sequences of stakeholder participants so that it finds a protocol continuation that favors the adversarial stakeholders. This leads to a so called “grinding” vulnerability, where adversarial parties may use computational resources to bias the leader election.

Our Results. We present “Ouroboros”, a provably secure proof of stake system. To the best of our knowledge this is the first blockchain protocol of its kind with a rigorous security analysis. In more detail, our results are as follows.

First, we provide a model that formalizes the problem of realizing a PoS-based blockchain protocol. The model we introduce is in the spirit of [24], focusing on *persistence* and *liveness*, two formal properties of a robust transaction ledger. *Persistence* states that once a node of the system proclaims a certain transaction as “stable”, the remaining nodes, if queried and responding honestly,

¹See “Proof of stake instead of proof of work”, Bitcoin forum thread. Posts by user “QuantumMechanic” and others. (<https://bitcointalk.org/index.php?topic=27787.0.>).

²A non-exhaustive list includes NXT, Neucoin, Blackcoin, Tendermint, Bitshares.

will also report it as stable. Here, stability is to be understood as a predicate that will be parameterized by some security parameter k that will affect the certainty with which the property holds. (E.g., “more than k blocks deep”.) *Liveness* ensures that once an honestly generated transaction has been made available for a sufficient amount of time to the network nodes, say u time steps, it will become stable. The conjunction of liveness and persistence provides a robust transaction ledger in the sense that honestly generated transactions are adopted and become immutable. Our model is suitably amended to facilitate PoS-based dynamics.

Second, we describe a novel blockchain protocol based on PoS. Our protocol assumes that parties can freely create accounts and receive and make payments, and that stake shifts over time. We utilize a (very simple) secure multiparty implementation of a coin-flipping protocol to produce the randomness for the leader election process. This distinguishes our approach (and prevents so called “grinding attacks”) from other previous solutions that either defined such values deterministically based on the current state of the blockchain or used collective coin flipping as a way to introduce entropy [4]. Also, unique to our approach is the fact that the system ignores round-to-round stake modifications. Instead, a snapshot of the current set of stakeholders is taken in regular intervals called *epochs*; in each such interval a secure multiparty computation takes place utilizing the blockchain itself as the broadcast channel. Specifically, in each epoch a set of randomly selected stakeholders form a committee which is then responsible for executing the coin-flipping protocol. The outcome of the protocol determines the set of next stakeholders to execute the protocol in the next epoch as well as the outcomes of all leader elections for the epoch.

Third, we provide a set of formal arguments establishing that no adversary can break persistence and liveness. Our protocol is secure under a number of plausible assumptions: (1) the network is synchronous in the sense that an upper bound can be determined during which any honest stakeholder is able to communicate with any other stakeholder, (2) a number of stakeholders drawn from the honest majority is available as needed to participate in each epoch, (3) the stakeholders do not remain offline for long periods of time, (4) the adaptivity of corruptions is subject to a small delay that is measured in rounds linear in the security parameter (or alternatively, the players have access to a sender-anonymous broadcast channel). At the core of our security arguments is a probabilistic argument regarding a combinatorial notion of “forkable strings” which we formulate, prove and also verify experimentally. In our analysis we also distinguish *covert attacks*, a special class of general forking attacks. “Covertness” here is interpreted in the spirit of covert adversaries against secure multiparty computation protocols, cf. [3], where the adversary wishes to break the protocol but prefers not to be caught doing so. We show that covertly forkable strings are a subclass of the forkable strings with much smaller density; this permits us to provide two distinct security arguments that achieve different trade-offs in terms of efficiency and security guarantees. Our forkable string analysis is a natural and fairly general tool that can be applied as part of a security argument the PoS setting.

Fourth, we turn our attention to the incentive structure of the protocol. We present a novel reward mechanism for incentivizing the participants to the system which we prove to be an (approximate) Nash equilibrium. In this way, attacks like *block withholding* and *selfish-mining* [21, 38] are mitigated by our design. The core idea behind the reward mechanism is to provide positive payoff for those protocol actions that cannot be stifled by a coalition of parties that diverges from the protocol. In this way, it is possible to show that, under plausible assumptions, namely that certain protocol execution costs are small, following the protocol faithfully is an equilibrium when all players are rational.

Fifth, we introduce a stake delegation mechanism that can be seamlessly added to our blockchain protocol. Delegation is particularly useful in our context as we would like to allow our protocol to scale even in a setting where the set of stakeholders is highly fragmented. In such cases, the

delegation mechanism can enable stakeholders to delegate their “voting rights”, i.e., the right of participating in the committees running the leader selection protocol in each epoch. As in *liquid democracy*, (a.k.a. delegative democracy [23]), stakeholders have the ability to revoke their delegative appointment when they wish independently of each other.

Given our model and protocol description we also explore how various attacks considered in practice can be addressed within our framework. Specifically, we discuss double spending attacks, transaction denial attacks, 51% attacks, nothing-at-stake, desynchronization attacks and others. Finally, we present evidence regarding the efficiency of our design. First we consider double spending attacks. For illustrative purposes, we perform a comparison with Nakamoto’s analysis for bitcoin regarding transaction confirmation time with assurance 99.9%. Against covert adversaries, the transaction confirmation time is from 10 to 16 times faster than that of bitcoin, depending on the adversarial hashing power; for general adversaries confirmation time is from 5 to 10 times faster. Moreover, our concrete analysis of double-spending attacks relies on our combinatorial analysis of forkable and covertly forkable strings and applies to a much broader class of adversarial behavior than Nakamoto’s more simplified analysis.³ We then survey our prototype implementation and report on benchmark experiments run in the Amazon cloud that showcase the power of our proof of stake blockchain protocol in terms of performance.

Related Work. In parallel to the development of Ouroboros, a number of other protocols were developed targeting various positions in the design space of distributed ledgers based on PoS. Sleepy consensus [6] considers a fixed stakeholder distribution (i.e., stake does not evolve over time) and targets a “mixed” corruption setting, where the adversary is allowed to be adaptive as well as perform fail-stop and recover corruptions in addition to Byzantine faults. It is actually straightforward to extend our analysis in this mixed corruption setting, cf. Remark 2; nevertheless, the resulting security can be argued only in the “corruptions with delay” setting, and thus is not fully adaptive. Snow White [7] addresses an evolving stakeholder distribution and uses a corruption delay mechanism similar to ours for arguing security. Nevertheless, contrary to our protocol, the Snow White design is susceptible to a “grinding” type of attack that can bias high probability events in favor of the adversary. While this does not hurt security asymptotically, it prevents a concrete parameterisation that does not take into account adversarial computing power. Algorand [27] provides a distributed ledger following a Byzantine agreement *per block* approach that can withstand adaptive corruptions. Given that agreement needs to be reached for each block, such protocols will produce blocks at a rate substantially slower than a PoS blockchain (where the slow down matches the expected length of the execution of the Byzantine agreement protocol) but they are free of forks. In this respect, despite the existence of forks, blockchain protocols exhibit the flexibility of permitting the clients to set the level of risk that they are willing to undertake, allowing low risk profile clients to enjoy faster processing times in the optimistic sense. Finally, Fruitchain [36] provides a reward mechanism and an approximate Nash equilibrium proof for a PoW-based blockchain. We use a similar reward mechanism at the blockchain level, nevertheless our underlying mechanics are different since we have to operate in a PoS setting. The core of the idea is to provide a PoS analogue of “endorsing” inputs in a *fair proportion* using the same logic as the PoW-based byzantine agreement protocol for honest majority from [24].

³Nakamoto’s simplifications are pointed out in [24]: the analysis considers only the setting where a block withholding attacker acts without interaction as opposed to a more general attacker that, for instance, tries strategically to split the honest parties in more than one chains during the course of the double spending attack.

Paper overview. We lay out the basic model in Sec. 2. To simplify the analysis of our protocol, we present it in four stages that are outlined in Sec. 3. In short, in Sec. 4 we describe and analyze the protocol in the static setting; we then transition to the dynamic setting in Sec. 5. Our incentive mechanism and the equilibrium argument are presented in Sec. 7. We then present the protocol enhancement with anonymous channels in Sec. 6 and with a delegation mechanism in Sec. 8. Following this, in Sec. 9 we discuss the resilience of the protocol under various particular attacks of interest. In Sec. 10 we discuss transaction confirmation times as well as general performance results obtained from a prototype implementation running in the Amazon cloud.

2 Model

Time, slots, and synchrony. We consider a setting where time is divided into discrete units called *slots*. A ledger, described in more detail below, associates with each time slot (at most) one ledger *block*. Players are equipped with (roughly synchronized) clocks that indicate the current slot. This will permit them to carry out a distributed protocol intending to collectively assign a block to this current slot. In general, each slot sl_r is indexed by an integer $r \in \{1, 2, \dots\}$, and we assume that the real time window that corresponds to each slot has the following properties.

- The current slot is determined by a publicly-known and monotonically increasing function of current time.
- Each player has access to the current time. Any discrepancies between parties' local time are insignificant in comparison with the length of time represented by a slot.
- The length of the time window that corresponds to a slot is sufficient to guarantee that any message transmitted by an honest party at the beginning of the time window will be received by any other honest party by the end of that time window (even accounting for small inconsistencies in parties' local clocks). In particular, while network delays may occur, they never exceed the slot time window.

Transaction Ledger Properties. A protocol Π implements a robust transaction ledger provided that the ledger that Π maintains is divided into “blocks” (assigned to time slots) that determine the order with which transactions are incorporated in the ledger. It should also satisfy the following two properties.

- **Persistence.** Once a node of the system proclaims a certain transaction tx as *stable*, the remaining nodes, if queried, will either report tx in the same position in the ledger or will not report as stable any transaction in conflict to tx . Here the notion of stability is a predicate that is parameterized by a security parameter k ; specifically, a transaction is declared *stable* if and only if it is in a block that is more than k blocks deep in the ledger.
- **Liveness.** If all honest nodes in the system attempt to include a certain transaction, then after the passing of time corresponding to u slots (called the transaction confirmation time), all nodes, if queried and responding honestly, will report the transaction as stable.

In [26, 35] it was shown that persistence and liveness can be derived from the following three elementary properties provided that protocol Π derives the ledger from a data structure in the form of a blockchain.

- **Common Prefix (CP); with parameters** $k \in \mathbb{N}$. The chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of the slots $sl_1 < sl_2$ are such that $\mathcal{C}_1^{[k]} \preceq \mathcal{C}_2$, where $\mathcal{C}_1^{[k]}$ denotes the chain obtained by removing the last k blocks from \mathcal{C}_1 , and \preceq denotes the prefix relation.
- **Chain Quality (CQ); with parameters** $\mu \in (0, 1]$ **and** $\ell \in \mathbb{N}$. Consider any portion of length at least ℓ of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most $1 - \mu$. We call μ the chain quality coefficient.
- **Chain Growth (CG); with parameters** $\tau \in (0, 1], s \in \mathbb{N}$. Consider the chains $\mathcal{C}_1, \mathcal{C}_2$ possessed by two honest parties at the onset of two slots sl_1, sl_2 with sl_2 at least s slots ahead of sl_1 . Then it holds that $\text{len}(\mathcal{C}_2) - \text{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.

Some remarks are in place. Regarding common prefix, we capture a strong notion of common prefix, cf. [26]. Regarding chain quality, μ , as a function of the ratio of adversarial parties, satisfies $\mu(\alpha) \geq \alpha$ for protocols of interest. In an ideal setting, μ would be $1 - \alpha$: in this case, the percentage of malicious blocks in any sufficiently long chain segment is proportional to the cumulative stake of a set of (malicious) stakeholders.

It is worth noting that for bitcoin we have $\mu(\alpha) = (1 - 2\alpha)/(1 - \alpha)$, and this bound is in fact tight—see [24], which argues this guarantee on chain quality. The same will hold true for our protocol construction. As we will show, this will still be sufficient for our incentive mechanism to work properly.

Finally chain growth concerns the rate at which the chain grows (for honest parties). As in the case of bitcoin, the *longest* chain plays a preferred role in our protocol; this provides an easy guarantee of chain growth.

Security Model. We adopt the model introduced by [24] for analysing security of blockchain protocols enhanced with an ideal functionality \mathcal{F} . We denote by $\text{VIEW}_{\Pi, \mathcal{A}, \mathcal{Z}}^{P, \mathcal{F}}(\lambda)$ the view of party P after the execution of protocol Π with adversary \mathcal{A} , environment \mathcal{Z} , security parameter κ and access to ideal functionality \mathcal{F} . Similarly we denote by $\text{EXEC}_{\Pi, \mathcal{A}, \mathcal{Z}}^{P, \mathcal{F}}(\lambda)$ the output of \mathcal{Z} .

We note that multiple different “functionalities” will be encompassed by \mathcal{F} . Contrary to [24], our analysis is in the “standard model”, and without a random oracle functionality. The first interfaces we incorporate in the ideal functionality used in the protocol are the “diffuse” and “key and transaction” functionality, denoted \mathcal{F}_{D+KT} and described below. Note that the diffuse functionality is also the mechanism via which we will obtain the synchronization of the protocol.

Diffuse functionality. The diffuse functionality maintains an *incoming string* for each party U_i that participates. A party, if activated, is allowed at any moment to fetch the contents of its incoming string; one may think of this as a mailbox. Additionally, parties can instruct the functionality to *diffuse* a message, in which case the message will be appended to each party’s incoming string. The functionality maintains rounds (slots) and all parties are allowed to diffuse once in a round. Rounds do not advance unless all parties have diffused a message. The adversary, when activated, may also interact with the functionality and is allowed to read all inboxes and all diffuse requests and deliver messages to the inboxes in any order it prefers. At the end of the round, the functionality will ensure that all inboxes contain all messages that have been diffused (but not necessarily in the same order they have been requested to be diffused). The current slot index may be requested at any time by any party. If a stakeholder does not fetch in a certain slot the messages written to its incoming string, they are flushed.

Key and Transaction functionality. The key registration functionality is initialized with n users, U_1, \dots, U_n and their respective stake s_1, \dots, s_n ; given such initialization, the functionality will consult with the adversary and will accept a (possibly empty) sequence of $(\text{Corrupt}, U)$ messages and mark the corresponding users U as corrupt. For the corrupt users without a public-key registered the functionality will allow the adversary to set their public-keys while for honest users the functionality will sample public/secret-key pairs and record them based on a digital signature algorithm. Public-keys of corrupt users will be marked as such. Subsequently, any sequence of the following actions may take place: (i) A user may request to retrieve its public and secret-key whereupon the functionality will return it to the user. (ii) The whole directory of public-keys may be required whereupon the functionality will return it to the requesting user. (iii) A new user may be requested to be created by a message $(\text{Create}, U, \mathcal{C})$ from the environment, in which case the functionality will follow the same procedure as before: it will consult the adversary regarding the corruption status of U and will set its public and possibly secret-key depending on the corruption status; moreover it will store \mathcal{C} as the suggested initial state. The functionality will return the public-key back to the environment upon successful completion of this interaction. (iv) An existing user may be requested to be corrupted by the adversary via a message $(\text{Corrupt}, U)$. A user can only be corrupted after a delay of D slots; specifically, after a corruption request is registered the secret-key will be released after D slots have passed according to the round counter maintained in the Diffuse component of the functionality.

Given the above we will assume that the execution of the protocol is with respect to a functionality \mathcal{F} that is incorporating the above two functionalities as well as possibly additional functionalities to be explained below. Note that a corrupted stakeholder U will relinquish its entire state to \mathcal{A} ; from this point on, the adversary will be activated in place of the stakeholder U . Beyond any restrictions imposed by \mathcal{F} , the adversary can only corrupt a stakeholder if it is given permission by the environment \mathcal{Z} running the protocol execution. The permission is in the form of a message $(\text{Corrupt}, U)$ which is provided to the adversary by the environment. In summary, regarding activations we have the following.

- At each slot sl_j , the environment \mathcal{Z} is allowed to activate any subset of stakeholders it wishes. Each one of them will possibly produce messages that are to be transmitted to other stakeholders.
- The adversary is activated at least as the last entity in each sl_j , (as well as during all adversarial party activations).

It is easy to see that the model above confers such sweeping power on the adversary that one cannot establish any significant guarantees on protocols of interest. It is thus important to restrict the environment suitably (taking into account the details of the protocol) so that we may be able to argue security. With foresight, the restrictions we will impose on the environment are as follows.

Restrictions imposed on the environment. The environment, which is responsible for activating the honest parties in each round, will be subject to the following constraints regarding the activation of the honest parties running the protocol.

- In each slot there will be at least one honest activated party.

- There will be a parameter $k \in \mathbb{Z}$ that will signify the maximum number of slots that an honest shareholder can be offline. In case an honest stakeholder is spawned after the beginning of the protocol via $(\text{Create}, U, \mathcal{C})$ its initialization chain \mathcal{C} provided by the environment should match an honest parties' chain which was active in the previous slot.
- In each slot sl_r , and for each active stakeholder U_j there will be a set $\mathbb{S}_j(r)$ of public-keys and stake pairs of the form $(vk_i, s_i) \in \{0, 1\}^* \times \mathbb{N}$, for $j = 1, \dots, n_r$ where n_r is the number of users introduced up to that slot that will represent who are the active participants in the view of U_j . Public-keys will be marked as “corrupted” if the corresponding stakeholder has been corrupted. We will say the adversary is restricted to less than 50% relative stake if it holds that the total stake of the corrupted keys divided by the total stake $\sum_i s_i$ is less than 50% in all possible $\mathbb{S}_j(r)$. In case the above is violated an event $\text{Bad}^{1/2}$ becomes true for the given execution.

We note that the offline restriction stated above is very conservative and our protocol can tolerate much longer offline times depending on the way the course of the execution proceeds; nevertheless, for the sake of simplicity, we use the above restriction. Finally, we note that in all our proofs, whenever we say that a property Q holds with high probability over all executions, we will in fact argue that $Q \vee \text{Bad}^{1/2}$ holds with high probability over all executions. This captures the fact that we exclude environments and adversaries that trigger $\text{Bad}^{1/2}$ with non-negligible probability.

3 Our Protocol: Overview

We first provide a general overview of our protocol design approach. The protocol's specifics depend on a number of parameters as follows: (i) k is the number of blocks a certain message should have “on top of it” in order to become part of the immutable history of the ledger, (ii) ϵ is the advantage in terms of stake of the honest stakeholders against the adversarial ones; (iii) D is the corruption delay that is imposed on the adversary, i.e., an honest stakeholder will be corrupted after D slots when a corrupt message is delivered by the adversary during an execution; (iv) L is the lifetime of the system, measured in slots; (v) R is the length of an epoch, measured in slots.

We present our protocol description in four *stages* successively improving the adversarial model it can withstand. In all stages an “ideal functionality” $\mathcal{F}_{LS}^{\mathcal{D}, F}$ is available to the participants. The functionality captures the resources that are available to the parties as preconditions for the secure operation of the protocol (e.g., the genesis block will be specified by $\mathcal{F}_{LS}^{\mathcal{D}, F}$).

Stage 1: Static stake; $D = L$. In the first stage, the trust assumption is static and remains with the initial set of stakeholders. There is an initial stake distribution which is hardcoded into the genesis block that includes the public-keys of the stakeholders, $\{(vk_i, s_i)\}_{i=1}^n$. Based on our restrictions to the environment, honest majority with advantage ϵ is assumed among those initial stakeholders. Specifically, the environment initially will allow the corruption of a number of stakeholders whose relative stake represents $\frac{1-\epsilon}{2}$ for some $\epsilon > 0$. The environment allows party corruption by providing tokens of the form $(\text{Corrupt}, U)$ to the adversary; note that due to the corruption delay imposed in this first stage any further corruptions will be against parties that have no stake initially and hence the corruption model is akin to “static corruption.” $\mathcal{F}_{LS}^{\mathcal{D}, F}$ will subsequently sample ρ which will seed a “weighted by stake” stakeholder sampling and in this way lead to the election of a subset of m keys $vk_{i_1}, \dots, vk_{i_m}$ to form the committee that will possess honest majority with overwhelming probability in m , (this uses the fact that the relative stake possessed by malicious parties is $\frac{1-\epsilon}{2}$; a linear dependency of m to ϵ^{-2} will be imposed at this

stage). In more detail, the committee will be selected implicitly by appointing a stakeholder with probability proportional to its stake to each one of the L slots. Subsequently, stakeholders will issue blocks following the schedule that is determined by the slot assignment. The longest chain rule will be applied and it will be possible for the adversary to fork the blockchain views of the honest parties. Nevertheless, we will prove with a Markov chain argument that the probability that a fork can be maintained over a sequence of n slots drops exponentially with at least \sqrt{n} , cf. Theorem 4.13 against general adversaries. An even more favorable analysis can be made against covert adversaries, i.e., adversaries that prefer to remain “under the radar” cf. Theorem 4.23.

Stage 2: Dynamic state with a beacon, epoch period of R slots, $D = R \ll L$. The central idea for the extension of the lifetime of the above protocol is to consider the sequential composition of several invocations of it. We detail a way to do that, under the assumption that a trusted beacon emits a uniformly random string in regular intervals. More specifically, the beacon, during slots $\{j \cdot R + 1, \dots, (j+1)R\}$, reveals the j -th random string that seeds the leader election function. The critical difference compared to the static state protocol is that the stake distribution is allowed to change and is drawn from the blockchain itself. This means that at a certain slot sl that belongs to the j -th epoch (with $j \geq 2$), the stake distribution that is used is the one reported in the most recent block with time stamp less than $j \cdot R - 2k$.

Regarding the evolving stake distribution, transactions will be continuously generated and transferred between stakeholders via the environment and players will incorporate posted transactions in the blockchain based ledgers that they maintain. In order to accomodate the new accounts that are being created, the $\mathcal{F}_{LS}^{\mathcal{D}, F}$ functionality enables a new (vk, sk) to be created on demand and assigned to a new party U_i . Specifically, the environment can create new parties who will interact with $\mathcal{F}_{LS}^{\mathcal{D}, F}$ for their public/secret-key in this way treating it as a trusted component that maintains the secret of their wallet. Note that the adversary can interfere with the creation of a new party, corrupt it, and supply its own (adversarially created) public-key instead. As before, the environment, may request transactions between accounts from stakeholders and it can also generate transactions in collaboration with the adversary on behalf of the corrupted accounts. Recall that our assumption is that at any slot, in the view of any honest player, the stakeholder distribution satisfies honest majority with advantage ϵ (note that different honest players might perceive a different stakeholder distribution in a certain slot). Furthermore, the stake can shift by at most σ statistical distance over a certain number of slots. The statistical distance here will be measured considering the underlying distribution to be the weighted-by-stake sampler and how it changes over the specified time interval. The security proof can be seen as an induction in the number of epochs L/R with the base case supplied by the proof of the static stake protocol. In the end we will argue that in this setting, a $\frac{1-\epsilon}{2} - \sigma$ bound in adversarial stake is sufficient for security of a single draw (and observe that the size of committee, m , now should be selected to overcome also an additive term of size $\ln(L/R)$ given that the lifetime of the systems includes such a number of successive epochs). The corruption delay remains at $D = R$ which can be selected arbitrarily smaller than L , thus enabling the adversary to perform adaptive corruptions as long as this is not instantaneous.

Stage 3: Dynamic state without a beacon, epoch period of R slots, $R = \Theta(k)$ and delay $D \in (R, 2R) \ll L$. In the third stage, we remove the dependency to the beacon, by introducing a secure multiparty protocol with “guaranteed output delivery” that simulates it. In this way, we can obtain the long-livedness of the protocol as described in the stage 2 design but only under the assumption of the stage 1 design, i.e., the mere availability of an initial random string and an initial stakeholder distribution with honest majority. The core idea is the following: given we

guarantee that an honest majority among elected stakeholders will hold with very high probability, we can further use this elected set as participants to an instance of a secure multiparty computation (MPC) protocol. This will require the choice of the length of the epoch to be sufficient so that it can accommodate a run of the MPC protocol. From a security point of view, the main difference with the previous case, is that the output of the beacon will become known to the adversary before it may become known to the honest parties. Nevertheless, we will prove that the honest parties will also inevitably learn it after a short number of slots. To account for the fact that the adversary gets this headstart (which it may exploit by performing adaptive corruptions) we increase the wait time for corruption from R to a suitable value in $(R, 2R)$ that negates this advantage and depends on the secure MPC design. A feature of this stage from a cryptographic design perspective is the use of the ledger itself for the simulation of a reliable broadcast that supports the MPC protocol.

Stage 4: Input endorsers, stakeholder delegates, anonymous communication. In the final stage of our design, we augment the protocol with two new roles for the entities that are running the protocol and consider the benefits of anonymous communication. Input-endorsers create a second layer of transaction endorsing prior to block inclusion. This mechanism enables the protocol to withstand deviations such as selfish mining and enables us to show that honest behaviour is an approximate Nash equilibrium under reasonable assumptions regarding the costs of running the protocol. Note that input-endorsers are assigned to slots in the same way that slot leaders are, and inputs included in blocks are only acceptable if they are endorsed by an eligible input-endorser. Second, the *delegation* feature allows stakeholders to transfer committee participation to selected delegates that assume the responsibility of the stakeholders in running the protocol (including participation to the MPC and issuance of blocks). Delegation naturally gives rise to “stake pools” that can act in the same way as mining pools in bitcoin. Finally, we observe that by including an anonymous communication layer we can remove the corruption delay requirement that is imposed in our analysis. This is done at the expense of increasing the online time requirements for the honest parties.⁴

4 Our Protocol: Static State

4.1 Basic Concepts and Protocol Description

We begin by describing the blockchain protocol π_{SPoS} in the “static stake” setting, where leaders are assigned to blockchain slots with probability proportional to their (fixed) initial stake which will be the effective stake distribution throughout the execution. To simplify our presentation, we abstract this leader selection process, treating it simply as an “ideal functionality” that faithfully carries out the process of randomly assigning stakeholders to slots. In the following section, we explain how to instantiate this functionality with a specific secure computation.

We remark that—even with an ideal leader assignment process—analyzing the standard “longest chain” preference rule in our PoS setting appears to require significant new ideas. The challenge arises because large collections of slots (epochs, as described above) are assigned to stakeholders at once; while this has favorable properties from an efficiency (and incentive) perspective, it furnishes the adversary a novel means of attack. Specifically, an adversary in control of a certain population of stakeholders can, at the beginning of an epoch, choose when standard “chain update” broadcast messages are delivered to honest parties with full knowledge of future assignments of slots to stakeholders. In contrast, adversaries in typical PoW settings are constrained to make such decisions

⁴In follow-up work we show how the same can be achieved efficiently, see [18].

in an online fashion. We remark that this can have a dramatic effect on the ability of an adversary to produce alternate chains; see the discussion on “forkable strings” below for detailed discussion.

In the static stake case, we assume that a fixed collection of n stakeholders U_1, \dots, U_n interact throughout the protocol. Stakeholder U_i possesses s_i stake before the protocol starts. For each stakeholder U_i a verification and signing key pair $(\text{vk}_i, \text{sk}_i)$ for a prescribed signature scheme is generated; we assume without loss of generality that the verification keys vk_1, \dots are known by all stakeholders. Before describing the protocol, we establish basic definitions following the notation of [24].

Definition 4.1 (Genesis Block). *The genesis block B_0 contains the list of stakeholders identified by their public-keys, their respective stakes $(\text{vk}_1, s_1), \dots, (\text{vk}_n, s_n)$ and auxiliary information ρ .*

With foresight we note that the auxiliary information ρ will be used to seed the slot leader election process.

Definition 4.2 (State). *A state is a string $st \in \{0, 1\}^\lambda$.*

Definition 4.3 (Block). *A block B generated at a slot $sl_i \in \{sl_1, \dots, sl_R\}$ contains the current state $st \in \{0, 1\}^\lambda$, data $d \in \{0, 1\}^*$, the slot number sl_i and a signature $\sigma = \text{Sign}_{\text{sk}_i}(st, d, sl_i)$ computed under sk_i corresponding to the stakeholder U_i generating the block.*

Definition 4.4 (Blockchain). *A blockchain (or simply chain) relative to the genesis block B_0 is a sequence of blocks B_1, \dots, B_n associated with a strictly increasing sequence of slots for which the state st_i of B_i is equal to $H(B_{i-1})$, where H is a prescribed collision-resistant hash function. The length of a chain $\text{len}(\mathcal{C}) = n$ is its number of blocks. The block B_n is the head of the chain, denoted $\text{head}(\mathcal{C})$. We treat the empty string ε as a legal chain and by convention set $\text{head}(\varepsilon) = \varepsilon$.*

Let \mathcal{C} be a chain of length n and k be any non-negative integer. We denote by $\mathcal{C}^{\lceil k}$ the chain resulting from removal of the k rightmost blocks of \mathcal{C} . If $k \geq \text{len}(\mathcal{C})$ we define $\mathcal{C}^{\lceil k} = \varepsilon$. We let $\mathcal{C}_1 \preceq \mathcal{C}_2$ indicate that the chain \mathcal{C}_1 is a prefix of the chain \mathcal{C}_2 .

Definition 4.5 (Epoch). *An epoch is a set of R adjacent slots $S = \{sl_1, \dots, sl_R\}$.*

(The value R is a parameter of the protocol we analyze in this section.)

Definition 4.6 (Adversarial Stake Ratio). *Let $U_{\mathcal{A}}$ be the set of stakeholders controlled by an adversary \mathcal{A} . Then the adversarial stake ratio is defined as*

$$\alpha = \frac{\sum_{j \in U_{\mathcal{A}}} s_j}{\sum_{i=1}^n s_i},$$

where n is the total number of stakeholders and s_i is stakeholder U_i ’s stake.

Slot Leader Selection. In the protocol described in this section, for each $0 < j \leq R$, a slot leader E_j is determined who has the (sole) right to generate a block at sl_j . Specifically, for each slot a stakeholder U_i is selected as the slot leader with probability p_i proportional to its stake registered in the genesis block B_0 ; these assignments are independent between slots. In this static stake case, the genesis block as well as the procedure for selecting slot leaders are determined by an ideal functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}$, defined in Figure 1. This functionality is parameterized by the list $\{(\text{vk}_1, s_1), \dots, (\text{vk}_n, s_n)\}$ assigning to each stakeholder its respective stake, a distribution \mathcal{D} that provides auxiliary information ρ and a *leader selection function* F defined below.

Definition 4.7 (Leader Selection Process). A leader selection process *with respect to stakeholder distribution* $\mathbb{S} = \{(\text{vk}_1, s_1), \dots, (\text{vk}_n, s_n)\}$, (\mathcal{D}, F) is a pair consisting of a distribution and a deterministic function such that, when $\rho \leftarrow \mathcal{D}$ it holds that for all $sl_j \in \{sl_1, \dots, sl_R\}$, $F(\mathbb{S}, \rho, sl_j)$ outputs $U_i \in \{U_1, \dots, U_n\}$ with probability

$$p_i = \frac{s_i}{\sum_{k=1}^n s_k}$$

where s_i is the stake held by stakeholder U_i (we call this “weighing by stake”); furthermore the family of random variables $\{F(\mathbb{S}, \rho, sl_j)\}_{j=1}^R$ are independent.

We note that sampling proportional to stake can be implemented in a straightforward manner. For instance, a simple process operates as follows. Let $\tilde{p}_i = s_i / \sum_{j=i}^n s_j$. For each $i = 1, \dots, n-1$, provided that no stakeholder has yet been selected, the process flips a \tilde{p}_i -biased coin; if the result of the coin is 1, the party U_i is selected for the slot and the process is complete. (Note that $\tilde{p}_n = 1$, so the process is certain to complete with a unique leader.) When we implement this process as a function $F(\cdot)$, sufficient randomness must be allocated to simulate the biased coin flips. If we implement the above with λ precision for each individual coin flip, then selecting a stakeholder will require $n \lceil \log \lambda \rceil$ random bits in total. Note that using a pseudorandom number generator (PRG) one may use a shorter “seed” string and then stretch it using the PRG to the appropriate length.

Functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$

$\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$ incorporates the diffuse and key/transaction functionality \mathcal{F}_{D+KT} from Section 2 and is parameterized by the public keys and respective stakes of the initial stakeholders $\mathbb{S}_0 = \{(U_1, s_1), \dots, (U_n, s_n)\}$, a distribution \mathcal{D} and a function F so that (\mathcal{D}, F) is a leader selection process. In addition, $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$ is parameterized by **mode**, which determines how signature verification keys are generated. When $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$ is instantiated with **mode** = **SIG** (resp. **mode** = \mathcal{F}_{DSIG}) it is denoted $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$ (resp. $\mathcal{F}_{LS}^{\mathcal{D}, F}[\mathcal{F}_{DSIG}]$). $\mathcal{F}_{LS}^{\mathcal{D}, F}$ interacts with stakeholders as follows:

- **Signature Key Pair Generation:** $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$ generates signing and verification keys sk_i, vk_i for stakeholder U_i by executing $\text{KG}(1^\kappa)$ for $i = 1, \dots, n$. $\mathcal{F}_{LS}^{\mathcal{D}, F}[\mathcal{F}_{DSIG}]$ generates $(\text{sk}_i, \text{vk}_i)$ by querying \mathcal{F}_{DSIG} (Figure 3) with (KeyGen, sid_i) on behalf of U_i (with a unique session identifier sid_i related to U_i) and setting $(\text{sk}_i = sid_i, \text{vk}_i = v_i)$ (received from \mathcal{F}_{DSIG} as response) for $i = 1, \dots, n$. $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$ sets $\mathbb{S}'_0 = \{(\text{vk}_1, s_1), \dots, (\text{vk}_n, s_n)\}$.
- **Genesis Block Generation** Upon receiving $(\text{genblock_req}, U_i)$ from stakeholder U_i , $\mathcal{F}_{LS}^{\mathcal{D}, F}$ proceeds as follows. If ρ has not been set, $\mathcal{F}_{LS}^{\mathcal{D}, F}$ samples $\rho \leftarrow \mathcal{D}$. In any case, $\mathcal{F}_{LS}^{\mathcal{D}, F}$ sends $(\text{genblock}, \mathbb{S}'_0, \rho, F)$ to U_i .
- **Signatures and Verification.** $\mathcal{F}_{LS}^{\mathcal{D}, F}[\mathcal{F}_{DSIG}]$ provides access to the \mathcal{F}_{DSIG} interface.

Figure 1: Functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$.

A Protocol in the $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{mode}]$ -hybrid model. We start by describing a simple PoS based blockchain protocol considering static stake in the $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$ -hybrid model, i.e., where the genesis block B_0 (and consequently the slot leaders) are determined by the ideal functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$. $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$ provides the stakeholders with a genesis block containing a stake distribution indexed by signature verification keys generated by a EUF-CMA signature scheme, while $\mathcal{F}_{LS}^{\mathcal{D}, F}[\mathcal{F}_{DSIG}]$ obtains such keys from a signature ideal functionality \mathcal{F}_{DSIG} . This subtle difference comes into play when

describing an ideal version of π_{SPoS} used in an intermediate hybrid argument of the security proof, which will be discussed in Section 4.2. The stakeholders U_1, \dots, U_n interact among themselves and with $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}$ through Protocol π_{SPoS} described in Figure 2.

The protocol relies on a $\text{maxvalid}_S(\mathcal{C}, \mathbb{C})$ function that chooses a chain given the current chain \mathcal{C} and a set of valid chains \mathbb{C} that are available in the network. In the static case we analyze the simple “longest chain” rule. (In the dynamic case the rule is parameterized by a common chain length; see Section 5.)

Function $\text{maxvalid}(\mathcal{C}, \mathbb{C})$: Returns the longest chain from $\mathbb{C} \cup \{\mathcal{C}\}$. Ties are broken in favor of \mathcal{C} , if it has maximum length, or arbitrarily otherwise.

Protocol π_{SPoS}

π_{SPoS} is a protocol run by stakeholders U_1, \dots, U_n interacting with $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\text{SIG}]$ over a sequence of slots $S = \{sl_1, \dots, sl_R\}$. π_{SPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \dots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\text{genblock_req}, U_i)$ to $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\text{SIG}]$, receiving $(\text{genblock}, \mathbb{S}_0, \rho, \mathcal{F})$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain \mathcal{C} , sets the local blockchain to \mathcal{C} and the initial internal state $st = H(\text{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl_j \in S$, every stakeholder U_i performs the following steps:
 - (a) Collect all valid chains received via broadcast into a set \mathbb{C} , verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\text{Vrf}_{\text{vk}'}(\sigma', (st', d', sl')) = 1$, where vk' is the verification key of the stakeholder $U' = \mathcal{F}(\mathbb{S}_0, \rho, sl')$. U_i computes $\mathcal{C}' = \text{maxvalid}(\mathcal{C}, \mathbb{C})$, sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
 - (b) If U_i is the slot leader determined by $\mathcal{F}(\mathbb{S}_0, \rho, sl_j)$, it generates a new block $B = (st, d, sl_j, \sigma)$ where st is its current state, $d \in \{0, 1\}^*$ is the transaction data and $\sigma = \text{Sign}_{\text{sk}_i}(st, d, sl_j)$ is a signature on (st, d, sl_j) . U_i computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts \mathcal{C}' , sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
3. **Transaction generation** Given a transaction template tx , U_i returns $\sigma = \text{Sign}_{\text{sk}_i}(tx)$, provided that tx is consistent with the state of the ledger in the view of U_i .

Figure 2: Protocol π_{SPoS} .

4.2 Security Analysis of an Ideal Protocol

As a first step of the security analysis of π_{SPoS} , we will introduce an *idealized protocol* π_{iSPoS} and present an intermediate hybrid argument that shows that it is computationally indistinguishable from π_{SPoS} . Instead of relying on $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\text{SIG}]$ and an EUF-CMA signature scheme, π_{iSPoS} operates with an ideal signature scheme. To that end, π_{iSPoS} interacts with $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\mathcal{F}_{\text{DSIG}}]$ for obtaining signing and verification keys for the ideal signature scheme employed in the protocol. In the next sessions, we will prove that π_{iSPoS} is secure through a series of combinatorial arguments. The reason we first present this hybrid is that we intend to insulate these combinatorial arguments from the specific details of the underlying signature schemes used to instantiate π_{SPoS} and the biases that these schemes might introduce in the distributions of π_{SPoS} , concentrating instead on idealized executions where signature schemes are perfectly realized, which reflects the true nature of our protocol.

Functionality $\mathcal{F}_{\text{DSIG}}$

$\mathcal{F}_{\text{DSIG}}$ interacts with stakeholders as follows:

- **Key Generation** Upon receiving a message (KeyGen, sid) from a stakeholder U_i , verify that $sid = (U_i, sid')$ for some sid' . If not, then ignore the request. Else, hand (KeyGen, sid) to the adversary. Upon receiving $(\text{VerificationKey}, sid, v)$ from the adversary, output $(\text{VerificationKey}, sid, v)$ to U_i , and record the pair (U_i, v) .
- **Signature Generation** Upon receiving a message (Sign, sid, m) from U_i , verify that $sid = (U_i, sid')$ for some sid' . If not, then ignore the request. Else, send (Sign, sid, m) to the adversary. Upon receiving $(\text{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to U_i and halt. Else, output $(\text{Signature}, sid, m, \sigma)$ to U_i , and record the entry $(m, \sigma, v, 0)$.
- **Signature Verification** Upon receiving a message $(\text{Verify}, sid, m, \sigma, v')$ from some stakeholder U_i , hand $(\text{Verify}, sid, m, \sigma, v')$ to the adversary. Upon receiving $(\text{Verified}, sid, m, \phi)$ from the adversary do:
 1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m , then the verification succeeds.)
 2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m , then the verification fails.)
 3. Else, if there is an entry (m, σ, v', f') recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
 4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ) .

Output $(\text{Verified}, sid, m, f)$ to U_i .

Figure 3: Functionality $\mathcal{F}_{\text{DSIG}}$.

First, in Figure 3, we present Functionality $\mathcal{F}_{\text{DSIG}}$ as defined in [14], where it is also shown that EUF-CMA signature schemes realize $\mathcal{F}_{\text{DSIG}}$. Notice that this fact will be used to show that our idealized protocol can actually be realized based on practical digital signature schemes such DSA and ECDSA) and ultimately that π_{iSPoS} is indistinguishable from π_{SPoS} .

The idealized protocol π_{iSPoS} is run by the stakeholders interacting with $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\mathcal{F}_{\text{DSIG}}]$ and $\mathcal{F}_{\text{DSIG}}$. Basically, π_{iSPoS} behaves exactly as π_{SPoS} except for calls to $\text{Vrf}_{\text{vk}}(\sigma)$ and $\text{Sign}_{\text{sk}}(m)$. Namely, instead of locally computing $\text{Sign}_{\text{sk}_i}(m)$, U_i sends (Sign, sid, m) to $\mathcal{F}_{\text{DSIG}}$, receiving $(\text{Signature}, sid, m, \sigma)$ and outputting σ as the signature. Moreover, instead locally computing $\text{Vrf}_{\text{vk}'}(\sigma, m)$, U_i sends $(\text{Verify}, sid_i, m, \sigma, v')$ to $\mathcal{F}_{\text{DSIG}}$ (where v' corresponds to verification key vk'), outputting the value f received in message $(\text{Verified}, sid_i, m, f)$. Protocol π_{iSPoS} is described in Figure 4. This idealized description will be further developed when arguing about the dynamic stake case, where additional building blocks must be considered in the idealized protocol.

The following proposition is an immediate corollary of the results in [14] showing that EUF-CMA signature schemes realize $\mathcal{F}_{\text{DSIG}}$.

Proposition 4.8. *For each PPT \mathcal{A}, \mathcal{Z} it holds that there is a PPT \mathcal{S} so that $\text{EXEC}_{\pi_{\text{SPoS}}, \mathcal{A}, \mathcal{Z}}^{P, \mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\text{SIG}]}(\lambda)$ and $\text{EXEC}_{\pi_{\text{iSPoS}}, \mathcal{S}, \mathcal{Z}}^{P, \mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\mathcal{F}_{\text{DSIG}}]}(\lambda)$ are computationally indistinguishable.*

In light of the above proposition in the remaining of the analysis we will focus on the properties

Protocol π_{iSPoS}

π_{iSPoS} is a protocol run by stakeholders U_1, \dots, U_n interacting with $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\mathcal{F}_{\text{DSIG}}]$ over a sequence of slots $S = \{sl_1, \dots, sl_R\}$. π_{iSPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \dots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\text{genblock.req}, U_i)$ to $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}[\mathcal{F}_{\text{DSIG}}]$, receiving $(\text{genblock}, \mathbb{S}_0, \rho, \mathcal{F})$ as answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain \mathcal{C} , sets the local blockchain to \mathcal{C} and the initial internal state $st = H(\text{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl_j \in S$, every stakeholder U_i performs the following steps:
 - (a) Collect all valid chains received via broadcast into a set \mathbb{C} , verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\mathcal{F}_{\text{DSIG}}$ answers with $(\text{Verified}, sid, (st', d', sl'), 1)$ upon being queried with $(\text{Verify}, sid, (st', d', sl'), \sigma', \text{vk}')$, where vk' is the verification key of the stakeholder $U' = \mathcal{F}(\mathbb{S}_0, \rho, sl')$. U_i computes $\mathcal{C}' = \text{maxvalid}(\mathcal{C}, \mathbb{C})$, sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
 - (b) If U_i is the slot leader determined by $\mathcal{F}(\mathbb{S}_0, \rho, sl_j)$, it generates a new block $B = (st, d, sl_j, \sigma)$ where st is its current state, $d \in \{0, 1\}^*$ is the transaction data and σ is obtained from $\mathcal{F}_{\text{DSIG}}$'s answer $(\text{Signature}, sid, (st, d, sl_j), \sigma)$ upon being queried with $(\text{Sign}, sid_i, (st, d, sl_j))$. U_i computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts \mathcal{C}' , sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
3. **Transaction generation** Given a transaction template tx , U_i returns σ obtained from $\mathcal{F}_{\text{DSIG}}$'s answer $(\text{Signature}, sid_i, tx, \sigma)$ upon being queried with (Sign, sid_i, tx) , provided that tx is consistent with the state of the ledger in the view of U_i .

Figure 4: Protocol π_{iSPoS} .

of the protocol π_{iSPoS} (note that this implication does not apply to any⁵ possible property one might consider in an execution for π_{iSPoS} ; nevertheless the properties we will prove for π_{iSPoS} are all verifiable by the environment \mathcal{Z} and as a result they can be inherited by π_{SPoS} due to proposition).

4.3 Forkable Strings

In our security arguments we routinely use elements of $\{0, 1\}^n$ to indicate which slots—among a particular window of slots of length n —have been assigned to adversarial stakeholders. When strings have this interpretation we refer to them as *characteristic strings*.

Definition 4.9 (Characteristic String). *Fix an execution with genesis block B_0 , adversary \mathcal{A} , and environment \mathcal{Z} . Let $S = \{sl_{i+1}, \dots, sl_{i+n}\}$ denote a sequence of slots of length $|S| = n$. The characteristic string $w \in \{0, 1\}^n$ of S is defined so that $w_k = 1$ if and only if the adversary controls the slot leader of slot sl_{i+k} . For such a characteristic string $w \in \{0, 1\}^*$ we say that the index i is adversarial if $w_i = 1$ and honest otherwise.*

We start with some intuition on our approach to analyze the protocol. Let $w \in \{0, 1\}^n$ be a characteristic string for a sequence of slots S . Consider two observers that (i.) go offline immediately prior to the commencement of S , (ii.) have the same view \mathcal{C}_0 of the current chain prior to the commencement of S , and (iii.) come back online at the last slot of S and request an update of their chain. A fundamental concern in our analysis is the possibility that such observers can be presented

⁵An example of such a property would be a property testing a non-trivial fact about the parties' private states.

with a “diverging” view over the sequence S : specifically, the possibility that the adversary can force the two observers to adopt two different chains $\mathcal{C}_1, \mathcal{C}_2$ whose common prefix is \mathcal{C}_0 .

We observe that not all characteristic strings permit this. For instance the (entirely honest) string 0^n ensures that the two observers will adopt the same chain \mathcal{C} which will consist of n new blocks on top of the common prefix \mathcal{C}_0 . On the other hand, other strings do not guarantee such common extension of \mathcal{C}_0 ; in the case of 1^n , it is possible for the adversary to produce two completely different histories during the sequence of slots S and thus furnish to the two observers two distinct chains $\mathcal{C}_1, \mathcal{C}_2$ that only share the common prefix \mathcal{C}_0 . In the remainder of this section, we establish that strings that permit such “forkings” are quite rare—indeed, we show that they have density $2^{-\Omega(\sqrt{n})}$ so long as the fraction of adversarial slots is $1/2 - \epsilon$.

To reason about such “forkings” of a characteristic string $w \in \{0, 1\}^n$, we define below a formal notion of “fork” that captures the relationship between the chains broadcast by *honest* slot leaders during an execution of the protocol π_{iSPoS} . In preparation for the definition, we recall that honest players always choose to extend a maximum length chain among those available to the player on the network. Furthermore, if such a maximal chain \mathcal{C} includes a block B previously broadcast by an honest player, the prefix of \mathcal{C} prior to B must entirely agree with the chain (terminating at B) broadcast by this previous honest player. This “confluence” property follows immediately from the fact that the state of any honest block effectively commits to a unique chain beginning at the genesis block. To conclude, any chain \mathcal{C} broadcast by an honest player must begin with a chain produced by a previously honest player (or, alternatively, the genesis block), continue with a possibly empty sequence of adversarial blocks and, finally, terminate with an honest block. It follows that the chains broadcast by honest players form a natural directed tree. The fact that honest players reliably broadcast their chains and always build on the longest available chain introduces a second important property of this tree: the “depths” of the various honest blocks added by honest players during the protocol must all be distinct.

Of course, the actual chains induced by an execution of π_{iSPoS} are comprised of blocks containing a variety of data that are immaterial for reasoning about forking. For this reason the formal notion of fork below merely reflects the directed tree formed by the relevant chains and the *identities of the players*—expressed as indices in the string w —responsible for generating the blocks in these chains.

Forks and forkable strings. We define, below, the basic combinatorial structures we use to reason about the possible views observed by honest players during a protocol execution with this characteristic string.

Definition 4.10 (Fork). *Let $w \in \{0, 1\}^n$ and let $H = \{i \mid w_i = 0\}$ denote the set of honest indices. A fork for the string w is a directed, rooted tree $F = (V, E)$ with a labeling $\ell : V \rightarrow \{0, 1, \dots, n\}$ so that*

- each edge of F is directed away from the root;
- the root $r \in V$ is given the label $\ell(r) = 0$;
- the labels along any directed path in the tree are strictly increasing;
- each honest index $i \in H$ is the label of exactly one vertex of F ;
- the function $\mathbf{d} : H \rightarrow \{1, \dots, n\}$, defined so that $\mathbf{d}(i)$ is the depth in F of the unique vertex v for which $\ell(v) = i$, is strictly increasing. (Specifically, if $i, j \in H$ and $i < j$, then $\mathbf{d}(i) < \mathbf{d}(j)$.)

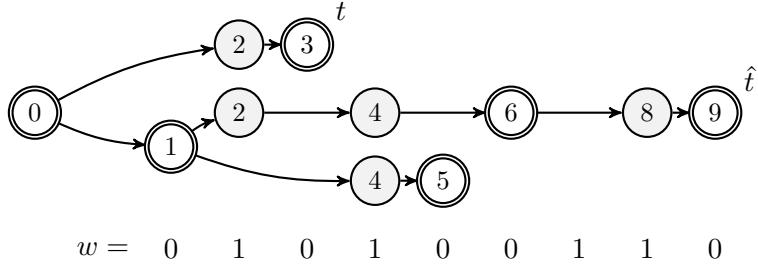


Figure 5: A fork F for the string $w = 010100110$; vertices appear with their labels and honest vertices are highlighted with double borders. Note that the depths of the (honest) vertices associated with the honest indices of w are strictly increasing. Two tines are distinguished in the figure: one, labeled \hat{t} , terminates at the vertex labeled 9 and is the longest tine in the fork; a second tine t terminates at the vertex labeled 3. The quantity $\text{gap}(t)$ indicates the difference in length between t and \hat{t} ; in this case $\text{gap}(t) = 4$. The quantity $\text{reserve}(t) = |\{i \mid \ell(v) < i \leq |w| \text{ and } w_i = 1\}|$ indicates the number of adversarial indices appearing after the label of the last honest vertex v of the tine; in this case $\text{reserve}(t) = 3$. As each leaf of F is honest, F is closed.

As a matter of notation, we write $F \vdash w$ to indicate that F is a fork for the string w . We say that a fork is trivial if it contains a single vertex, the root.

Definition 4.11 (Tines, depth, and height; the \sim relation). *A path in a fork F originating at the root is called a tine. For a tine t we let $\text{length}(t)$ denote its length, equal to the number of edges on the path. For a vertex v , we let $\text{depth}(v)$ denote the length of the (unique) tine terminating at v . The height of a fork (as usual for a tree) is defined to be the length of the longest tine.*

We overload the notation $\ell()$ so that it applies to tines, by defining $\ell(t) \triangleq \ell(v)$, where v is the terminal vertex on the tine t . For two tines t_1 and t_2 of a fork F , we write $t_1 \sim t_2$ if they share an edge. Note that \sim is an equivalence relation on the set of nontrivial tines; on the other hand, if t_ϵ denotes the “empty” tine consisting solely of the root vertex then $t_\epsilon \not\sim t$ for any tine t .

If a vertex v of a fork is labeled with an adversarial index (i.e., $w_{\ell(v)} = 1$) we say that the vertex is *adversarial*; otherwise, we say that the vertex is *honest*. For convenience, we declare the root vertex to be honest. We extend this terminology to tines: a tine is *honest* if it terminates with an honest vertex and *adversarial* otherwise. By this convention the empty tine t_ϵ is honest.

See Figure 5 for an example, which also demonstrates some of the quantities defined above and in the remainder of this section. The fork shown in the figure reflects an execution in which (i.) the honest player associated with the first slot builds directly on the genesis block (as it must), (ii.) the honest player associated with the third slot is shown a chain of length 1 produced by the adversarial player of slot 2 (in addition to the honestly generated chain of step (i.)), which it elects to extend, (iii.) the honest player associated with slot 5 is shown a chain of length 2 building on the chain of step (i.) augmented with a further adversarial block produced by the player of slot 4, etc.

Definition 4.12. *We say that a fork is flat if it has two tines $t_1 \not\sim t_2$ of length equal to the height of the fork. A string $w \in \{0, 1\}^*$ is said to be forkable if there is a flat fork $F \vdash w$.*

Note that in order for an execution of π_{iSPOS} to yield two entirely disjoint chains of maximum length, the characteristic string associated with the execution must be forkable. Our goal is to establish the following upper bound on the number of forkable strings.

Theorem 4.13. Let $\epsilon \in (0, 1)$ and let w be a string drawn from $\{0, 1\}^n$ by independently assigning each $w_i = 1$ with probability $(1 - \epsilon)/2$. Then $\Pr[w \text{ is forkable}] = 2^{-\Omega(\sqrt{n})}$.

Note that in subsequent work, Russell et al. [37] improved this bound to $2^{-\Omega(n)}$.

Structural features of forks: closed forks, prefixes, reach, and margin. We begin by defining a natural notion of inclusion for two forks:

Definition 4.14 (Fork prefixes). If w is a prefix of the string $w' \in \{0, 1\}^*$, $F \vdash w$, and $F' \vdash w'$, we say that F is a prefix of F' , written $F \sqsubseteq F'$, if F is a consistently-labeled subgraph of F' . Specifically, every vertex and edge of F appears in F' and, furthermore, the labels given to any vertex appearing in both F and F' are identical.

If $F \sqsubseteq F'$, each tine of F appears as the prefix of a tine in F' . In particular, the labels appearing on any tine terminating at a common vertex are identical and, moreover, the depth of any honest vertex appearing in both F and F' is identical.

In many cases, it is convenient to work with forks that do not “commit” anything beyond final honest indices.

Definition 4.15 (Closed forks). A fork is closed if each leaf is honest. By convention the trivial fork, consisting solely of a root vertex, is closed.

Note that a closed fork has a unique longest tine (as all maximal tines terminate with an honest vertex, and these must have distinct depths). Note, additionally, that if \check{w} is a prefix of w and $F \vdash w$, then there is a unique closed fork $\check{F} \vdash \check{w}$ for which $\check{F} \sqsubseteq F$. In particular, taking $\check{w} = w$, we note that for any fork $F \vdash w$, there is a unique closed fork $\overline{F} \vdash w$ for which $\overline{F} \sqsubseteq F$; in this case we say that \overline{F} is the closure of F .

Definition 4.16 (Gap, reserve and reach). Let $F \vdash w$ be a closed fork and let \hat{t} denote the (unique) tine of maximum length in F . We define the gap of a tine t , denoted $\text{gap}(t)$, to be the difference in length between \hat{t} and t ; thus

$$\text{gap}(t) = \text{length}(\hat{t}) - \text{length}(t).$$

We define the reserve of a tine t to be the number of adversarial indices appearing in w after the last index in t ; specifically, if t is given by the path (r, v_1, \dots, v_k) , where r is the root of F , we define

$$\text{reserve}(t) = |\{i \mid w_i = 1 \text{ and } i > \ell(v_k)\}|.$$

We remark that this quantity depends both on F and the specific string w associated with F . Finally, for a tine t we define

$$\text{reach}(t) = \text{reserve}(t) - \text{gap}(t).$$

Definition 4.17 (Margin). For a closed fork $F \vdash w$ we define $\rho(F)$ to be the maximum reach taken over all tines in F :

$$\rho(F) = \max_t \text{reach}(t).$$

Likewise, we define the margin of F , denoted $\mu(F)$, to be the “penultimate” reach taken over edge-disjoint tines of F : specifically,

$$\text{margin}(F) = \mu(F) = \max_{t_1 \neq t_2} (\min\{\text{reach}(t_1), \text{reach}(t_2)\}). \quad (1)$$

We remark that the maxima above can always be obtained by honest tines. Specifically, if t is an adversarial tine of a fork $F \vdash w$, $\text{reach}(t) \leq \text{reach}(\bar{t})$, where \bar{t} is the longest honest prefix of t .

As \sim is an equivalence relation on the nonempty tines, it follows that there is always a pair of (edge-disjoint) tines t_1 and t_2 achieving the maximum in the defining equation (1) which satisfy $\text{reach}(t_1) = \rho(F) \geq \text{reach}(t_2) = \mu(F)$.

The relevance of margin to the notion of forkability is reflected in the following proposition.

Proposition 4.18. *A string w is forkable if and only if there is a closed fork $F \vdash w$ for which $\text{margin}(F) \geq 0$.*

Proof. If w has no honest indices, then the trivial fork consisting of a single root node is flat, closed, and has non-negative margin; thus the two conditions are equivalent. Consider a forkable string w with at least one honest index and let \hat{i} denote the largest honest index of w . Let F be a flat fork for w and let $\bar{F} \vdash w$ be the closure of F (obtained from F by removing any adversarial vertices from the ends of the tines of F). Note that the tine \hat{t} containing \hat{i} is the longest tine in \bar{F} , as this is the largest honest index of w . On the other hand, F is flat, in which case there are two edge-disjoint tines t_1 and t_2 with length at least that of \hat{t} . The prefixes of these two tines in \bar{F} must clearly have reserve no less than gap (and hence non-negative reach); thus $\text{margin}(\bar{F}) \geq 0$ as desired.

On the other hand, suppose w has a closed fork with $\text{margin}(F) \geq 0$, in which case there are two edge-disjoint tines of F , t_1 and t_2 , for which $\text{reach}(t_i) \geq 0$. Then we can produce a flat fork by simply adding to each t_i a path of $\text{gap}(t_i)$ vertices labeled with the subsequent adversarial indices promised by the definition of $\text{reserve}()$. \square

In light of this proposition, for a string w we focus our attention on the quantities

$$\rho(w) = \max_{\substack{F \vdash w, \\ F \text{ closed}}} \rho(F), \quad \mu(w) = \max_{\substack{F \vdash w, \\ F \text{ closed}}} \mu(F),$$

and, for convenience,

$$\mathbf{m}(w) = (\rho(w), \mu(w)).$$

Note that this overloads the notation $\rho(\cdot)$ and $\mu(\cdot)$ so that they apply to both forks and strings, but the setting will be clear from context. We remark that the definitions do not guarantee a priori that $\rho(w)$ and $\mu(w)$ can be achieved by the same fork, though this will be established in the lemma below. In any case, it is clear that $\rho(w) \geq 0$ and $\rho(w) \geq \mu(w)$ for all strings w ; furthermore, by Proposition 4.18 a string w is forkable if and only if $\mu(w) \geq 0$. We refer to $\mu(w)$ as the *margin* of the string w .

In preparation for the proof of Theorem 4.13, we establish a recursive description for these quantities.

Lemma 4.19. $\mathbf{m}(\epsilon) = (0, 0)$ and, for all nonempty strings $w \in \{0, 1\}^*$,

$$\begin{aligned} \mathbf{m}(w1) &= (\rho(w) + 1, \mu(w) + 1), \text{ and} \\ \mathbf{m}(w0) &= \begin{cases} (\rho(w) - 1, 0) & \text{if } \rho(w) > \mu(w) = 0, \\ (0, \mu(w) - 1) & \text{if } \rho(w) = 0, \\ (\rho(w) - 1, \mu(w) - 1) & \text{otherwise.} \end{cases} \end{aligned}$$

Furthermore, for every string w , there is a closed fork $F_w \vdash w$ for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$.

Proof. The proof proceeds by induction. If $w = \epsilon$, define F_ϵ to be the trivial fork; $F_\epsilon \vdash w$ is the unique closed fork for this string and $\mathbf{m}(\epsilon) = (0, 0) = (\rho(F_\epsilon), \mu(F_\epsilon))$, as desired.

In general, we consider $\mathbf{m}(w')$ for a string $w' = wx$ —where $w \in \{0, 1\}^*$ and $x \in \{0, 1\}$; the argument recursively expands $\mathbf{m}(w')$ in terms of $\mathbf{m}(w)$ and the value of the last symbol x . In each case, we consider the relationship between two closed forks $F \sqsubseteq F'$ where $F \vdash w$ and $F' \vdash w' = wx$.

In the case where $x = 1$, we must have $F = F'$ as graphs, because the forks are assumed to be closed; it is easy to see that the reach of any tine t of $F \vdash w$ has increased by exactly one when viewed as a tine of $F' \vdash w'$. We write $\text{reach}_{F'}(t) = \text{reach}_F(t) + 1$, where we introduce the notation $\text{reach}_\square()$ to denote the reach in a particular fork. It follows that $\rho(F') = \rho(F) + 1$ and $\mu(F') = \mu(F) + 1$. If $F^* \vdash w'$ is a closed fork for which $\rho(F^*) = \rho(w')$, note that F^* may be treated as a fork for w and, applying the argument above, we find that $\rho(w') \leq \rho(w) + 1$. A similar argument implies that $\mu(w') \leq \mu(w) + 1$. On the other hand, by induction there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$ and hence $\mathbf{m}(w') \geq (\rho(w) + 1, \mu(w) + 1)$. We conclude that

$$\mathbf{m}(w') = (\rho(w) + 1, \mu(w) + 1). \quad (2)$$

Moreover, $\mathbf{m}(w') = (\rho(F_w), \mu(F_w))$, where F_w is treated as a fork for $w' = w1$.

The case when $x = 0$ is more delicate. As above, we consider the relationship between two closed forks $F \vdash w$ and $F' \vdash w' = w0$ for which $F \sqsubseteq F'$. Here F' is necessarily obtained from F by appending a path labeled with a string of the form $1^a 0$ to the end of a tine t of F . (In fact, it is easy to see that we may always assume that this is appended to an honest tine.) In order for this to be possible, $\text{gap}(t) \leq \text{reserve}(t)$ (which is to say that $\text{reach}(t) \geq 0$) and, in particular, $\text{gap}(t) \leq a \leq \text{reserve}(t)$: for the first inequality, note that the depth of the new honest vertex must exceed that of the deepest (honest) vertex in F and hence $a \geq \text{gap}(t)$; as for the second inequality, there are only $\text{reserve}(t)$ possible adversarial indices that may be added to t and hence $a \leq \text{reserve}(t)$. We define the quantity $\tilde{a} \geq 0$ by the equation $a = \text{gap}(t) + \tilde{a}$ and let t' denote the tine (of F') resulting by extending t in this way. We say that \tilde{a} is the *parameter* for this pair of forks $F \sqsubseteq F'$.

Of course, every honest tine t of F is an honest tine of F' and it is clear that $\text{reach}_{F'}(t) = \text{reach}_F(t) - (\tilde{a} + 1)$, as the length of the longest tine t' in F' exceeds the length of the longest tine of F by exactly $\tilde{a} + 1$. Note that the reach of the new honest tine t' (in F') is always 0, as both $\text{gap}(t')$ and $\text{reserve}(t')$ are zero. It remains to describe how $\mu(w)$ and $\rho(w)$ are determined by this process.

The case $\rho(w) > \mu(w) = 0$. By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$.

Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \text{reach}(t_1)$ and $\mu(F_w) = \text{reach}(t_2)$. Define $F' \vdash w'$ to be the fork obtained by extending the tine t_2 of F_w with parameter $\tilde{a} = 0$ to yield a new tine t'_2 in F' . Then $\text{reach}_{F'}(t_1) = \rho(w) - 1$ and $\text{reach}_{F'}(t'_2) = 0$. It follows that $\rho(w0) \geq \rho(w) - 1$ and $\mu(w0) \geq 0$. We will show that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq 0$, in which case we can conclude that

$$\rho(w0) = \rho(w) - 1 \quad \text{and} \quad \mu(w0) = 0.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq 0$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let \tilde{a} denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\text{reach}_{F^*}(t^*) = \rho(w0)$. If t^* is a tine of F , $\text{reach}_{F^*}(t^*) = \text{reach}_F(t^*) - (\tilde{a} + 1) \leq \rho(w) - 1$. Otherwise t^* was obtained by extension and $\text{reach}_{F^*}(t^*) = 0 \leq \rho(w) - 1$ by assumption. In

either case $\rho(w0) \leq \rho(w) - 1$, as desired. It remains to show that $\mu(w0) \leq 0$. Now consider $F^* \vdash w0$ to be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\text{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\text{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let \tilde{a} be the parameter for this extension. If both t_1^* and t_2^* are tines of F , $\text{reach}_{F^*}(t_i^*) = \text{reach}_F(t_i^*) - (\tilde{a} + 1)$ and, in particular, $\text{reach}_F(t_1^*) \geq \text{reach}_F(t_2^*)$. It follows that $\text{reach}_F(t_2^*) \leq \mu(F) \leq \mu(w) = 0$ and hence that $\mu(w0) < 0$. Otherwise, one of the two tines was the result of extension and has zero reach() in F^* . As $\text{reach}_{F^*}(t_1^*) \geq \text{reach}_{F^*}(t_2^*)$, in either case it follows that $\mu(F^*) = \text{reach}_{F^*}(t_2^*) \leq 0$, as desired.

The case $\rho(w) = 0$. By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$. Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \text{reach}(t_1)$ and $\mu(F_w) = \text{reach}(t_2)$. Define $F' \vdash w'$ to be the fork obtained by extending the tine t_1 of F_w with parameter $\tilde{a} = 0$ to yield a new tine t'_1 in F' . Then $\text{reach}_{F'}(t'_1) = 0$ and $\text{reach}_{F'}(t_2) = \text{reach}_F(t_2) - 1$. It follows that $\rho(w0) \geq 0$ and $\mu(w0) \geq \mu(w) - 1$. We will show that $\rho(w0) \leq 0$ and that $\mu(w0) \leq \mu(w) - 1$, in which case we can conclude that

$$\rho(w0) = 0 \quad \text{and} \quad \mu(w0) = \mu(w) - 1.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq 0$ and that $\mu(w0) \leq \mu(w) - 1$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let \tilde{a} denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\text{reach}_{F^*}(t^*) = \rho(w0)$. Note that t^* cannot be a tine of F ; if it were then $\text{reach}_{F^*}(t^*) = \text{reach}_F(t^*) - (\tilde{a} + 1) \leq \rho(w) - 1 < 0$ which contradicts $\rho(w0) \geq 0$. Thus t^* was obtained by extension and $\text{reach}_{F^*}(t^*) = 0$. It remains to show that $\mu(w0) \leq 0$. Now let $F^* \vdash w0$ be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\text{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\text{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let \tilde{a} be the parameter for this extension. Similarly, t_1^* cannot be a tine of F ; if it were, $\rho(F^*) = \text{reach}_{F^*}(t_1^*) = \text{reach}_F(t_1^*) - (\tilde{a} + 1) \leq \rho(w) - 1 \leq \rho(w) - 1 < 0$ which contradicts $\rho(F) \geq 0$. It follows that t_1^* must extend a tine t_1 of F for which $\text{reach}_F(t_1) = 0$, because extension can only occur for tines of non-negative reach and $\rho(F) = 0 = \rho(w)$. Thus t_2^* is a tine of F and $t_1 \not\sim t_2^*$ so that $\text{reach}_F(t_2^*) \leq \mu(F) \leq \mu(w)$ and we conclude that $\mu(w0) = \text{reach}_{F^*}(t_2^*) \leq \text{reach}_F(t_2^*) - 1 \leq \mu(w) - 1$, as desired.

The case $\rho(w) > 0, \mu(w) \neq 0$. By induction, there is a fork F_w for which $\mathbf{m}(w) = (\rho(F_w), \mu(F_w))$. Let t_1 and t_2 be edge-disjoint tines of F_w for which $\rho(F_w) = \text{reach}(t_1)$ and $\mu(F_w) = \text{reach}(t_2)$. In fact, any extension of F_w will suffice for the construction; for concreteness, define $F' \vdash w'$ to be the fork obtained by extending the tine t_1 of F_w with parameter $\tilde{a} = 0$. Then $\text{reach}_{F'}(t_i) = \text{reach}_{F_w}(t_i) - 1$. It follows that $\rho(w0) \geq \rho(w) - 1$ and $\mu(w0) \geq \mu(w) - 1$. We will show that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq \mu(w) - 1$, in which case we can conclude that

$$\rho(w0) = \rho(w) - 1 \quad \text{and} \quad \mu(w0) = \mu(w) - 1.$$

Moreover, the fork $F_{w'} = F'$ achieves these statistics, as desired.

We return to establish that $\rho(w0) \leq \rho(w) - 1$ and that $\mu(w0) \leq \mu(w) - 1$. Let $F^* \vdash w0$ be a closed fork for which $\rho(w0) = \rho(F^*)$ and let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$; as above, let \tilde{a} denote the parameter for this extension. Let t^* be an honest tine of F^* so that $\text{reach}_{F^*}(t^*) = \rho(w0)$. Note that if t^* is a tine of F then $\text{reach}_{F^*}(t^*) = \text{reach}_F(t^*) - (\tilde{a} + 1) \leq$

$\rho(w) - 1$; otherwise t^* is obtained by extension and $\text{reach}_{F^*}(t^*) = 0 \leq \rho(w) - 1$, as desired. (Recall that $\rho(w) > 0$.) It remains to show that $\mu(w0) \leq \mu(w) - 1$. Now let $F^* \vdash w0$ be a closed fork for which $\mu(F^*) = \mu(w0)$. Let t_1^* and t_2^* be two edge-disjoint honest tines of F^* so that $\text{reach}_{F^*}(t_1^*) = \rho(F^*)$ and $\text{reach}_{F^*}(t_2^*) = \mu(F^*) = \mu(w0)$. Let $F \vdash w$ be the unique closed fork for which $F \sqsubseteq F^*$ and let \tilde{a} be the parameter for this extension. If both t_1^* and t_2^* are tines of F then $\text{reach}_{F^*}(t_i^*) = \text{reach}_F(t_i^*) - (\tilde{a} + 1)$ and, in particular, $\text{reach}_F(t_1^*) \geq \text{reach}_F(t_2^*)$ so that $\text{reach}_F(t_2^*) \leq \mu(w)$ and $\text{reach}_{F^*}(t_2^*) \leq \mu(w) - 1$, as desired.

To complete the argument, we consider the case that one of the tines t_i^* arises by extension. Note that in this case $\text{reach}_{F^*}(t_2^*) \leq 0$, as either t_2^* is obtained by extension so that it has zero reach, or t_2^* is obtained by extension so that $\text{reach}_{F^*}(t_2^*) \leq \text{reach}_{F^*}(t_1^*) = 0$. Here we further separate the analysis into two cases depending on the sign of $\mu(w)$:

- If $\mu(w) > 0$, then $\text{reach}_{F^*}(t_2^*) \leq 0 \leq \mu(w) - 1$, as desired.
- If $\mu(w) < 0$ then t_2^* cannot be the extension of a tine in F . To see this, suppose to the contrary that t_2^* extends a tine t_2 of F ; then $\text{reach}_F(t_2) \geq 0$. Additionally, t_1^* must be a tine of F , edge-disjoint from t_2 , and $\text{reach}_F(t_1^*) = \text{reach}_{F^*}(t_1^*) + (\tilde{a} + 1) > 0$. It follows that $\mu(w) \geq \mu(F) \geq 0$, a contradiction.

The other possibility is that t_1^* is an extension of a tine t_1 of F in which case $\text{reach}_F(t_1) \geq 0$. Note that t_2^* is a tine of F and edge-disjoint from t_1 ; thus $\min(\text{reach}_F(t_2^*), \text{reach}_F(t_1)) \leq \mu(F) < 0$ and $\text{reach}_F(t_2^*) \leq \mu(F)$. We conclude that $\text{reach}_{F^*}(t_2^*) = \text{reach}_F(t_2^*) - (\tilde{a} + 1) \leq \mu(w) - 1$, as desired. \square

With this recursive description in place, we return to the proof of Theorem 4.13, which we restate below for convenience.

Theorem 4.13, restated. *Let $\epsilon \in (0, 1)$ and let w be a string drawn from $\{0, 1\}^n$ by independently assigning each $w_i = 1$ with probability $(1 - \epsilon)/2$. Then*

$$\Pr[w \text{ is forkable}] = 2^{-\Omega(\sqrt{n})}.$$

Proof of Theorem 4.13. The theorem concerns the probability distribution on $\{0, 1\}^n$ given by independently selecting each $w_i \in \{0, 1\}$ so that

$$\Pr[w_i = 0] = \frac{1 + \epsilon}{2} = 1 - \Pr[w_i = 1].$$

when w is drawn with this distribution. For the string $w_1 \dots w_n$ chosen with the probability distribution above, define the random variables

$$R_t = \rho(w_1 \dots w_t) \quad \text{and} \quad M_t = \mu(w_1 \dots w_t).$$

Our goal is to establish that

$$\Pr[w \text{ forkable}] = \Pr[M_n \geq 0] = 2^{-\Omega(\sqrt{n})}.$$

We extract from the statement of Lemma 4.19 some facts about these random variables.

$$R_t > 0 \implies \begin{cases} R_{t+1} = R_t + 1 & \text{if } w_{t+1} = 1, \\ R_{t+1} = R_t - 1 & \text{if } w_{t+1} = 0; \end{cases} \quad (3)$$

$$M_t < 0 \implies \begin{cases} M_{t+1} = M_t + 1 & \text{if } w_{t+1} = 1, \\ M_{t+1} = M_t - 1 & \text{if } w_{t+1} = 0; \end{cases} \quad (4)$$

$$R_t = 0 \implies \begin{cases} R_{t+1} = 1 & \text{if } w_{t+1} = 1, \\ R_{t+1} = 0 & \text{if } w_{t+1} = 0, \\ M_{t+1} < 0 & \text{if } w_t = 0. \end{cases} \quad (5)$$

In light of the properties (3) above, the random variables R_t are quite well-behaved when positive—in particular, considering the distribution placed on each w_i , they simply follow the familiar biased random walk of Figure 6. Likewise, considering the properties (4), the random variables M_t follow a biased random walk when negative. The remainder of the proof combines these probability laws with (5) and the fact that $M_t() \leq R_t()$ to establish that $M_n < 0$ with high probability.

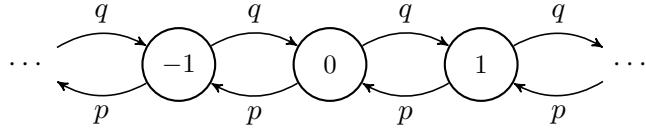


Figure 6: The simple biased walk where $p = (1 + \epsilon)/2$ and $q = 1 - p$.

We recall two basic facts about the standard biased walk associated with the Markov chain of Figure 6. Let $Z_i \in \{\pm 1\}$ (for $i = 1, 2, \dots$) denote a family of independent random variables for which $\Pr[Z_i = 1] = (1 - \epsilon)/2$. Then the biased walk given by the variables $Y_t = \sum_i^t Z_i$ has the following properties.

Constant escape probability; gambler's ruin. With constant probability, depending only on ϵ , $Y_t \neq 1$ for all $t > 0$. In general, for each $k > 0$,

$$\Pr[\exists t, Y_t = k] = \alpha^k, \quad (6)$$

for a constant $\alpha < 1$ depending only on ϵ . (In fact, the constant α is $(1 - \epsilon)/(1 + \epsilon)$; see, e.g., [25, Chapter 12] for a complete development.)

Concentration (the Chernoff bound). Consider T steps of the biased walk beginning at state 0; then the resulting value is tightly concentrated around $-\epsilon T$. Specifically, $\mathbb{E}[Y_T] = -\epsilon T$ and

$$\Pr\left[Y_T > -\frac{\epsilon T}{2}\right] = 2^{-\Omega(T)}. \quad (7)$$

(The constant hidden in the $\Omega()$ notation depends only on ϵ . See, e.g., [1, Cor. A.1.14].)

Partitioning the string w , we write $w = w^{(1)} \dots w^{(\sqrt{n})}$ where $w^{(t)} = w_{1+a_{t-1}} \dots w_{a_t}$ and $a_t = \lceil t\sqrt{n} \rceil$, for $t = 0, 1, \dots$. Let $R_{(0)} = 0$ and $R_{(t)} = R_{a_t}$; similarly define $M_{(0)} = 0$ and $M_{(t)} = M_{a_t}$. Fix $\delta \ll \epsilon$ to be a small constant.

We define three events based on the random variables $R_{(t)}$ and $M_{(t)}$:

Hot We let Hot_t denote the event that $R_{(t)} \geq \delta\sqrt{n}$ and $M_{(t)} \geq -\delta\sqrt{n}$.

Volatile We let Vol_t denote the event that $-\delta\sqrt{n} \leq M_{(t)} \leq R_{(t)} < \delta\sqrt{n}$.

Cold We let Cold_t denote the event that $M_{(t)} < -\delta\sqrt{n}$.

Note that for each t , exactly one of these events occurs—they partition the probability space. Then we will establish that

$$\Pr[\text{Cold}_{t+1} \mid \text{Cold}_t] \geq 1 - 2^{-\Omega(\sqrt{n})}, \quad (8)$$

$$\Pr[\text{Cold}_{t+1} \mid \text{Vol}_t] \geq \Omega(\epsilon), \quad (9)$$

$$\Pr[\text{Hot}_{t+1} \mid \text{Vol}_t] \leq 2^{-\Omega(\sqrt{n})}. \quad (10)$$

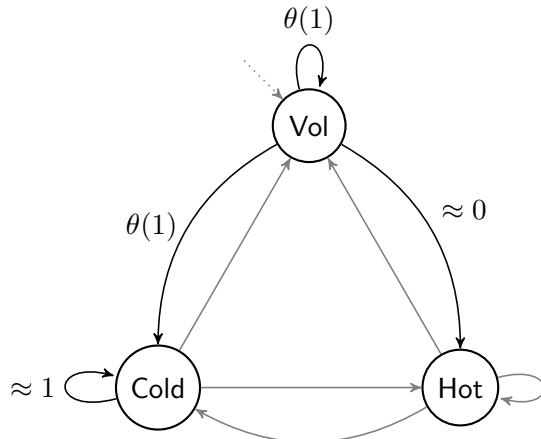


Figure 7: An illustration of the transitions between Cold, Vol, and Hot.

Note that the event Vol_0 occurs by definition. Assuming these inequalities, we observe that the system is very likely to eventually become cold, and stay that way. In this case, $\text{Cold}_{\sqrt{n}}$ occurs, $M_n^o < \delta\sqrt{n} < 0$, and w is not forkable. Specifically, note that the probability that the system ever transitions from volatile to hot is no more than $2^{-\Omega(\sqrt{n})}$ (as transition from Vol to Hot is bounded above by $2^{-\Omega(\sqrt{n})}$, and there are no more than \sqrt{n} possible transition opportunities). Note, also, that while the system is volatile, it transitions to cold with constant probability during each period. In particular, the probability that the system is volatile for the entire process is no more than $2^{-\Omega(\sqrt{n})}$. Finally, note that the probability that the system ever transitions out of the cold state is no more than $2^{-\Omega(\sqrt{n})}$ (again, there are at most \sqrt{n} possible times when this could happen, and any individual transition occurs with probability $2^{-\Omega(\sqrt{n})}$). It follows that the system is cold at the end of the process with probability $1 - 2^{-\Omega(\sqrt{n})}$.

It remains to establish the three inequalities (8), (9), and (10).

Inequality (8): This follows directly from (3) and (6). Specifically, in light of (4) the random variables M_i follow the probability law of the simple biased walk when they are negative. Conditioned on $M_{(t)} = M_{at} < -\delta\sqrt{n}$, the probability that any future M_s ever climbs to the value -1 is no more than $\alpha^{-\delta\sqrt{n}} = 2^{-\Omega(n)}$, as desired. (Here $\alpha < 1$ is a fixed constant that depends only on ϵ .)

Inequality (9): This follows from (3), (5), (6), and (7). Specifically, conditioned on Vol_t , $R_{(t)} \leq \delta\sqrt{n}$. Recall from (3) that the random variables R_i follow the probability law of the simple biased walk when they are positive. Let D be the event that $R_i > 0$ for all $a_t \leq i < a_t + 2\delta\sqrt{n}$. According to (7), then, where we take $T = 2\delta\sqrt{n}$, $\Pr[D] \leq 2^{-\Omega(\sqrt{n})}$. With near certainty, then, the random variables R_i visit the value 0 during this period. Observe that if $R_i = 0$ then, by (5), $M_{i+1} \leq -1$ with constant probability and (conditioned on this), by (6), with constant probability the subsequent random variables M_j do not return to the value 0. Additionally, in light of (7), the probability that there is a sequence $w_i \dots w_j$ of length at least $2(\delta/\epsilon)\sqrt{n}$ for which

$$\left(\sum_{k=i}^j \begin{cases} 1 & \text{if } w_k = 1, \\ -1 & \text{if } w_k = 0. \end{cases} \right) \geq -\delta\sqrt{n}$$

is no more than $(\sqrt{n})^2 2^{-\Omega(\sqrt{n})}$. It follows that with constant probability, the walk (of R_i) hits 0, as described above, and then M_i terminates at a value less than $-\delta\sqrt{n}$.

Inequality (10): This follows from (3), (5), (6), and (7). Specifically, conditioned on Vol_t , $R_{(t)} \leq \delta\sqrt{n}$. Recall from (3) that the random variables R_i follow the probability law of the simple biased walk when they are positive. Let D be the event that $R_i > 0$ for all $a_t \leq i < a_t + 2\delta\sqrt{n}$. According to (7), then, where we take $T = 2\delta\sqrt{n}$, $\Pr[D] \leq 2^{-\Omega(\sqrt{n})}$. With near certainty, then, the random variables R_i visit the value 0 during this period. Conditioned on \bar{D} , in order for $R_{a_{t+1}} \geq \delta\sqrt{n}$ there must be a sequence of these random variables $0 = R_i, R_{i+1}, \dots, R_j = \lfloor \delta\sqrt{n} \rfloor$ so that none of these take the value 0 except the first. (Such a sequence arises by taking i to be the last time the variables R_{a_t}, \dots visit 0 and j the first subsequent time that the sequence is larger than $\delta\sqrt{n}$.) In light of (6), the probability of such a subsequence appearing at a particular value for i is no more than $\alpha^{-\delta\sqrt{n}}$. It follows that the probability that $R_{a_{t+1}} \geq \delta\sqrt{n}$ is less than $\sqrt{n}\alpha^{-\delta\sqrt{n}} = 2^{-\Omega(\sqrt{n})}$, as desired. \square

Exact probabilities of forkability for explicit values of n . In order to gain further insight regarding the density of forkable strings, we exactly computed the probability that a string w drawn from the binomial distribution with parameter $p \in \{.40, .41, \dots, .50\}$ is forkable for several different lengths. These results are presented in Figure 8.

4.3.1 Covert adversaries, covert forks, and covertly forkable strings

The general notion of fork defined in Definition 4.10 above reflects the possibility that adversarial slot leaders may broadcast multiple blocks for a single slot; such adversaries may simultaneously extend many different chains. While this provides the adversary significant opportunities to interfere with the protocol, it leaves a suspicious “audit trail”—multiple signed blocks for the same slot—which conspicuously deviates from the protocol.

This motivates our consideration of a restricted class of *covert* adversaries, who broadcast no more than one block per slot. Such an adversary may still deviate from the protocol by extending short chains, but does not produce such suspicious evidence and hence its strategy is more “deniable”: it can blame network delays for its actions.⁶

Such an adversary yields a restricted notion of fork, defined below:

⁶Contrast this with a more general adversary that attempts to fork by signing two different blocks for the same slot; such an adversary cannot merely blame the network for such a deviation.

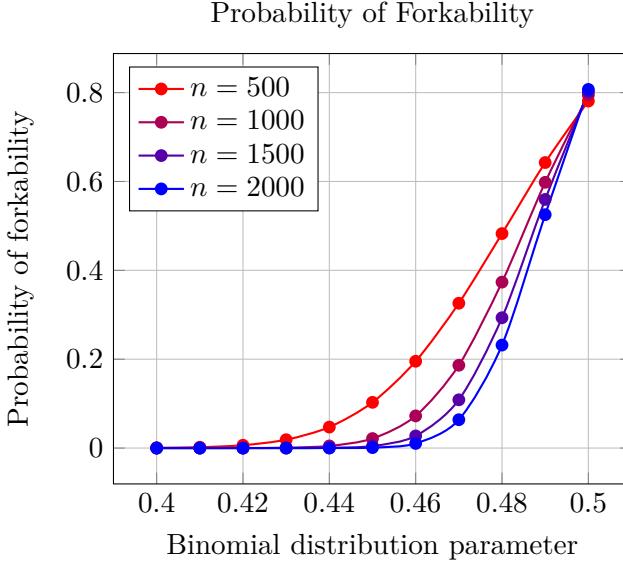


Figure 8: Graphs of the probability that a string drawn from the binomial distribution is forkable. Graphs for string lengths $n = 500, 1000, 1500, 2000$ are shown with parameters .40, .41, .42, .43, .44, .45, .46, .47, .48, .49, .50.

Definition 4.20. Let $F \vdash w$ be a fork for a string $w \in \{0, 1\}^*$. We say that F is covert if the labeling $\ell : V \rightarrow \{0, 1, \dots\}$ is injective. In particular, no adversarial index is labeled by more than one node.

As in the general case, we define a notion of forkable string for such adversaries.

Definition 4.21. We say that a string w is covertly forkable if there is a flat covert fork $F \vdash w$.

Covert adversaries and forks have much simpler structure than general adversaries. In particular, a string is covertly forkable if and only if a majority of its indices are adversarial. This provides an analogue of Proposition 4.18 for covertly forkable strings.

Proposition 4.22. A string $w \in \{0, 1\}^n$ is covertly forkable if and only if $\text{wt}(w) \geq n/2$.

Proof. Let w be a covertly forkable string and $F \vdash w$ a flat covert fork. As F is flat, there are two edge disjoint tines, t_1 and t_2 , with length equal to $\text{height}(F)$ and it follows that the number of vertices in F is at least $2 \cdot \text{height}(F) + 1$. In this covert case the labeling function is injective, and it follows that $n \geq 2 \cdot \text{height}(F)$. (Recall that the root vertex is labeled by 0, which is not an index into w .) On the other hand, the height of F is at least the number of honest indices of w . We conclude that the length of w is at least twice the number of honest indices, as desired.

If $\text{wt}(w) \geq n/2$, we can produce a flat covert fork $F \vdash w$ by placing all honest indices on a common tine t_1 and selecting $\text{length}(t_1)$ adversarial indices to form an edge-disjoint second tine t_2 . \square

As the structure of covertly forkable strings is so simple, an analogue of Theorem 4.13 for the density of covertly forkable strings follows directly from standard large deviation bounds.

Theorem 4.23. Let $\epsilon \in (0, 1)$ and let w be a string drawn from $\{0, 1\}^n$ by independently assigning each $w_i = 1$ with probability $(1 - \epsilon)/2$. Then

$$\Pr[w \text{ is covertly forkable}] = 2^{-\Theta(n)}.$$

Proof. This follows from standard estimates for the cumulative density function of the binomial distribution. \square

Exact probabilities of covert forkability for explicit values of n . For comparison with the general case, we computed the probability that a string drawn from the binomial distribution is covertly forkable. These results are presented in Figure 9. (Note that these probabilities are simply appropriate evaluations of the cumulative density function of the binomial distribution.) Analogous results for the general case appeared in Figure 8.

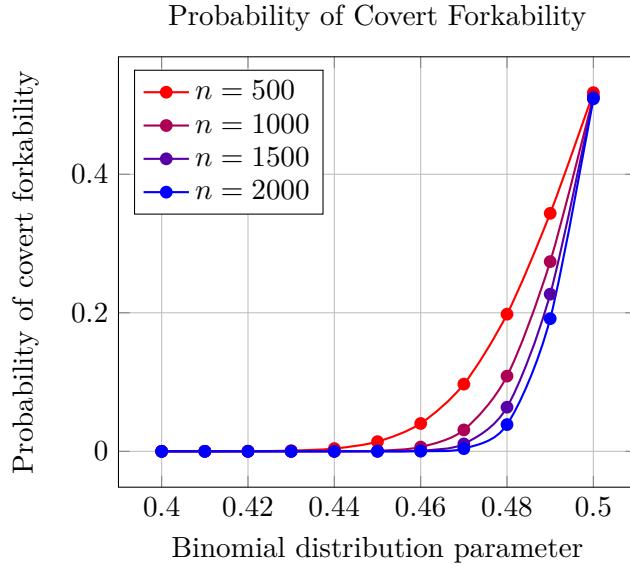


Figure 9: Graphs of the probability that a string drawn from the binomial distribution is covertly forkable. Graphs for string lengths $n = 500, 1000, 1500, 2000$ are shown with parameters $.40, .41, \dots, .49, .50$.

4.4 Common Prefix

Recall that the chains constructed by honest players during an execution of π_{iSPoS} correspond to tines of a fork, as defined and studied in the previous sections. The random assignment of slots to stakeholders given by $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \text{F}}$ guarantees that the coordinates of the associated characteristic string w follow the binomial distribution with probability equal to the adversarial stake. Thus Theorem 4.13 establishes that no execution of the protocol π_{iSPoS} can induce two tines (chains) of maximal length with no common prefix.

In the context of π_{iSPoS} , however, we wish to establish a much stronger *common prefix* property: any pair of chains which could, in principle, be presented by the adversary to an honest party must have a “recent” common prefix, in the sense that removing a small number of blocks from the shorter chain results in a prefix of the longer chain.

To formally articulate and prove this property, we introduce some further definitions regarding tines and forks. We borrow the “truncation operator”, described earlier in the paper for chains: for a tine t we let $t^{[k]}$ denote the tine obtained by removing the last k edges; if $\text{length}(t) \leq k$, we define $t^{[k]}$ to consist solely of the root.

Definition 4.24 (Viability). Let $F \vdash w$ be a fork for a string $w \in \{0, 1\}^n$ and let t be a tine of F . We say that t is viable if, for all honest indices $h \leq \ell(t)$, we have

$$\mathbf{d}(h) \leq \text{length}(t).$$

(Recall that $\ell(t)$ is the label of the terminal vertex of t .)

If t is viable, an external (honest) observer witnessing the execution at time $\ell(t)$ —if provided the tine t along with all honest tines generated up to time $\ell(t)$ —could conceivably select t via the `maxvalid()` rule. Observe that any honest tine is viable: by definition, the depth of the terminal vertex of an honest tine exceeds that of all prior honest vertices.

Definition 4.25 (Divergence). Let F be a fork for a string $w \in \{0, 1\}^*$. For two viable tines t_1 and t_2 of F , define their divergence to be the quantity

$$\text{div}(t_1, t_2) = \min_i (\text{length}(t_i) - \text{length}(t_1 \cap t_2)),$$

where $t_1 \cap t_2$ denotes the common prefix of t_1 and t_2 . We overload this notation by defining divergence for F as the maximum over all pairs of viable tines:

$$\text{div}(F) = \max_{\substack{t_1, t_2 \text{ viable} \\ \text{tines of } F}} \text{div}(t_1, t_2).$$

Finally, define the divergence of w to be the maximum such divergence over all all possible forks for w :

$$\text{div}(w) = \max_{F \vdash w} \text{div}(F).$$

Observe that if $\text{div}(t_1, t_2) \leq k$ and, say, $\text{length}(t_1) \leq \text{length}(t_2)$, the tine $t_1^{[k]}$ is a prefix of t_2 .

We first establish that a string with large divergence must have a large forkable substring. We then apply this in Theorem 4.27 below to conclude that characteristic strings arising from π_{iSPoS} are unlikely to have large divergence and, hence, possess the common prefix property.

Theorem 4.26. Let $w \in \{0, 1\}^*$. Then there is forkable substring \check{w} of w with $|\check{w}| \geq \text{div}(w)$.

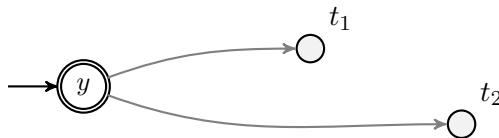
Proof. Consider a fork $F \vdash w$ and a pair of viable tines (t_1, t_2) for which

$$\text{div}(t_1, t_2) = \text{div}(w). \tag{11}$$

For simplicity, we assume the tines have been labeled so that $\ell(t_1) < \ell(t_2)$ and further that

$$|\ell(t_2) - \ell(t_1)| \text{ is minimum among all pairs of tines for which (11) holds.} \tag{12}$$

We begin by identifying the substring \check{w} ; the remainder of the proof is devoted to constructing a flat fork for \check{w} to establish forkability. Let y denote the last vertex on the tine $t_1 \cap t_2$, as in the diagram below, and let $\alpha \triangleq \ell(y) = \ell(t_1 \cap t_2)$.



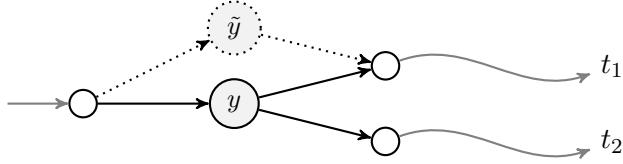
Let β denote the smallest honest index of w for which $\beta \geq \ell(t_2)$, with the convention that if there is no such index we define $\beta = n + 1$. Observe that, in any case, $\ell(t_1) < \ell(t_2)$ and hence that $\beta - 1 \geq \ell(t_1)$. These indices, α and β , distinguish the substring $\check{w} = w_{\alpha+1} \dots w_{\beta-1}$, which will be the subject of the remainder of the proof. As the function $\ell(\cdot)$ is strictly increasing along any tine, observe that

$$\begin{aligned} |\check{w}| &= \beta - \alpha - 1 \geq \ell(t_1) - \ell(y) \geq \text{length}(t_1) - \text{length}(t_1 \cap t_2) \\ &\geq \min(\text{length}(t_1), \text{length}(t_2)) - \text{length}(t_1 \cap t_2) = \text{div}(w), \end{aligned}$$

so \check{w} has the desired length and it suffices to establish that it is forkable.

We briefly summarize the proof before presenting the details. We begin by establishing several structural properties of the tines t_1 and t_2 that follow from the assumptions (11) and (12) above. To establish that \check{w} is forkable we then extract from F a flat fork (for \check{w}) in two steps: (i.) the fork F is subjected to some minor restructuring to ensure that all “long” tines pass through y ; (ii.) a flat fork is constructed by treating the vertex y as the root of a portion of the subtree of F labeled with indices of \check{w} . At the conclusion of the construction, segments of the two tines t_1 and t_2 will yield the required “long, disjoint” tines satisfying the definition of forkable.

We observe, first of all, that the vertex y cannot be adversarial: otherwise it is easy to construct an alternative fork $\tilde{F} \vdash w$ and a pair of tines in \tilde{F} that achieve larger divergence. Specifically, construct \tilde{F} from F by adding a new (adversarial) vertex \tilde{y} to F for which $\ell(\tilde{y}) = \ell(y)$, adding an edge to \tilde{y} from the vertex preceding y , and replacing the edge of t_1 following y with one from \tilde{y} ; then the other relevant properties of the fork are maintained, but the divergence of the resulting tines has increased by one. (See the diagram below.)



A similar argument implies that the fork $F_0 \vdash w_1 \dots w_\alpha$ obtained by including only those vertices of F with labels less than or equal to $\alpha = \ell(y)$ has a unique vertex of depth $\text{depth}(y)$ (namely, y itself). In the presence of another vertex \tilde{y} (of F_0) with depth $\text{depth}(y)$, “redirecting” t_1 through \tilde{y} (as in the argument above) would likewise result in a fork with larger divergence. Note that $\ell(\cdot)$ would indeed be increasing along this new tine (resulting from redirecting t_1) because $\ell(\tilde{y}) \leq \ell(y)$ according to the definition of F_0 . As α is the last index of the string, this additionally implies that F_0 has no vertices of depth exceeding $\text{depth}(y)$.

We remark that the minimality assumption (12) implies that any honest index h for which $h < \beta$ has depth no more than $\min(\text{length}(t_1), \text{length}(t_2))$: specifically,

$$h < \beta \implies \mathbf{d}(h) \leq \min(\text{length}(t_1), \text{length}(t_2)). \quad (13)$$

To see this, consider an honest index $h < \beta$ and a tine t_h for which $\ell(t_h) = h$. Recall that t_1 and t_2 are viable; as $h < \ell(t_2)$ it follows immediately that $\mathbf{d}(h) \leq \text{length}(t_2)$. Similarly, if $h \leq \ell(t_1)$ then $\mathbf{d}(h) \leq \text{length}(t_1)$, so it remains to settle the case when $\ell(t_1) < h < \ell(t_2)$: in particular, in this regime we wish to likewise guarantee that $\mathbf{d}(h) \leq \text{length}(t_1)$. For the sake of contradiction, assume that $\text{length}(t_h) = \mathbf{d}(h) > \text{length}(t_1)$. Considering the tine t_h , we separately investigate two cases depending on whether t_h shares an edge with t_1 after the vertex y . If, indeed, t_h and t_1 share an edge after the vertex y then t_h and t_2 do not share such an edge, and we observe that $\text{div}(t_h, t_2) \geq \text{div}(t_1, t_2)$ while $|\ell(t_2) - h| < |\ell(t_2) - \ell(t_1)|$ which contradicts (12). If, on the other

hand, t_h shares no edge with t_1 after y , we similarly observe that $\text{div}(t_1, t_h) \geq \text{div}(t_1, t_2)$ while $|t_h - \ell(t_1)| < |\ell(t_2) - \ell(t_1)|$, which contradicts (12).

In light of the remarks above, we observe that the fork F may be “pinched” at y to yield an essentially identical fork $F^{>y\triangleleft} \vdash w$ with the exception that all tines of length exceeding $\text{depth}(y)$ pass through the vertex y . Specifically, the fork $F^{>y\triangleleft} \vdash w$ is defined to be the graph obtained from F by changing every edge of F directed towards a vertex of depth $\text{depth}(y) + 1$ so that it originates from y . To see that the resulting tree is a well-defined fork, it suffices to check that $\ell(\cdot)$ is still increasing along all tines of $F^{>y\triangleleft}$. For this purpose, consider the effect of this pinching on an individual tine t terminating at a particular vertex v —it is replaced with a tine $t^{>y\triangleleft}$ defined so that:

- If $\text{length}(t) \leq \text{depth}(y)$, the tine t is unchanged: $t^{>y\triangleleft} = t$.
- Otherwise, $\text{length}(t) > \text{depth}(y)$ and t has a vertex z of depth $\text{depth}(y) + 1$; note that $\ell(z) > \ell(y)$ because F_0 contains no vertices of depth exceeding $\text{depth}(y)$. Then $t^{>y\triangleleft}$ is defined to be the path given by the tine terminating at y , a (new) edge from y to z , and the suffix of t beginning at z . (As $\ell(z) > \ell(y)$ this has the increasing label property.)

Thus the tree $F^{>y\triangleleft}$ is a legal fork on the same vertex set; note that depths of vertices in F and $F^{>y\triangleleft}$ are identical.

By excising the tree rooted at y from this pinched fork $F^{>y\triangleleft}$ we may extract a fork for the string $w_{\alpha+1} \dots w_n$. Specifically, consider the induced subgraph $F^{y\triangleleft}$ of $F^{>y\triangleleft}$ given by the vertices $\{y\} \cup \{z \mid \text{depth}(z) > \text{depth}(y)\}$. By treating y as a root vertex and suitably defining the labels $\ell^{y\triangleleft}$ of $F^{y\triangleleft}$ so that $\ell^{y\triangleleft}(z) = \ell(z) - \ell(y)$, this subgraph has the defining properties of a fork for $w_{\alpha+1} \dots w_n$. In particular, considering that α is honest it follows that each honest index $h > \alpha$ has depth $\mathbf{d}(h) > \text{length}(y)$ and hence labels a vertex in $F^{y\triangleleft}$. For a tine t of $F^{>y\triangleleft}$, we let $t^{y\triangleleft}$ denote the suffix of this tine beginning at y , which forms a tine in $F^{y\triangleleft}$. (If $\text{length}(t) \leq \text{depth}(y)$, we define $t^{y\triangleleft}$ to consist solely of the vertex y .) Note that $t_1^{y\triangleleft}$ and $t_2^{y\triangleleft}$ share no edges in the fork $F^{y\triangleleft}$.

Finally, let \check{F} denote the tree obtained from $F^{y\triangleleft}$ as the union of all tines t of $F^{y\triangleleft}$ so that all labels of t are drawn from \check{w} (as it appears as a prefix of $w_{\alpha+1} \dots w_n$), and

$$\text{length}(t) \leq \max_{\substack{h \leq |\check{w}| \\ h \text{ honest}}} \mathbf{d}(h).$$

It is immediate that $\check{F} \vdash \check{w}$. To conclude the proof, we show that \check{F} is flat. For this purpose, we consider the tines $t_1^{y\triangleleft}$ and $t_2^{y\triangleleft}$. As mentioned above, they share no edges in $F^{y\triangleleft}$, and hence the prefixes \check{t}_1 and \check{t}_2 (of $t_1^{y\triangleleft}$ and $t_2^{y\triangleleft}$) appearing in \check{F} share no edges. We wish to see that these prefixes have maximum length in \check{F} , in which case \check{F} is flat, as desired. This is immediate for the tine \check{t}_1 because all labels of $t_1^{y\triangleleft}$ are drawn from \check{w} and, considering (13), its depth is at least that of all relevant honest vertices. As for \check{t}_2 , observe that if $\ell(t_2)$ is not honest then $\beta > \ell(t_2)$ so that, as with \check{t}_1 , the tine \check{t}_2 is labeled by \check{w} so that the same argument, relying on (13), ensures that \check{t}_2 has length at least that of all relevant honest vertices. If $\ell(t_2)$ is honest, $\beta = \ell(t_2)$, and the terminal vertex of $t_2^{y\triangleleft}$ does not appear in \check{F} (as it does not index \check{w}). In this case, however, $\text{length}(t_2^{y\triangleleft}) > \mathbf{d}(h)$ for any honest index of \check{w} , and it follows that $\text{length}(\check{t}_2) = \text{length}(t_2^{y\triangleleft}) - 1$ is at least the depth of any honest index of \check{w} , as desired. \square

Theorem 4.27. *Let $k, R \in \mathbb{N}$ and $\epsilon \in (0, 1)$. The probability that the π_{iSPos} protocol, when executed with a $(1 - \epsilon)/2$ fraction of adversarial stake, violates the common prefix property with parameter k throughout an epoch of R slots is no more than $\exp(-\Omega(\sqrt{k}) + \ln R)$; the constant hidden by the $\Omega()$ notation depends only on ϵ .*

sketch. Observe that an execution of π_{iSPoS} violates the common prefix property with parameters k, R precisely when the fork F induced by this execution has $\text{div}(F) \geq k$. Thus we wish to show that the probability that $\text{div}(w) \geq k$ is no more than $\exp(-\Omega(\sqrt{k}) + \log R)$. Let Bad denote the event that $\text{div}(w) \geq k$.

It follows from Theorem 4.26 that if $\text{div}(w) \geq k$, there is a forkable substring \check{w} of length at least k .

Thus

$$\begin{aligned} \Pr[\text{common prefix violation}] &\leq \Pr \left[\begin{array}{l} \exists \alpha, \beta \in \{1, \dots, R\} \text{ so that } \alpha + k - 1 \leq \beta \text{ and} \\ w_\alpha \dots w_\beta \text{ is forkable} \end{array} \right] \\ &\leq \underbrace{\sum_{1 \leq \alpha \leq R} \sum_{\alpha+k-1 \leq \beta \leq R} \Pr[w_\alpha \dots w_\beta \text{ is forkable}]}_{(*)}. \end{aligned}$$

Recall that the characteristic string $w \in \{0, 1\}^R$ for such an execution of π_{iSPoS} is determined by assigning each $w_i = 1$ independently with probability $(1 - \epsilon)/2$. According to Theorem 4.13 the probability that a string of length t drawn from this distribution is forkable is no more than $\exp(-c\sqrt{t})$ for a positive constant c . Note that for any $\alpha \geq 1$,

$$\sum_{t=\alpha+k-1}^R e^{-c\sqrt{t}} \leq \int_{k-1}^{\infty} e^{-c\sqrt{t}} dt = (2/c^2)(1 + c\sqrt{k-1})e^{-c\sqrt{k-1}} = e^{-\Omega(\sqrt{k})}$$

and it follows that the sum $(*)$ above is $\exp(-\Omega(\sqrt{k}))$. Thus

$$\Pr[\text{common prefix violation}] \leq R \cdot \exp(-\Omega(\sqrt{k})) \leq \exp(\ln R - \Omega(\sqrt{k})),$$

as desired. \square

4.4.1 Common prefix with covert adversaries

We revisit the notion of common prefix in the setting of covert adversaries. We define the *covert divergence* of w to be the maximum divergence over all possible covert forks for w :

$$\text{cdiv}(w) = \max_{\substack{F \vdash w \\ F \text{ covert}}} \text{div}(F).$$

As in the setting with general adversaries, we wish to establish that a string with large covert divergence must have a large covertly forkable substring. A direct analogue of Theorem 4.27 then implies that characteristic strings arising from π_{iSPoS} are unlikely to have large covert divergence and, hence, possess the common prefix property against covert adversaries.

We record an analogue of Theorem 4.26 for covert adversaries.

Theorem 4.28. *Let $w \in \{0, 1\}^*$. Then there is a covertly forkable substring \check{w} of w with $|\check{w}| \geq \text{cdiv}(w)$.*

Proof. We are more brief, as portions of the proof have direct analogs in the proof of Theorem 4.26. Consider a covert fork $F \vdash w$ and a pair of viable tines (t_1, t_2) of F for which $\text{div}(t_1, t_2) = \text{cdiv}(w)$; we assume the tines are identified so that $\ell(t_1) < \ell(t_2)$ and, as in the proof of the general case, assume that this pair of tines minimizes the quantity $|\ell(t_2) - \ell(t_1)|$ among all pairs with divergence equal to $\text{cdiv}(w)$.

Let y denote the last vertex on the tine $t_1 \cap t_2$. In contrast to the setting with a general adversary, it is not clear that y is honest and this motivates a slightly different choice for the beginning of the string \check{w} : define α to be the largest honest index of w on the tine $t_1 \cap t_2$, with the convention that $\alpha = 0$ if there is no such index. As in the proof of Theorem 4.26, define β to be the smallest honest index of w for which $\beta \geq \ell(t_2)$, with the convention that $\beta = n + 1$ if there is no such honest index. Then define $\check{w} = w_{\alpha+1} \dots w_{\beta-1}$; as in the proof of Theorem 4.26 it is easy to confirm that $|\check{w}| = (\beta - 1) - \alpha \geq \ell(t_1) - \ell(t_1 \cap t_2) \geq \text{cdiv}(w)$. The remainder of the proof argues that \check{w} is covertly forkable.

As in the proof of Theorem 4.26, the depth $\mathbf{d}(h)$ of any honest index $h < \beta$ is no more than $\min(\text{length}(t_1), \text{length}(t_2))$: if $h \leq \ell(t_1)$ this follows directly from the definition of viability. Otherwise, $\ell(t_1) < h < \ell(t_2)$ and we consider the tine t_h labeled with h : if $\text{length}(t_h) \geq \min(\text{length}(t_1), \text{length}(t_2))$ then the tine t_h , coupled with either t_1 or t_2 , would produce a pair of tines with divergence no less than $\text{div}(t_1, t_2)$, but for which $|\ell(\cdot) - \ell(\cdot)|$ is strictly less than $|\ell(t_1) - \ell(t_2)|$.

To complete the proof, we define an injective function $i : H \rightarrow A$, where H denotes the set of honest indices in $\{\alpha + 1, \dots, \beta - 1\}$ and A the complement—the set of adversarial indices of \check{w} . The existence of such a function implies that $|H| \leq |A|$ and hence that \check{w} is covertly forkable by the criterion given in Proposition 4.22. Let $A' \subset A$ denote the set of adversarial indices of \check{w} appearing as a label on either of the two tines t_1 and t_2 . The function i is defined as follows: $i(h)$, for an honest index $h \in H$, is the smallest (adversarial) index of A' which labels a vertex at depth equal to $\mathbf{d}(h)$. Assuming that this function is well-defined it is clearly injective, as labels cannot appear on multiple vertices of a covert fork and depths of honest vertices are pairwise distinct.

To confirm that $i(h)$ is well-defined, note that for any $h \in H$ we must have $\mathbf{d}(\alpha) < \mathbf{d}(h) \leq \min(\text{length}(t_1), \text{length}(t_2))$ and hence there is at least one vertex v on each of t_1 and t_2 with depth equal to $\mathbf{d}(h)$; furthermore, by the defining properties of α and β , this vertex is labeled with an index of \check{w} . If $\mathbf{d}(h) \leq \text{length}(t_1 \cap t_2)$, there is a common vertex v on these tines for which $\text{length}(v) = \mathbf{d}(h)$; note that this vertex cannot be honest by the definition of α , so $i(h) = \ell(v)$ is well-defined in this case. If $\mathbf{d}(h) > \text{length}(t_1 \cap t_2)$, the two tines have distinct vertices at depth $\mathbf{d}(h)$, and one of these must then be adversarial—thus $i(h)$ is well-defined in this case as well. \square

Finally, we remark that the proof of Theorem 4.27 applies with minor adaptations to the covert case.

Theorem 4.29. *Let $k, R \in \mathbb{N}$ and $\epsilon \in (0, 1)$. The probability that the π_{iSPoS} protocol, when executed with a $(1 - \epsilon)/2$ fraction of adversarial stake and a covert adversary, violates the common prefix property with parameter k throughout a period of R is no more than $\exp(-\Omega(k) + \ln R)$; the constant hidden by the $\Omega()$ notation depends only on ϵ .*

Proof. The proof of Theorem 4.27 applies directly; in this case the asymptotics rely on Theorem 4.23 and the following bound applied in a way that the constant c depends only on ϵ .

$$\sum_{t=k}^{\infty} e^{-ct} \leq \int_{k-1}^{\infty} e^{-ct} dt = e^{-\Theta(k)}. \quad \square$$

4.5 Chain Growth and Chain Quality

Anticipating these two proofs, we record an additive Chernoff–Hoeffding bound. (See, e.g., [29] for a proof.)

Theorem 4.30 (Chernoff–Hoeffding bound). *Let X_1, \dots, X_T be independent random variables with $\mathbb{E}[X_i] = p_i$ and $X_i \in [0, 1]$. Let $X = \sum_{i=1}^T X_i$ and $\mu = \sum_{i=1}^T p_i = \mathbb{E}[X]$. Then, for all $\delta \geq 0$,*

$$\Pr[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \quad \text{and} \quad \Pr[X \leq (1 - \delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}.$$

We will start with the chain growth property.

Theorem 4.31. *The π_{iSPoS} protocol satisfies the chain growth property with parameters $\tau = 1 - \alpha, s \in \mathbb{N}$ throughout an epoch of R slots with probability at least $1 - \exp(-\Omega(\epsilon^2 s) + \ln R)$ against an adversary holding an $\alpha - \epsilon$ portion of the total stake.*

Proof. Define $\text{Ham}_a(\alpha)$ to be the event that the Hamming weight ratio of the characteristic string that corresponds to the slots $[a, a + s - 1]$ is no more than α . Given that the adversarial stake is $\alpha - \epsilon$, each of the k slots has probability $\alpha - \epsilon$ being assigned to the adversary and thus the probability that the Hamming weight is more than αs drops exponentially in s . Specifically, using the additive version of the Chernoff bound, we have that $\Pr[\neg \text{Ham}_a(\alpha)] \leq \exp(-2\epsilon^2 s)$. It follows that,

$$\Pr[\text{Ham}_\alpha] \geq 1 - \exp(-2\epsilon^2 s).$$

Given the above we know that when Ham_α happens there will be at least $(1 - \alpha)s$ honest slots in the period of s rounds. Given that each honest slot enables an honest party to produce a block, all honest parties will advance by at least that many blocks. Using a union bound, it follows that the speed coefficient can be set to $\tau = (1 - \alpha)$ and it is satisfied with probability at least $1 - \exp(-2\epsilon^2 s + \ln(R))$. \square

Having established chain growth we now turn our attention to chain quality. Recall that the chain quality property with parameters μ and ℓ asserts that among every ℓ consecutive blocks in a chain (possessed by an honest user), the fraction of adversarial blocks is no more than μ .

Theorem 4.32. *Let $\alpha - \epsilon$ be the adversarial stake ratio. The π_{iSPoS} protocol satisfies the chain quality property with parameters $\mu(\alpha - \epsilon) = \alpha/(1 - \alpha)$ and $\ell \in \mathbb{N}$ throughout an epoch of R slots with probability at least*

$$1 - \exp(-\Omega(\epsilon^2 \alpha \ell) + \ln R).$$

Proof. First, from the proof for chain growth (Theorem 4.31), we know that with high probability a segment of ℓ rounds will involve at least $(1 - \alpha)\ell$ slots with honest leaders; hence the resulting chain must advance by at least $(1 - \alpha)\ell$ blocks. By similar reasoning, the adversarial parties are associated with no more than $\alpha\ell$ slots, and thus can contribute no more than $\alpha\ell$ blocks to any particular chain over this period. It follows that the associated chain possessed by any honest party contains a fraction $\alpha/(1 - \alpha)$ of adversarial blocks with probability $1 - \exp(-\Omega(\epsilon^2 \min(\alpha, 1 - \alpha)\ell) + \ln R)$. \square

5 Our Protocol: Dynamic Stake

5.1 Using a Trusted Beacon

In the static version of the protocol in the previous section, we assumed that stake was static during the whole execution (i.e., one epoch), meaning that stake changing hands inside a given epoch does not affect leader election. Now we put forth a modification of protocol π_{SPoS} that can be executed over multiple epochs in such a way that each epoch's leader election process is parameterized by the stake distribution at a certain designated point of the previous epoch, allowing for change in

the stake distribution across epochs to affect the leader election process. As before, we construct the protocol in a hybrid model, enhancing the $\mathcal{F}_{LS}^{\mathcal{D}, F}$ ideal functionality to now provide randomness and auxiliary information for the leader election process throughout the epochs (the enhanced functionality will be called $\mathcal{F}_{DLS}^{\mathcal{D}, F}$). We then discuss how to implement $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ using only $\mathcal{F}_{LS}^{\mathcal{D}, F}$ and in this way reduce the assumption back to the simple common random string selected at setup.

Before describing the protocol for the case of dynamic stake, we need to explain the modification of $\mathcal{F}_{LS}^{\mathcal{D}, F}$ so that multiple epochs are considered. The resulting functionality, $\mathcal{F}_{DLS}^{\mathcal{D}, F}$, allows stakeholders to query it for the leader selection data specific to each epoch. $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ is parameterized by the initial stake of each stakeholder before the first epoch e_1 starts; in subsequent epochs, parties will take into consideration the stake distribution in the latest block of the previous epoch's first $R - 2k$ slots. Given that there is no predetermined view of the stakeholder distribution, the functionality $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ will provide only a random string and will leave the interpretation according to the stakeholder distribution to the party that is calling it. The effective stakeholder distribution is the sequence $\mathbb{S}_1, \mathbb{S}_2, \dots$ defined as follows: \mathbb{S}_1 is the initial stakeholder distribution; for slots $\{(j-1)R + 1, \dots, jR\}$ for $j \geq 2$ the effective stakeholder \mathbb{S}_j is determined by the stake allocation that is found in the latest block with time stamp at most $(j-1)R - 2k$, provided all honest parties agree on it, or is undefined if the honest parties disagree on it. The functionality $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ is defined in Figure 10.

Functionality $\mathcal{F}_{DLS}^{\mathcal{D}, F}$

$\mathcal{F}_{DLS}^{\mathcal{D}, F}$ incorporates the diffuse and key/transaction functionality from Section 2 and is parameterized by the public keys and respective stakes of the initial (before epoch e_1 starts) stakeholders $\mathbb{S}_0 = \{(\mathbf{vk}_1, s_1^0), \dots, (\mathbf{vk}_n, s_n^0)\}$ a distribution \mathcal{D} and a leader selection function F . In addition, $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ operates as follows:

- **Genesis Block Generation** Upon receiving $(\text{genblock_req}, U_i)$ from stakeholder U_i it operates as functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$ on that message.
- **Signature Key Pair Generation** It operates as functionality $\mathcal{F}_{LS}^{\mathcal{D}, F}[\text{SIG}]$.
- **Epoch Randomness Update** Upon receiving $(\text{epochrnd_req}, U_i, e_j)$ from stakeholder U_i , if $j \geq 2$ is the current epoch, $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ proceeds as follows. If ρ^j has not been set, $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ samples $\rho^j \leftarrow \mathcal{D}$. Then, $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ sends $(\text{epochrnd}, \rho^j)$ to U_i .

Figure 10: Functionality $\mathcal{F}_{DLS}^{\mathcal{D}, F}$.

We now describe protocol π_{DPoS} , which is a modified version of π_{SPoS} that updates its genesis block B_0 (and thus the leader selection process) for every new epoch. The protocol also adopts an adaptation of the static maxvalid_S function, defined so that it narrows selection to those chains which share common prefix. Specifically, it adopts the following rule, parameterized by a prefix length k :

Function $\text{maxvalid}(\mathcal{C}, \mathbb{C})$. Returns the longest chain from $\mathbb{C} \cup \{\mathcal{C}\}$ that does not fork from \mathcal{C} more than k blocks. If multiple exist it returns \mathcal{C} , if this is one of them, or it returns the one that is listed first in \mathbb{C} .

Protocol π_{DPoS} is described in Figure 11 and functions in the $\mathcal{F}_{DLS}^{\mathcal{D}, F}$ -hybrid model.

Remark 1. *The modification to $\text{maxvalid}(\cdot)$ to not diverge more than k blocks from the last chain possessed will require stakeholders to be online at least every k slots. The relevance of the rule*

Protocol π_{DPoS}

π_{DPoS} is a protocol run by a set of stakeholders, initially equal to U_1, \dots, U_n , interacting with $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, F}$ over a sequence of L slots $S = \{sl_1, \dots, sl_L\}$. π_{DPoS} proceeds as follows:

1. **Initialization** Stakeholder $U_i \in \{U_1, \dots, U_n\}$, receives from the key registration interface its public and secret key. Then it receives the current slot from the diffuse interface and in case it is sl_1 it sends $(\text{genblock_req}, U_i)$ to $\mathcal{F}_{\text{LS}}^{\mathcal{D}, F}$, receiving $(\text{genblock}, \mathbb{S}_0, \rho, F)$ as the answer. U_i sets the local blockchain $\mathcal{C} = B_0 = (\mathbb{S}_0, \rho)$ and the initial internal state $st = H(B_0)$. Otherwise, it receives from the key registration interface the initial chain \mathcal{C} , sets the local blockchain as \mathcal{C} and the initial internal state $st = H(\text{head}(\mathcal{C}))$.
2. **Chain Extension** For every slot $sl \in S$, every online stakeholder U_i performs the following steps:
 - (a) If a new epoch e_j , with $j \geq 2$, has started, U_i defines \mathbb{S}_j to be the stakeholder distribution drawn from the most recent block with time stamp less than $jR - 2k$ as reflected in \mathcal{C} and sends $(\text{epochrnd_req}, U_i, e_j)$ to $\mathcal{F}_{\text{LS}}^{\mathcal{D}, F}$, receiving $(\text{epochrnd}, \rho^j)$ as answer.
 - (b) Collect all valid chains received via broadcast into a set \mathbb{C} , verifying that for every chain $\mathcal{C}' \in \mathbb{C}$ and every block $B' = (st', d', sl', \sigma') \in \mathcal{C}'$ it holds that $\text{Vrf}_{\text{vk}'}(\sigma', (st', d', sl')) = 1$, where vk' is the verification key of the stakeholder $U' = F(\mathbb{S}_{j'}, \rho^{j'}, sl')$ with $e_{j'}$ being the epoch in which the slot B' belongs (as determined by sl'). U_i computes $\mathcal{C}' = \text{maxvalid}(\mathcal{C}, \mathbb{C})$, sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
 - (c) If U_i is the slot leader determined by $F(\mathbb{S}_j, \rho^j, sl)$ in the current epoch e_j , it generates a new block $B = (st, d, sl, \sigma)$ where st is its current state, $d \in \{0, 1\}^*$ is the data and $\sigma = \text{Sign}_{\text{sk}_i}(st, d, sl)$ is a signature on (st, d, sl) . U_i computes $\mathcal{C}' = \mathcal{C}|B$, broadcasts \mathcal{C}' , sets \mathcal{C}' as the new local chain and sets state $st = H(\text{head}(\mathcal{C}'))$.
3. **Transaction generation** as in protocol π_{SPoS} .

Figure 11: Protocol π_{DPoS}

comes from the fact that as stake shifts over time, it will be feasible for the adversary to corrupt stakeholders that used to possess a stake majority at some point without triggering $\text{Bad}^{1/2}$ and thus any adversarial chains produced due to such an event should be rejected. It is worth noting that this restriction can be easily lifted if one can trust honest stakeholders to securely erase their memory; in such case, a forward secure signature can be employed to thwart any past corruption attempt that tries to circumvent $\text{Bad}^{1/2}$.

5.2 Simulating a Trusted Beacon

While protocol π_{DPoS} handles multiple epochs and takes into consideration changes in the stake distribution, it still relies on $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, F}$ to perform the leader selection process. In this section, we show how to implement $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, F}$ through Protocol π_{DLS} , which allows the stakeholders to compute the randomness and auxiliary information necessary in the leader election.

Recall, that the only essential difference between $\mathcal{F}_{\text{LS}}^{\mathcal{D}, F}$ and $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, F}$ is the continuous generation of random strings ρ^2, ρ^3, \dots for epochs e_2, e_3, \dots . The idea is simple, protocol π_{DLS} will use a coin tossing protocol to generate unbiased randomness that can be used to define the values $\rho^j, j \geq 2$ bootstrapping on the initial random string and initial honest stakeholder distribution. However, notice that the adversary could cause a simple coin tossing protocol to fail by aborting. Thus, we build a coin tossing scheme with “guaranteed output delivery.”

Protocol π_{DLS} is described in Figure 13 and uses a publicly verifiable secret sharing (PVSS) [39].

As in the static stake case, we need to define an idealized protocol that behaves as if the

computationally secure primitives that are employed in the real protocol behave perfectly. Once again we will base our combinatorial arguments on this idealized version. We remark that we depart π_{ISPoS} as previously defined, adding further considerations about an ideal execution of the coin tossing procedure that generates randomness for the leader selection process. The assumption we will use about the PVSS scheme is that the resulting coin-flipping protocol simulates a perfect beacon with distinguishing advantage ϵ_{DLS} . Simulation here suggests that, in the case of honest majority, there is a simulator that interacts with the adversary and produces indistinguishable protocol transcripts when given the beacon value after the commitment stage. We remark that using [39] as a PVSS, a simulator can achieve simulatability in the random oracle model by taking advantage of the programmability of the oracle. Using a random oracle is by no means necessary though and the same benefits may be obtained by a CRS embedded into the genesis block.

Commitments and Coin Tossing. A coin tossing protocol allows two or more parties to obtain a uniformly random string. A classic approach to construct such a protocol is by using commitment schemes. In a commitment scheme, a *committer* carries out a *commitment phase*, which sends evidence of a given value to a *receiver* without revealing it; later on, in an *opening phase*, the committer can send that value to the receiver and convince it that the value is identical to the value committed to in the commitment phase. Such a scheme is called *binding* if it is hard for the committer to convince the receiver that he was committed to any value other than the one for which he sent evidence in the commitment phase, and it is called *hiding* if it is hard for the receiver to learn anything about the value before the opening phase. We denote the commitment phase with randomness r and message m by $\text{Com}(r, m)$ and the opening as $\text{Open}(r, m)$.

In a standard two-party coin tossing protocol [9], one party starts by sampling a uniformly random string u_1 and sending $\text{Com}(r, u_1)$. Next, the other party sends another uniformly random string u_2 in the clear. Finally, the first party opens u_1 by sending $\text{Open}(r, u_1)$ and both parties compute output $u = u_1 \oplus u_2$. Note, however, that in this classical protocol the committer may selectively choose to “abort” the protocol (by not opening the commitment) once he observes the value u_2 . While this is an intrinsic problem of the two-party setting, we can avoid this problem in the multi-party setting by relying on a verifiable secret sharing scheme and an honest majority amongst the protocol participants.

Verifiable Secret Sharing (VSS). A secret sharing scheme allows a *dealer* P_D to split a secret σ into n *shares* distributed to parties P_1, \dots, P_n , such that no adversary corrupting up to t parties can recover σ . In a Verifiable Secret Sharing (VSS) scheme [22], there is the additional guarantee that the honest parties can recover σ even if the adversary corrupts the shares held by the parties that it controls and even if the dealer itself is malicious. We define a VSS scheme as a pair of efficient dealing and reconstruction algorithms (Deal , Rec). The dealing algorithm $\text{Deal}(n, \sigma)$ takes as input the number of shares to be generated n along with the secret σ and outputs shares $\sigma_1, \dots, \sigma_n$. The reconstruction algorithm Rec takes as input shares $\sigma_1, \dots, \sigma_n$ and outputs the secret σ as long as no more than t shares are corrupted (unavailable shares are set to \perp and considered corrupted). Schoenmakers [39] developed a simple VSS scheme based on discrete logarithms suitable for our purposes.

Constructing Protocol π_{DLS} . The main problem to be solved when realizing $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ with a protocol run by the stakeholders is that of generating uniform randomness for the leader selection process while tolerating adversaries that may try to interfere by aborting or feeding incorrect information to parties. In order to generate uniform randomness ρ^j for epoch e_j , $j \geq 2$, the

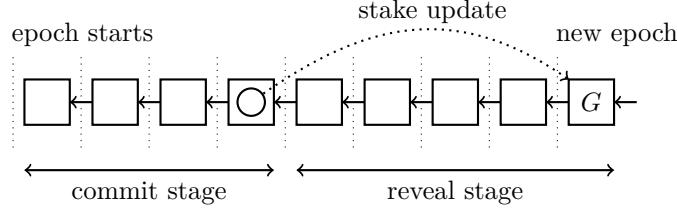


Figure 12: The two stages of the protocol π_{DPOS} that use the blockchain as a broadcast channel.

elected stakeholders for epoch e_{j-1} will employ a coin tossing scheme for which all honest parties are guaranteed to receive output as long as there is an honest majority. The protocol has two stages, commit and reveal which are split into phases. The stages of the protocol are presented in Figure 12. The *Commitment Phase* covers the whole commitment stage, and proceeds as follows: for $1 \leq i \leq n$, stakeholder U_i samples a uniformly random string $u_i \in \{0,1\}^{R \log \tau}$ and randomness r_i for the underlying commitment scheme, generates shares $\sigma_1^i, \dots, \sigma_n^i$, and posts $\text{Com}(r_i, u_i)$ to the blockchain together with the encryptions of the all the shares under the public-key of each respective shareholder. After $4k$ slots, players remove the k most recent blocks of their chain, and if commitments from a majority of stakeholders are posted on the blockchain and shares from a majority of stakeholders have been received, the reveal stage starts (in the other case the protocol halts). In the reveal stage there are two phase: the *Reveal Phase* and the *Recovery Phase*. In the reveal phase, for $1 \leq i \leq n$, stakeholder U_i posts $\text{Open}(r_i, u_i)$ to the blockchain. After $4k$ slots players remove the most recent k blocks and identify all stakeholders that have issued openings of the form $\text{Open}(r_i, u_i)$. In the final *Recovery Phase*, lasting $2k$ slots, if a stakeholder U^a that initially submitted a commitment is identified as not posting an opening to its commitment, the honest parties can post all shares $\sigma_1^a, \dots, \sigma_n^a$ in order to use $\text{Rec}(\sigma_1^a, \dots, \sigma_n^a)$ to reconstruct u^a . Finally, each stakeholder uses the values u_i obtained in the second round to compute $\rho^j = \sum_i u_i$. Protocol π_{DLS} is described in Figure 13. We remark that it is possible to run the reveal and recovery phases in parallel, however for improved efficiency we choose to run them sequentially.

5.3 Robust Transaction Ledger

We are now ready to state the main result of the section that establishes that the π_{DPOS} protocol with the protocol π_{DLS} as a sub-routine implements a robust transaction ledger under the environmental conditions that we have assumed. Recall that in the dynamic stake case we have to ensure that the adversary cannot exploit the way stake changes over time and corrupt a set of stakeholders that will enable the control of the majority of an elected committee of stakeholders in an epoch. In order to capture this dependency on stake “shifts”, we introduce the following property.

Definition 5.1. Consider two slots sl_1, sl_2 and an execution \mathcal{E} . The stake shift between sl_1, sl_2 is the maximum possible statistical distance of the two weighted-by-stake distributions that are defined using the stake reflected in the chain C_1 of some honest stakeholder active at sl_1 and the chain C_2 of some honest stakeholder active at sl_2 respectively.

Given the definition above we can now state the following theorem.

Theorem 5.2. Fix parameters $k, R, L \in \mathbb{N}, \epsilon, \sigma \in (0, 1)$. Let $R = 10k$ be the epoch length and L the total lifetime of the system. Assume the adversary is restricted to $\frac{1-\epsilon}{2} - \sigma$ relative stake and that the π_{SPOS} protocol satisfies the common prefix property with parameters R, k and probability of error ϵ_{CP} , the chain quality property with parameters $\mu \geq 1/k, k$ and probability of error ϵ_{CQ} and

Protocol π_{DLS}

π_{DLS} is a protocol run by a subset of elected stakeholders each one corresponding to a slot during an epoch e_j that lasts $R = 10k$ slots, without loss of generality denoted by U_1, \dots, U_R (which are not necessarily distinct), and entails the following phases.

1. **Commitment Phase** ($4k$ slots) When epoch e_j starts, for $1 \leq i \leq n$, stakeholder U_i samples a uniformly random string u_i and randomness r_i for the underlying commitment scheme, generates shares $\sigma_1^i, \dots, \sigma_n^i \leftarrow \text{Deal}(n, u_i)$ and encrypts each share σ_k^i under stakeholder U_k 's public-key. Finally, U_i posts the encrypted shares and commitments $\text{Com}(r_i, u_i)$ to the blockchain.
2. **Reveal Phase** ($4k$ slots) After slot $4k$, for $1 \leq i \leq n$, stakeholder U_i opens its commitment by posting $\text{Open}(r_i, u_i)$ to the blockchain provided that the blockchain contain valid shares from the majority of U_1, \dots, U_R ; if not, each U_i terminates.
3. **Recovery Phase** ($2k$ slots) After slot $8k$, for any stakeholder U^a that has not participated in the reveal phase, i.e., it has not posted in $\mathcal{C}^{[k]}$ an $\text{Open}(r_a, u_a)$ message, for $1 \leq i \leq R$, U_i submits its share σ_i^a for insertion to the blockchain. When all shares $\sigma_1^a, \dots, \sigma_n^a$ are available, each stakeholder U_i can compute $\text{Rec}(\sigma_1^a, \dots, \sigma_n^a)$ to reconstruct u_a (independently of whether U^a opens the commitment or not).

The simulation of `epochrnd_req` is then as follows.

- Given input $(\text{genblock_req}, U_i, e_j, \mathbb{S}_j)$, the stakeholder uses the commitment values in the blockchain to compute $\rho^j = \sum_{l \in \mathbb{L}} u_l$ where \mathbb{L} is the subset of stakeholders that were elected in epoch e_j . It returns $(\text{genblock}, B_0, \mathbb{S}_j)$ with $B_0 = (\mathbb{S}_j, \rho^j)$.

Figure 13: Protocol π_{DLS} .

the chain growth property with parameters $\tau \geq 1/2, k$ and probability of error ϵ_{CG} . Furthermore, assume that π_{DLS} simulates a perfect beacon with distinguishing advantage ϵ_{DLS} .

Then, the π_{DPOS} protocol satisfies persistence with parameters k and liveness with parameters $u = 2k$ throughout a period of L slots (or $\text{Bad}^{1/2}$ happens) with probability $1 - (L/R)(\epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}} + \epsilon_{\text{DLS}})$, assuming that σ is the maximum stake shift over $10k$ slots, corruption delay $D \geq 2R - 4k$ and no honest player is offline for more than k slots.

Proof. (sketch) Let us first consider the execution of π_{DPOS} when $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ is used instead of π_{DLS} . Let BAD_r be the event that any of the three properties $\text{CP}, \text{CQ}, \text{CG}$ is violated at round $r \geq 1$ while no violation of any of them occurred prior to r . It is easy to see that $\Pr[\cup_{r \leq R} \text{BAD}_r] \leq \epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}}$. Conditioning now on the negation of this event, we can repeat the argument for the second epoch, since $D \geq R$ and thus the adversary cannot influence the stakeholder selection for the second epoch. It follows that $\Pr[\cup_{r \leq L} \text{BAD}_r] \leq (L/R)(\epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}})$. It is easy now to see that persistence and liveness hold conditioning on the negation of the above event: a violation of persistence would violate common prefix. On the other hand, a violation of liveness would violate either chain growth or chain quality for the stated parameters.

Observe that the above result will continue to hold even if $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ was weakened to allow the adversary access to the random value of the next epoch $6k$ slots ahead of the end of the epoch. This is because the corruption delay $D \geq 2R - 4k = 16k$.

Finally, we examine what happens when $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ is substituted by $\mathcal{F}_{\text{LS}}^{\mathcal{D}, \mathcal{F}}$ and the execution of protocol π_{DLS} . Consider an execution with environment \mathcal{Z} and adversary \mathcal{A} and event BAD that happens with some probability β in this execution. We construct an adversary \mathcal{A}^* that operates in an execution with $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ weakened as in the previous paragraph, and induces the event BAD with roughly the same probability β . \mathcal{A}^* would operate as follows: in the first $4k$ slots, it will use an honest party to insert in the blockchain the simulated commitments of the honest parties; this is

feasible for \mathcal{A}^* as in $4k$ slots, chain growth will result in the blockchain growing by at least $2k$ blocks and thus in the first k blocks there will be at least a single honest block included. Now \mathcal{A}^* will obtain from $\mathcal{F}_{\text{DLS}}^{\mathcal{D}, \mathcal{F}}$ the value of the beacon and it will simulate the opening of all the commitments on behalf of the honest parties. Finally, in the last $2k$ slots it will perform the forced opening of all the adversarial commitments that were not opened. The protocol simulation will be repeated for each epoch and the statement of the theorem follows. \square \square

Remark 2. *We note that it is easy to extend the adversarial model to include fail-stop (and recover) corruptions in addition to Byzantine corruptions. The advantage of this mixed corruption setting, is that it is feasible to prove that we can tolerate a large number of fail-stop corruptions (arbitrarily above 50%). The intuition behind this is simple: the forkable string analysis still applies even if an arbitrary percentage of slot leaders is rendered inactive. The only necessary provision for this would be expand the parameter k inverse proportionally to the rate of non-stopped parties. We omit further details.*

6 Anonymous Communication and Stronger Adversaries

The protocols constructed in the previous section are proven secure against *delayed* adaptive corruptions, meaning that, after requesting to corrupt a given party U_i , the adversary has to wait for D slots before the corruption actually happens. However it is desirable to make D as small as possible, or even eliminate it altogether to achieve security against a standard adaptive adversary.

The delay is required because the adversary must not be able to corrupt parties once it knows that they are the slot leaders for a given slot. However, notice that the slot leaders are selected by weighting public keys by stake, while the adversary can only choose to corrupt a user U_i without knowing its public key. Thus, the adversary must be able to observe communication between U_i and the Diffuse functionality in order to determine which public key is associated with user U_i and detect when U_i is selected as a slot leader. We will show that we can eliminate the delay by extending our model with a sender anonymous broadcast channel (provided by the Diffuse functionality) and having the environment activate all parties in every round. We introduce the following modifications in the ideal functionalities:

- **Diffuse Functionality:** The functionality will work as described in Section 2 except that it will remove all information about the sender U_s of every message before delivering it to the receiver U_r 's inbox (input tape), thus ensuring that the sender remain anonymous.⁷
- **Key and Transaction Functionality:** The functionality will work as described in Section 2 except that it will allow immediate corruption of a user U upon receiving a message $(\text{Corrupt}, U)$ from the adversary.

Apart from these modifications in the ideal functionalities, we also change the environment behavior by requiring that it activates *all* users at every slot sl_j . Having all parties being activated at every slot results in an anonymity set of size equal to the number of honest parties, making it difficult for the adversary to associate a given public key with a user (*i.e.* any of the honest parties could be associated with a given public key that is not associated with a corrupted party). In this extended model we can reprove Theorem 5.2 without a delay D by strengthening the restrictions that are imposed on the environment in the following way.

⁷In practice, a sender anonymous broadcast channel with properties akin to those of the Diffuse functionality can be implemented by Mix-networks [15] or DC-networks [16] that can be executed by the nodes running the protocol.

- We will say the adversary is *restricted to less than 50% relative stake for windows of length D* if for all sets of consecutive slots of length D , the sum over all corrupted keys of the maximum stake held by each key during this period of D slots (in any possible $\mathbb{S}_j(r)$ where U_j is an honest party) is no more than 50% of the minimum total stake during this period. In case the above is violated an event $\text{Bad}_D^{1/2}$ becomes true for the given execution.

Using the above strengthened condition, we can remove the corruption delay requirement D in Theorem 5.2 by assuming that $\text{Bad}^{1/2}$ is substituted with $\text{Bad}_D^{1/2}$.

7 Incentives

So far our analysis has focused on the cryptographic adversary setting where a set of honest players operate in the presence of an adversary. In this section we consider the setting of a coalition of rational players and their incentives to deviate from honest protocol operation.

7.1 Input Endorsers

In order to address incentives, we modify further our basic protocol to assign two different roles to stakeholders. As before in each epoch there is a set of elected stakeholders that runs the secure multiparty coin flipping protocol and are the slot leaders of the epoch. Together with those there is a (not necessarily disjoint) set of stakeholders called the endorsers. Now each slot has two types of stakeholders associated with it; the slot leader who will issue the block as before and the slot *endorser* who will endorse the input to be included in the block. Moreover, contrary to slot leaders, we can elect multiple slot endorsers for each slot, nevertheless, without loss of generality we just assume a single input endorser per slot in this description. While this seems like an insignificant modification it gives us a room for improvement because of the following reason: endorsers' contributions will be acceptable even if they are d slots late, where $d \in \mathbb{N}$ is a parameter.

Note that in case no valid endorser input is available when the slot leader is about to issue the block, the leader will go ahead and issue an empty block, i.e., a block without any actual inputs (e.g., transactions in the case of a transaction ledger). Note that slot endorsers just like slot leaders are selected by weighing by stake and thus they are a representative sample of the stakeholder population. In the case of a transaction ledger the same transaction might be included by many input endorsers simultaneously. In case that a transaction is multiply present in the blockchain its first occurrence only will be its “canonical” position in the ledger. The enhanced protocol, π_{DPOSWE} , can be easily seen to have the same persistence and liveness behaviour as π_{DPOS} : the modification with endorsers does not provide any possibility for the adversary to prevent the chain from growing, accepting inputs, or being consistent. However, if we measure chain quality in terms of number of *endorsed inputs* included this produces a more favorable result: it is easy to see that the number of endorsed inputs originating from a set of stakeholders S in any k -long portion of the chain is proportional to the relative stake of S with high probability. This stems from the fact that it is sufficient that a single honest block is created for all the endorsed inputs of the last d slots to be included in it. Assuming $d \geq 2k$, any set of stakeholders S will be an endorser in a subset of the d slots with probability proportional to its cumulative stake, and thus the result follows.

As in bitcoin, stakeholders that issue blocks are incentivized to participate in the protocol by collecting transaction fees. Contrary to bitcoin, of course, one does not need to incentivize stakeholders to invest computational resources to issue blocks. Rather, *availability* and *transaction verification* should be incentivized. Nevertheless, they have to be incentivized to be online often. Any stakeholder, at minimum, must be online and operational in the following circumstances.

- In the slot prior to a slot she is the elected shareholder so that she queries the network and obtains the currently longest blockchain as well as any endorsed inputs to include in the block.
- In the slot during which she is the elected shareholder so that she issues the block containing the endorsed inputs.
- In a slot during the commit stage of an epoch where she is supposed to issue the VSS commitment of her random string.
- In a slot during the reveal stage of an epoch where she is supposed to issue the required opening shares as well as the opening to her commitment.
- In general, in sufficient frequency, to check whether she is an elected shareholder for the next or current epoch.
- In a slot during which she is the elected input endorser so that she issues the endorsed input (e.g., the set of transactions) that requires processing all available transactions and verifying them.

In order to incentivize the above actions in the setting of a transaction ledger, fees can be collected from those that issue transactions to be included in the ledger which can then be transferred to the block issuers. In bitcoin, for instance, fees can be collected by the miner that produces a block of transactions as a reward. In our setting, similarly, a reward can be given to the parties that are issuing blocks and endorsing inputs. The reward mechanism does not have to be block dependent as advocated in [34]. In our setting, it is possible to collect all fees of transactions included in a sequence of blocks in a pool and then distribute that pool to all shareholders that participated during these slots. For example, all input endorsers that were active may receive reward proportional to the number of inputs they endorsed during a period of rounds (independently of the actual number of transactions they endorsed). Other ways to distribute transaction fees are also feasible (including the one that is used by bitcoin itself—even though the bitcoin method is known to be vulnerable to attacks, e.g., the selfing-mining attack).

The reward mechanism that we will pair with input endorsers operates as follows. First we set the endorsing acceptance window, d to be $d = 2k$. Let \mathcal{C} be a chain consisting of blocks B_0, B_1, \dots . Consider the sequence of blocks that cover the j -th epoch denoted by B_1, \dots, B_s with timestamps in $\{jR + 1, \dots, (j+1)R + 2k\}$ that contain an $r \geq 0$ sequence of endorsed inputs that originate from the j -th epoch (some of them may be included as part of the $j+1$ epoch). We define the total reward pool P_R to be equal to the sum of the transaction fees that are included in the endorsed inputs that correspond to the j -th epoch. If a transaction occurs multiple times (as part of different endorsed inputs) or even in conflicting versions, only the first occurrence of the transaction is taken into account (and is considered to be part of the ledger at that position) in the calculation of P , where the total order used is induced by the order the endorsed inputs that are included in \mathcal{C} . In the sequence of these blocks, we identify by L_1, \dots, L_R the slot leaders corresponding to the slots of the epoch and by E_1, \dots, E_r the input endorsers that contributed the sequence of r endorsed inputs. Subsequently, the i -th stakeholder U_i can claim a reward up to the amount $(\beta \cdot |\{j \mid U_i = E_j\}| / r + (1 - \beta) \cdot |\{j \mid U_i = L_j\}| / R)P$ where $\beta \in [0, 1]$. Claiming a reward is performed by issuing a “coinbase” type of transaction at any point after $4k$ blocks in a subsequent epoch to the one that a reward is being claimed from.

Observe that the above reward mechanism has the following features: (i) it rewards elected committee members for just being committee members, independently of whether they issued a

block or not, (ii) it rewards the input endorsers with the inputs that they have contributed. (iii) it rewards entities for epoch j , after slot $jR + 4k$.

We proceed to show that our system is a δ -Nash (approximate) equilibrium, cf. [31, Section 2.6.6]. Specifically, the theorem states that any coalition deviating from the protocol can add at most an additive δ to its total rewards.

A technical difficulty in the above formulation is that the number of players, their relative stake, as well as the rewards they receive are based on the transactions that are generated in the course of the protocol execution itself. To simplify the analysis we will consider a setting where the number of players is static, the stake they possess does not shift over time and the protocol has negligible cost to be executed. We observe that the total rewards (and hence also utility by our assumption on protocol costs) that any coalition V of honest players are able extract from the execution lasting $L = tR + 4k + 1$ slots, is equal to

$$\mathcal{R}_V(\mathcal{E}) = \sum_{j=1}^t P_{\text{all}}^{(j)} \left(\beta \frac{IE_V^j(\mathcal{E})}{R} + (1 - \beta) \frac{SL_V^j(\mathcal{E})}{r_j} \right)$$

for any execution \mathcal{E} where common prefix holds with parameter k , where r_j is the total endorsed inputs emitted in the j -th epoch (and possibly included at any time up to the first $2k$ slots of epoch $j + 1$), $P_{\text{all}}^{(j)}$ is the reward pool of epoch j , $SL_V^j(\mathcal{E})$ is the number of times a member of V was elected to be a slot leader in epoch j and $IE_V^j(\mathcal{E})$ the number of times a member of V was selected to endorse an input in epoch j .

Observe that the actual rewards obtained by a set of rational players V in an execution \mathcal{E} might be different from $\mathcal{R}_V(\mathcal{E})$; for instance, the coalition of V may never endorse a set of inputs in which case they will obtain a smaller number of rewards. Furthermore, observe that we leave the value of $\mathcal{R}_V(\mathcal{E})$ undefined when \mathcal{E} is an execution where common prefix fails: it will not make sense to consider this value for such executions since the view of the protocol of honest parties can be divergent; nevertheless this will not affect our overall analysis since such executions will happen with sufficiently small probability.

We will establish the fact that our protocol is a δ -Nash equilibrium by proving that the coalition V , even deviating from the proper protocol behavior, it cannot obtain utility that exceeds $\mathcal{R}_V(\mathcal{E}) + \delta$ for some suitable constant $\delta > 0$.

Theorem 7.1. *Fix any $\delta > 0$; the honest strategy in the protocol is a δ -Nash equilibrium against any coalition commanding a proportion of stake less than $(1 - \epsilon)/2 - \sigma$ for some constants $\epsilon, \sigma \in (0, 1)$ as in Theorem 5.2, provided that the maximum total rewards P_{all} provided in all possible protocol executions is bounded by a polynomial in λ , while $\epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}} + \epsilon_{\text{DLS}}$ is negligible in λ .*

Proof sketch. Consider a coalition of rational players V restricted as in the statement of the theorem, that engages in a protocol execution together with a number of other players that follow the protocol faithfully for a total number of L epochs. We will show that any deviation from the protocol will not result in substantially higher rewards for V . Observe that based on Theorem 5.2, no matter the strategy of V , with probability $1 - (L/R)(\epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}})$ the protocol will enable all users to obtain the rewards they are entitled to as slot leaders and input endorsers. The latter stems from the following. First, from persistence and liveness, at least one honest block will be included every k blocks and hence, in each epoch, all input endorsers that follow the protocol will have the opportunity to act as input endorsers as many times they were elected to be. Second, the rewards received will be proportional to the times each party is an input endorser and issued a block successfully as well as equal to the number of times it is a slot leader. We observe that except with probability $(L/R)(\epsilon_{\text{CQ}} + \epsilon_{\text{CP}} + \epsilon_{\text{CG}})$ the utility received by coalition V is equal to \mathcal{R}_V . It follows

that player V has expected utility at most $E[\mathcal{R}_V] + (L/R)(\epsilon_{CQ} + \epsilon_{CP} + \epsilon_{CG})P_{All}$, where P_{All} is the maximum amount of rewards produced in the lifetime of all possible executions. The result follows by the assumption in the statement of the theorem since $(L/R)(\epsilon_{CQ} + \epsilon_{CP} + \epsilon_{CG})P_{All} \leq \delta$. \square

Remark 3. *In the above theorem, for simplicity, we assumed that protocol costs are not affecting the final utility (in essence this means that protocol costs are assumed to be negligible). Nevertheless, it is straightforward to extend the proof to cover a setting where a negative term is introduced in the payoff function for each player proportional to the number of times inputs are endorsed and the number of messages transmitted for the MPC protocol. The proof would be resilient to these modifications because endorsed inputs and MPC protocol messages cannot be stifled by the adversary and hence the reward function can be designed with suitable weights for such actions that offsets their cost. Still note that the rewards provided are assumed to be “flat” for both slots and endorsed inputs and thus the costs would also have to be flat. We leave for future work the investigation of a more refined setting where costs and rewards are proportional to the actual computational steps needed to verify transactions and issue blocks.*

Remark 4. *The reward function described, only considers the number of times an entity was an input endorser without considering the amount of work that was put to verify the given transactions. Furthermore it is not sensitive to whether a slot leader issued a block or not in its assigned time slot. We next provide some context behind these choices. First suppose that slot leaders do not receive a reward when they do not issue a block. It is easy to see that when all parties follow the protocol the parties will receive the proportion from the reward pool that is associated to block issuance roughly proportional to their stake. Nevertheless, a malicious coalition can easily increase the ratio of these rewards by performing a block withholding attack (in this case this would amount to a selfish mining attack). Given that this happens with non-negligible probability a straightforward definition of $\mathcal{R}_V(\mathcal{E})$ that respects this assignment is vulnerable to attack and hence a δ -Nash equilibrium theorem cannot be shown. Next, we consider the case of extending the reward function so that input endorsers that are rewarded based on the transactions they verify (as opposed to the flat reward we considered in the above theorem). Special care is necessary to design this function. Indeed the straightforward way to implement it, which is if the first input endorser to verify a transaction that is part of the pool can make a higher claim for its fee, then there is a strategy for an adversary to deviate from the protocol and improve its ratio of rewards: perform block withholding and/or endorsed input censorship to remove endorsed inputs from the blockchain that originate to honest parties. Then include the removed transactions in endorsed input that will be transmitted in the last possible opportunity. As before, given the attack, the natural way to define $\mathcal{R}_V(\mathcal{E})$ is susceptible to it and hence a δ -Nash equilibrium theorem cannot be shown.*

A possible direction for ameliorating the problem raised in Remark 4 above, is to share the transaction fee between all the input endorsers that endorsed it. This suggests the following modification to the protocol: whenever you are an input endorser you should attempt to include all transactions that you have collected for a sequence of k slots and retransmit your endorsed input in case it is removed from the main chain. We leave the analysis of such class of reward mechanisms for future work.

8 Stake Delegation

As discussed in the previous section, stakeholders must be online in order to generate blocks when they are selected as slot leaders. However, this might be unattractive to stakeholders with a small stake in the system. Moreover, requiring that a majority of elected stakeholders participate in the

coin tossing protocol for refreshing randomness introduces a strain on the stakeholders and the network, since it might require broadcasting and storing a large number of commitments and shares.

We mitigate these issues by providing a method for reducing the size of the group of stakeholders that engage in the coin tossing protocol. Instead of the elected stakeholders directly forming the committee that will run coin tossing, a group of delegates will act on their behalf. In more detail, we put forth a *delegation scheme*, whereby stakeholders will authorize other entities, called delegates, who may be stakeholders themselves, to represent them in the coin tossing protocol. A delegate may participate in the protocol only if it represents a certain number of stakeholders whose aggregate stake exceeds a given threshold. Such a participation threshold ensures that a “fragmentation” attack, that aims to increase the delegate population in order to hurt the performance of the protocol, cannot incur a large penalty as it is capable to force the size of the committee that runs the protocol to be small (it is worth noting that the delegation mechanism is similar to *mining pools* in proof-of-work blockchain protocols).

8.1 Minimum Committee Size

To appreciate the benefits of delegation, recall that in the basic protocol (π_{DPoS}) a committee member selected by weighing by stake is honest with probability $1/2 + \epsilon$ (this being the fraction of the stake held by honest players). Thus, the number of honest players selected by k invocations of weighing by stake is a binomial distribution. We are interested in the probability of a malicious majority, which can be directly controlled by a Chernoff bound. Specifically, if we let Y be the number of times that a malicious committee member is elected then

$$\begin{aligned} \Pr[Y \geq k/2] &= \Pr[Y \geq (1 + \delta)(1/2 - \epsilon)k] \\ &\leq \exp(-\min\{\delta^2, \delta\}(1/2 - \epsilon)k/4) \\ &< \exp(-\delta^2(1/2 - \epsilon)k/4) \end{aligned}$$

for $\delta = 2\epsilon/(1 - 2\epsilon)$. Assuming $\epsilon < 1/4$, it follows that $\delta < 1$.

Consider the case that $\epsilon = 0.05$; then we have the bound $\exp(-0.00138 \cdot k)$ which provides an error of $1/1000$ as long as $k \geq 5000$. Similarly, in the case $\epsilon = 0.1$, we have the bound $\exp(-0.00625k)$ which provides the same error for $k \geq 1100$.

We observe that in order to withstand a significant number of epochs, say 2^{15} (which, if we equate a period with one day, will be 88 years), and require error probability 2^{-40} , we need that $k \geq 32648$.

In cases where the wealth in the system is not concentrated among a small set of stakeholders the above choice is bound to create a very large committee. (Of course, the maximum size of the committee is k .)

8.2 Delegation Scheme.

The concept of delegation is simple: any stakeholder can allow a *delegate* to generate blocks on her behalf. In the context of our protocol, where a slot leader signs the block it generates for a certain slot, such a scheme can be implemented in a straightforward way based on *proxy signatures* [10].

A stakeholder can transfer the right to generate blocks by creating a *proxy signing key* that allows the delegate to sign messages of the form (st, d, sl_j) (i.e., the format of messages signed in Protocol π_{DPoS} to authenticate a block). In order to limit the delegate’s block generation power to a certain range of epochs/slots, the stakeholder can limit the proxy signing key’s valid message space to strings ending with a slot number sl_j within a specific range of values. The delegate

can use a proxy signing key from a given stakeholder to simply run Protocol π_{DPos} on her behalf, signing the blocks this stakeholder was elected to generate with the proxy signing key. This simple scheme is secure due to the *Verifiability* and *Prevention of Misuse* properties of proxy signature schemes, which ensure that any stakeholder can verify that a proxy signing key was actually issued by a specific stakeholder to a specific delegate and that the delegate can only use these keys to sign messages inside the key’s valid message space, respectively. We remark that while proxy signatures can be described as a high level generic primitive, it is easy to construct such schemes from standard digital signature schemes through delegation-by-proxy as shown in [10]. In this construction, a stakeholder signs a certificate specifying the delegates identity (e.g., its public key) and the valid message space. Later on, the delegate can sign messages within the valid message space by providing signatures for these messages under its own public key along with the signed certificate. As an added advantage, proxy signature schemes can also be built from aggregate signatures in such a way that signatures generated under a proxy signing key have essentially the same size as regular signatures [10].

An important consideration in the above setting is the fact that a stakeholder may want to withdraw her support to a stakeholder prior to its proxy signing key expiration. Observe that proxy signing keys can be uniquely identified and thus they may be revoked by a certificate revocation list within the blockchain.

8.2.1 Eligibility threshold

Delegation as described above can ameliorate fragmentation that may occur in the stake distribution. Nevertheless, this does not prevent a malicious stakeholder from dividing its stake to multiple accounts and, by refraining from delegation, induce a very large committee size. To address this, as mentioned above, a threshold T , say 1%, may be applied. This means that any delegate representing less a fraction less than T of the total stake is automatically barred from being a committee member. This can be facilitated by redistributing the voting rights of delegates representing less than T to other delegates in a deterministic fashion (e.g., starting from those with the highest stake and breaking ties according to lexicographic order). Suppose that a committee has been formed, C_1, \dots, C_m , from a total of k draws of weighing by stake. Each committee member will hold k_i such votes where $\sum_{i=1}^m k_i = k$. Based on the eligibility threshold above it follows that $m \leq T^{-1}$ (the maximum value is the case when all stake is distributed in T^{-1} delegates each holding T of the stake).

9 Attacks Discussion

We next discuss a number of practical attacks and indicate how they are reflected by our modeling and mitigated.

Double spending attacks In a double spending attack, the adversary wishes to revert a transaction that is confirmed by the network. The objective of the attack is to issue a transaction, e.g., a payment from an adversarial account holder to a victim recipient, have the transaction confirmed and then revert the transaction by, e.g., including in the ledger a second conflicting transaction. Such an attack is not feasible under the conditions of Theorem 5.2. Indeed, persistence ensures that once the transaction is confirmed by an honest player, all other honest players from that point on will never disagree regarding this transaction. Thus it will be impossible to bring the system to a state where the confirmed transaction is invalidated (assuming all preconditions of the theorem hold). See the next section for an experimental discussion about double spending.

Grinding attacks In stake grinding attacks, the adversary tries to influence the slot leader selection process to improve its chances of being selected to generate blocks (which can be used to perform other attacks such as double spending). Basically, when generating a block that is taken as input by the slot leader selection process, the adversary first tests several possible block headers and content in order to find the one that gives it the best chance of being selected as a slot leader again in the future. While this attack affects PoS based cryptocurrencies that collect randomness for the slot leader selection process from raw data in the blockchain itself (*i.e.* from block headers and content), our protocol uses a standard coin tossing protocol that is proven to generate unbiased uniform randomness as discussed in Section 5.2. We show that an adversary cannot influence the randomness generated in Figure 13, which is guaranteed to be uniformly random, thus guaranteeing that slot leaders are selected with probability proportional to their stake.

Transaction denial attacks In a transaction denial attack, the adversary wishes to prevent a certain transaction from becoming confirmed. For instance, the adversary may want to target a specific account and prevent the account holder from issuing an outgoing transaction. Such an attack is not feasible under the conditions of Theorem 5.2. Indeed, liveness ensures that, provided the transaction is attempted to be inserted for a sufficient number of slots by the network, it will be eventually confirmed.

Desynchronization attacks In a desynchronization attack, a shareholder behaves honestly but is nevertheless incapable of synchronizing correctly with the rest of the network. This leads to ill-timed issuing of blocks and being offline during periods when the shareholder is supposed to participate. Such an attack can be mounted by preventing the party’s access to a time server or any other mechanism that allows synchronization between parties. Moreover, a desynchronization may also occur due to exceedingly long delays in message delivery. Our model allows parties to become desynchronized by incorporating them into the adversary. No guarantees of liveness and persistence are provided for desynchronized parties and thus we can get security as long as parties with less than 50% of stake get desynchronized. If more than parties get desynchronized our protocol can fail. More general models like partial synchrony [19, 35] are interesting to consider in the PoS design setting.

Eclipse attacks In an eclipse attack, message delivery to a shareholder is violated due to a subversion in the peer-to-peer message delivery mechanism. As in the case of desynchronization attacks, our model allows parties to be eclipse attacked by incorporating them into the adversary. No guarantees of liveness or persistence are provided for such parties.

51% attacks A 51% attack occurs whenever the adversary controls more than the majority of the stake in the system. It is easy to see that any sequence of slots in such a case is with very high probability forkable and thus once the system finds itself in such setting the honest stakeholders may be placed in different forks for long periods of time. Both persistence and liveness can be violated.

Bribery Attacks In bribery attacks [11], an adversary deliberately pays miners (through cryptocurrencies or fiat money) to work on specific blocks and forks, aiming at generating an arbitrary fork that benefits the adversary (*e.g.* by supporting a double spending attack). Miners of PoW based cryptocurrencies do not have to own any stake in order to mine blocks, which makes this attack strategy feasible. In this setting, if the adversary offers a bribe higher than the reward

for correctly generating a block, any rational miner has a clear incentive to accept the bribe and participate in the attack since it increases the miner’s financial outcome. However, in our PoS based protocol, malicious slot leaders who agree to deliberately attack the system not only risk to forego any potential profit they would earn from behaving honestly but may also risk to lose equity. Notice that slot leaders must have money invested in the system in order to be able to generate blocks and if an attack against the system is observed it might bring currency value down. Even if the bribe is higher than the reward for correct behavior, the loss from currency devaluation can easily offset any additional profits made by participating in this attack. Hence, bribery attacks may be less effective against a PoS based consensus protocol than a PoW based one. Currently our rationality model does not formally encompass this attack strategy and investigating its efficacy against PoS based consensus protocols is left as a future work.

Long-range attacks An attacker who wishes to double spend at a later point in time can mount a long-range attack [12] by computing a longer valid chain that starts right after the genesis block where it is the single stakeholder actively participating in the protocol. Even if this attacker owns a small fraction of the total stake, it can locally compute this chain generating only the blocks for slots where it is elected the slot leader and keep generating blocks ahead of current time until its alternative chain has more blocks than the main chain. Now, the attacker can post a transaction to the main chain, wait for it to be confirmed (and for goods to be delivered in exchange for the transaction) and present the longer alternative chain to invalidate its previously confirmed transaction. This attack is ineffective against Ouroboros for two reasons: Protocol π_{DLS} will only output valid leader selection data allowing for the protocol to continue if a majority of the stakeholders participate (or have delegates participate on their behalf) and stakeholders will reject blocks generated for slots that are far ahead of time. Since the alternative chain is generated artificially with blocks and protocol messages generated solely by an attacker who controls a small fraction of the stake the leader selection data needed to start new epochs will be considered invalid by other nodes. Even if the attacker could find a strategy to generate an alternative chain with valid leader selection data, presenting this chain and its blocks generated at slots that are far ahead of time would not result in a successful attack since those blocks far ahead of time would be rejected by the honest stakeholders and the final alternative chain would be shorter than the main chain.

Nothing at stake attacks The “nothing at stake” problem refers in general to attacks against PoS blockchain systems that are facilitated by shareholders continuing simultaneously multiple blockchains exploiting the fact that little computational effort is needed to build a PoS blockchain. Provided that stakeholders are frequently online, nothing at stake is taken care of by our analysis of forkable strings (even if the adversary brute-forces all possible strategies to fork the evolving blockchain in the near future, there is none that is viable), and our chain selection rule that instructs players to ignore very deep forks that deviate from the block they received the last time they were online. It is also worth noting that, contrary to PoW-based blockchains, in our protocol it is infeasible to have a fork generated in earnest by two shareholders. This is because slots are uniquely assigned and thus at any given moment there is a single uniquely identified shareholder that is elected to advance the blockchain. Players following the longest chain rule will adopt the newly minted block (unless the adversary presents at that moment an alternative blockchain using older blocks). It is remarked in [13] that the “tragedy of commons” might lead stakeholders in some PoS based schemes to adhere to attacks because they do not have the power to deter attacks by themselves and would incur financial losses even if they did not join the attack. This would lead rational stakeholders to accept small bribes in alternative currencies that might at least obtain

some financial gain. However, in the incentive structure of Ouroboros, slot leaders and endorsers who could potentially join an attack would receive rewards in both the main and the adversarial chain, resulting in those stakeholders not achieving higher profits by joining the attack.

Past majority attacks As stake moves our assumption is that only the *current* majority of stakeholders is honest. This means that past account keys (which potentially do not hold any stake at present) may be compromised. This leads to a potential vulnerability for any PoS system since a set of malicious shareholders from the past can build an alternative blockchain exploiting such old accounts and the fact that it is effortless to build such a blockchain. In light of Theorem 5.2 such attack can only occur against shareholders who are not frequently online to observe the evolution of the system or in case the stake shifts are higher than what is anticipated by the preconditions of the theorem. This can be seen a special instance of the nothing at stake problem, where the attacker no longer owns any stake in the system and is thus free from any financial losses when conducting the attack.

Selfish-mining In this type of attack, an attacker withholds blocks and releases them strategically attempting to drop honestly generated blocks from the main chain. In this way the attacker reduces chain growth and increases the relative ratio of adversarially generated blocks. In conventional reward schemes, as that of bitcoin, this has serious implications as it enables the attacker to obtain a higher rate of rewards compared to the rewards it would be receiving in case it was following the honest strategy. Using our reward mechanism however, selfish mining attacks are neutralized. The intuition behind this, is that input endorsers, who are the entities that receive rewards proportionally to their contributions, cannot be stifled because of block withholding: any input endorser can have its contribution accepted for a sufficiently long period of time after its endorsement took place, thus ensuring it will be incorporated into the blockchain (due to sufficient chain quality and chain growth). Given that input endorsers' contributions are (approximately) proportional to their stake this ensures that reward distribution cannot be affected substantially by block withholding.

10 Experimental Results

We have implemented a prototype instantiation of Ouroboros in Haskell as well as in the Rust-based Parity Ethereum client in order to evaluate its concrete performance. More specifically, we have implemented Protocol π_{DPOS} using Protocol π_{DLS} to generate leader selection parameters (i.e., generating fresh randomness for the weighed stake sampling procedure). For this instantiation, we use the PVSS scheme of [39] implemented over the elliptic curve secp256r1. This PVSS scheme's share verification information includes a commitment to the secret, which is also used as the commitment specified in protocol π_{DLS} ; this eliminates the need for a separate commitment to be generated and stored in the blockchain. In order to obtain better efficiency, the final output ρ of Protocol π_{DLS} is a uniformly random binary string of 32 bytes. This string is then used as a seed for a PRG (ChaCha in our implementation, [8]) and stretched into R random labels of $\log \tau$ bits corresponding to each slot in an epoch. The weighing by stake leader selection process is then implemented by using the random binary string associated to each epoch to perform the sequence of coin-flips for selecting a stakeholder. The signature scheme used for signing blocks is ECDSA, also implemented over curve secp256r1.

10.1 Transaction Confirmation Time Under Optimal Network Conditions

We first examine the time required for confirming a transaction in a setting where the network is not under substantial load and transactions are processed as they appear.

Adversary	BTC	OB Covert	OB General
0.10	50	3	5
0.15	80	5	8
0.20	110	7	12
0.25	150	11	18
0.30	240	18	31
0.35	410	34	60
0.40	890	78	148
0.45	3400	317	663

Figure 14: Transaction confirmation times in minutes that achieve assurance 99.9% against a hypothetical double spending attack with different levels of adversarial power for Bitcoin and Ouroboros (both covert and general adversaries).

In Fig. 14 we lay out a comparison in terms of transaction confirmation time between Bitcoin and Ouroboros showing how much a verifier has to wait to be sure that the best possible⁸ double-spending attack succeeds with probability less than 0.1%. In the case of Bitcoin, we consider a double-spending attacker that commands a certain percentage of total hashing power and wishes to revert a transaction. The attacker attempts to double-spend via a block-withholding attack as described in the same paper (the attacker mines a private fork and releases it when it is long enough). In the case of Ouroboros we consider a double spending attacker that attempts to brute force the space of all possible forks for the current slot leader distribution in a certain segment of the protocol and commands a certain percentage of the total stake. We consider both the covert and the general adversarial setting for Ouroboros.

In all of the scenarios, we measure the number of minutes that one has to wait in order to achieve probability of double spending less than 0.1%. In Fig. 15 we present a graph that illustrates the speedup graphically.

We note that the above measurements compare our Ouroboros implementation with Bitcoin in the way the two systems are parameterized (with 10 minute block production rate for Bitcoin and 20 second slots for Ouroboros, a conservative parameter selection). Exploring alternative parameterizations for Bitcoin (such as making the proof-of-work easier) can speed up the transaction processing, nevertheless this cannot be done without carefully measuring the impact on overall security.

10.2 Absolute Performance of Ouroboros

We implemented Ouroboros as an instance of the Rust-based Ethereum Parity client.⁹ Subsequently, experiments were run using Amazon’s Elastic Compute Cloud (EC2) ‘c4.2xlarge’ instances in the ‘us-east-1’ region with a smaller “runner” instance responsible for coordinating each of the “worker” instances.

Each experiment consists of several steps:

⁸The “best possible” is only in the the case of Ouroboros, for Bitcoin we use the best known attack.

⁹Ethcore - Parity. <https://ethcore.io/parity.html>

Confirmation time speed up of Ouroboros over BTC

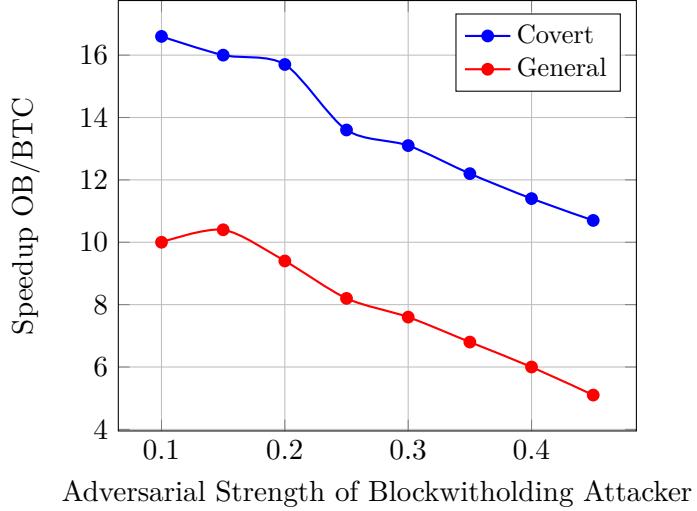


Figure 15: Ouroboros vs. Bitcoin speedup of transaction confirmation time against a hypothetical double spending attacker for assurance level 99.9%. Ouroboros is at least 10 to 5 times faster for regular adversaries and 16 to 10 times faster for covert adversaries.

1. Each worker instance builds a clean Docker image containing a specific revision of our fork of the Parity software¹⁰ containing the Ouroboros proof-of-concept changes based on the Parity 1.6.8 release.
2. Each worker instance is started in an “isolated” mode where none of the nodes talk to each other. During this period, a Parity account is recovered on each node and a start time for the network is established.
3. Each worker instance is restarted in a production mode that allows communication between the nodes and transactions to be mined.
4. A single worker instance is informed about all the other nodes. All nodes become aware of all other nodes via Parity’s peer-to-peer discovery methods.
5. Each worker instance has a number of transactions generated and ingested.

In each experiment, 650,000 total transactions are generated between the participating nodes who shared stake equally. The amount transferred in any given transaction is small enough to avoid any account running out of funds. Each instance generates all the transactions using a hard-coded shared random seed, then keeps the transactions originating from the local user account. 20 transactions are saved in a single JSON file, ready to be directly passed to the Parity RPC endpoint using the ‘curl’ command line tool. During ingestion, a single file of 20 transactions is ingested and one second is spent idle between each file to avoid overwhelming the instances with too many requests.

Various setups were tested, focusing on adjusting the Ouroboros slot duration and the number of participating nodes. 10, 20, 30, and 40 nodes were tested, ultimately limited by the number of

¹⁰ Available from <https://github.com/input-output-hk/parity/tree/experiment-2> (020fd77dc70d3f25e0e0f44bd6b1e19ccf3790d3)

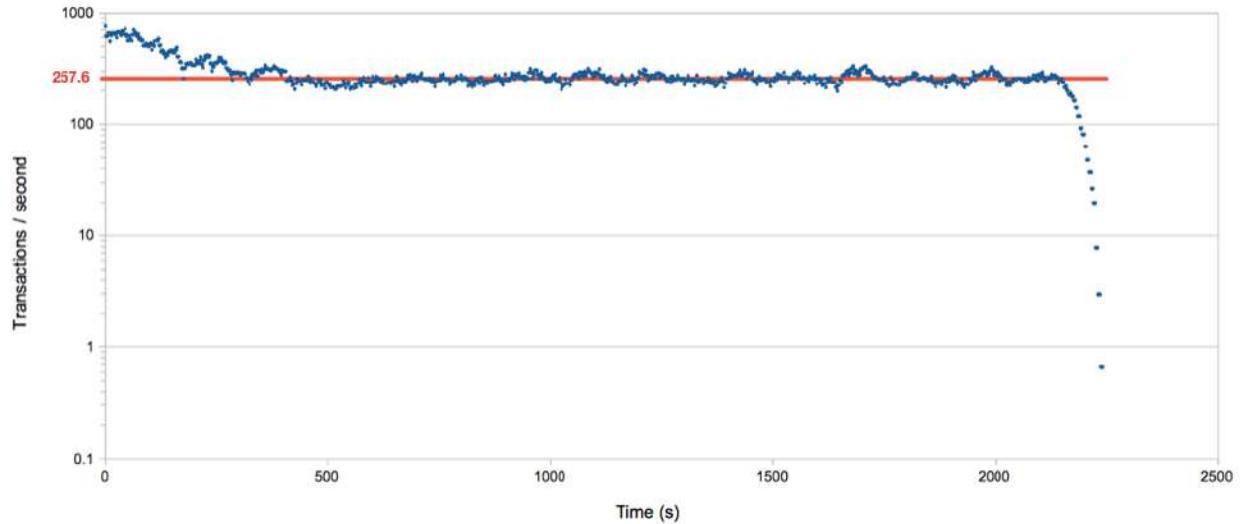


Figure 16: Measuring transactions per second in a 40 node, equal stake deployment with slot length of 5 seconds.

instances allowed in a single EC2 region. Slot durations of 5, 10, and 20 seconds were also tested. Variance between experiments was small. In Figure 16 we present the case of 40 nodes and slot length of 5 seconds that exhibits a median value of 257.6 transaction per second.

11 Acknowledgements

We thank Ioannis Konstantinou who contributed in a preliminary version of our protocol. We thank Lars Brünjes, Duncan Coutts, Kawin Worrasangasilpa for comments on previous drafts of the article. We thank Peter Gaži for comments on previous drafts of the article and assisting us to generalize Theorem 4.26 to viable forks. We thank George Agapov for the prototype implementation of our protocol in Haskell and Jake Goulding for the Parity based implementation.

References

- [1] Noga Alon and Joel Spencer. *The Probabilistic Method*. Wiley, 3rd edition, 2008.
- [2] Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 538–557. Springer, 2014.
- [3] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.
- [4] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *CoRR*, abs/1406.5694, 2014.

- [5] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]. *SIGMETRICS Performance Evaluation Review*, 42(3):34–37, 2014.
- [6] Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. *IACR Cryptology ePrint Archive*, 2016:918, 2016.
- [7] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
- [8] Daniel J. Bernstein. Chacha, a variant of salsa20. In *SASC: The State of the Art of Stream Ciphers.*, 2008.
- [9] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81, CRYPTO 81, IEEE Workshop on Communications Security, Santa Barbara, California, USA, August 24-26, 1981.*, pages 11–15. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04, 1981.
- [10] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *J. Cryptology*, 25(1):57–115, 2012.
- [11] Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 19–26. Springer, 2016.
- [12] Vitalik Buterin. Long-range attacks: The serious problem with adaptive proof of work. <https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/>, 2014.
- [13] Vitalik Buterin. Proof of stake faq. <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ>, 2016.
- [14] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.
- [15] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
- [16] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *J. Cryptology*, 1(1):65–75, 1988.
- [17] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *23rd Annual Network and Distributed System Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.
- [18] Bernardo Machado David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. *IACR Cryptology ePrint Archive*, 2017:573, 2017.

- [19] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.
- [20] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 585–605. Springer, 2015.
- [21] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Angelos D. Keromytis, editor, *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*. Springer, 2014.
- [22] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.
- [23] Bryan Ford. Delegative democracy. <http://www.brynosaurus.com/deleg/deleg.pdf>, 2002.
- [24] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
- [25] Charles M Grinstead and J Laurie Snell. *Introduction to Probability*. American Mathematical Society, 2nd edition, 1997.
- [26] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. <http://eprint.iacr.org/2015/1019>.
- [27] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.
- [28] Tal Moran and Ilan Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. <http://eprint.iacr.org/2016/035>.
- [29] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995.
- [30] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. <http://bitcoin.org/bitcoin.pdf>, 2008.
- [31] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
- [32] Karl J. O’Dwyer and David Malone. Bitcoin mining and its energy footprint. *ISSC 2014 / CICT 2014, Limerick, June 26-27, 2014*.
- [33] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. Spacemint: A cryptocurrency based on proofs of space. *IACR Cryptology ePrint Archive*, 2015:528, 2015.
- [34] Rafael Pass. Cryptography and game theory. Security and Cryptography for Networks, 2016, invited talk., 2016.

- [35] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
- [36] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. *IACR Cryptology ePrint Archive*, 2016:916, 2016.
- [37] Alexander Russell, Christopher Moore, Aggelos Kiayias, and Saad Quader. Forkable strings are rare. *Cryptology ePrint Archive*, Report 2017/241, March 2017. <http://eprint.iacr.org/2017/241>.
- [38] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.
- [39] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 1999.

CryptoNote v 2.0

Nicolas van Saberhagen

October 17, 2013

1 Introduction

“Bitcoin” [1] has been a successful implementation of the concept of p2p electronic cash. Both professionals and the general public have come to appreciate the convenient combination of public transactions and proof-of-work as a trust model. Today, the user base of electronic cash is growing at a steady pace; customers are attracted to low fees and the anonymity provided by electronic cash and merchants value its predicted and decentralized emission. Bitcoin has effectively proved that electronic cash can be as simple as paper money and as convenient as credit cards.

Unfortunately, Bitcoin suffers from several deficiencies. For example, the system’s distributed nature is inflexible, preventing the implementation of new features until almost all of the network users update their clients. Some critical flaws that cannot be fixed rapidly deter Bitcoin’s widespread propagation. In such inflexible models, it is more efficient to roll-out a new project rather than perpetually fix the original project.

In this paper, we study and propose solutions to the main deficiencies of Bitcoin. We believe that a system taking into account the solutions we propose will lead to a healthy competition among different electronic cash systems. We also propose our own electronic cash, “CryptoNote”, a name emphasizing the next breakthrough in electronic cash.

2 Bitcoin drawbacks and some possible solutions

2.1 Traceability of transactions

Privacy and anonymity are the most important aspects of electronic cash. Peer-to-peer payments seek to be concealed from third party’s view, a distinct difference when compared with traditional banking. In particular, T. Okamoto and K. Ohta described six criteria of ideal electronic cash, which included “privacy: relationship between the user and his purchases must be untraceable by anyone” [30]. From their description, we derived two properties which a fully anonymous electronic cash model must satisfy in order to comply with the requirements outlined by Okamoto and Ohta:

Untraceability: for each incoming transaction all possible senders are equiprobable.

Unlinkability: for any two outgoing transactions it is impossible to prove they were sent to the same person.

Unfortunately, Bitcoin does not satisfy the untraceability requirement. Since all the transactions that take place between the network’s participants are public, any transaction can be

unambiguously traced to a unique origin and final recipient. Even if two participants exchange funds in an indirect way, a properly engineered path-finding method will reveal the origin and final recipient.

It is also suspected that Bitcoin does not satisfy the second property. Some researchers stated ([33, 35, 29, 31]) that a careful blockchain analysis may reveal a connection between the users of the Bitcoin network and their transactions. Although a number of methods are disputed [25], it is suspected that a lot of hidden personal information can be extracted from the public database.

Bitcoin's failure to satisfy the two properties outlined above leads us to conclude that it is not an anonymous but a pseudo-anonymous electronic cash system. Users were quick to develop solutions to circumvent this shortcoming. Two direct solutions were "laundering services" [2] and the development of distributed methods [3, 4]. Both solutions are based on the idea of mixing several public transactions and sending them through some intermediary address; which in turn suffers the drawback of requiring a trusted third party.

Recently, a more creative scheme was proposed by I. Miers et al. [28]: "Zerocoin". Zerocoin utilizes a cryptographic one-way accumulators and zero-knowledge proofs which permit users to "convert" bitcoins to zerocoins and spend them using anonymous proof of ownership instead of explicit public-key based digital signatures. However, such knowledge proofs have a constant but inconvenient size - about 30kb (based on today's Bitcoin limits), which makes the proposal impractical. Authors admit that the protocol is unlikely to ever be accepted by the majority of Bitcoin users [5].

2.2 The proof-of-work function

Bitcoin creator Satoshi Nakamoto described the majority decision making algorithm as "one-CPU-one-vote" and used a CPU-bound pricing function (double SHA-256) for his proof-of-work scheme. Since users vote for the single history of transactions order [1], the reasonableness and consistency of this process are critical conditions for the whole system.

The security of this model suffers from two drawbacks. First, it requires 51% of the network's mining power to be under the control of honest users. Secondly, the system's progress (bug fixes, security fixes, etc...) require the overwhelming majority of users to support and agree to the changes (this occurs when the users update their wallet software) [6]. Finally this same voting mechanism is also used for collective polls about implementation of some features [7].

This permits us to conjecture the properties that must be satisfied by the proof-of-work pricing function. Such function must not enable a network participant to have a *significant* advantage over another participant; it requires a parity between common hardware and high cost of custom devices. From recent examples [8], we can see that the SHA-256 function used in the Bitcoin architecture does not possess this property as mining becomes more efficient on GPUs and ASIC devices when compared to high-end CPUs.

Therefore, Bitcoin creates favourable conditions for a large gap between the voting power of participants as it violates the "one-CPU-one-vote" principle since GPU and ASIC owners possess a much larger voting power when compared with CPU owners. It is a classical example of the Pareto principle where 20% of a system's participants control more than 80% of the votes.

One could argue that such inequality is not relevant to the network's security since it is not the small number of participants controlling the majority of the votes but the honesty of these participants that matters. However, such argument is somewhat flawed since it is rather the possibility of cheap specialized hardware appearing rather than the participants' honesty which poses a threat. To demonstrate this, let us take the following example. Suppose a malevolent individual gains significant mining power by creating his own mining farm through the cheap

hardware described previously. Suppose that the global hashrate decreases significantly, even for a moment, he can now use his mining power to fork the chain and double-spend. As we shall see later in this article, it is not unlikely for the previously described event to take place.

2.3 Irregular emission

Bitcoin has a predetermined emission rate: each solved block produces a fixed amount of coins. Approximately every four years this reward is halved. The original intention was to create a limited smooth emission with exponential decay, but in fact we have a piecewise linear emission function whose breakpoints may cause problems to the Bitcoin infrastructure.

When the breakpoint occurs, miners start to receive only half of the value of their previous reward. The absolute difference between 12.5 and 6.25 BTC (projected for the year 2020) may seem tolerable. However, when examining the 50 to 25 BTC drop that took place on November 28 2012, felt inappropriate for a significant number of members of the mining community. Figure 1 shows a dramatic decrease in the network's hashrate in the end of November, exactly when the halving took place. This event could have been the perfect moment for the malevolent individual described in the proof-of-work function section to carry-out a double spending attack [36].

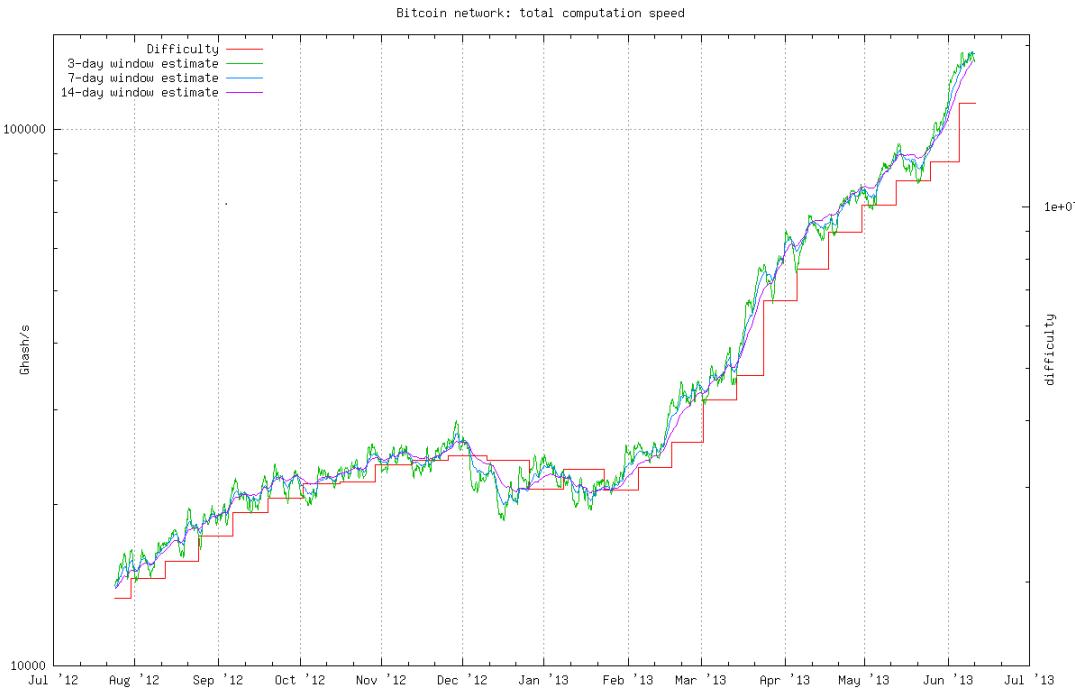


Fig. 1. Bitcoin hashrate chart
(source: <http://bitcoin.sipa.be>)

2.4 Hardcoded constants

Bitcoin has many hard-coded limits, where some are natural elements of the original design (e.g. block frequency, maximum amount of money supply, number of confirmations) whereas other seem to be artificial constraints. It is not so much the limits, as the inability of quickly changing

them if necessary that causes the main drawbacks. Unfortunately, it is hard to predict when the constants may need to be changed and replacing them may lead to terrible consequences.

A good example of a hardcoded limit change leading to disastrous consequences is the block size limit set to 250kb¹. This limit was sufficient to hold about 10000 standard transactions. In early 2013, this limit had almost been reached and an agreement was reached to increase the limit. The change was implemented in wallet version 0.8 and ended with a 24-blocks chain split and a successful double-spend attack [9]. While the bug was not in the Bitcoin protocol, but rather in the database engine it could have been easily caught by a simple stress test if there was no artificially introduced block size limit.

Constants also act as a form of centralization point. Despite the peer-to-peer nature of Bitcoin, an overwhelming majority of nodes use the official reference client [10] developed by a small group of people. This group makes the decision to implement changes to the protocol and most people accept these changes irrespective of their “correctness”. Some decisions caused heated discussions and even calls for boycott [11], which indicates that the community and the developers may disagree on some important points. It therefore seems logical to have a protocol with user-configurable and self-adjusting variables as a possible way to avoid these problems.

2.5 Bulky scripts

The scripting system in Bitcoin is a heavy and complex feature. It *potentially* allows one to create sophisticated transactions [12], but some of its features are disabled due to security concerns and some have never even been used [13]. The script (including both senders’ and receivers’ parts) for the most popular transaction in Bitcoin looks like this:

```
<sig> <pubKey> OP DUP OP HASH160 <pubKeyHash> OP EQUALVERIFY OP CHECKSIG.
```

The script is 164 bytes long whereas its only purpose is to check if the receiver possess the secret key required to verify his signature.

3 The CryptoNote Technology

Now that we have covered the limitations of the Bitcoin technology, we will concentrate on presenting the features of CryptoNote.

4 Untraceable Transactions

In this section we propose a scheme of fully anonymous transactions satisfying both untraceability and unlinkability conditions. An important feature of our solution is its autonomy: the sender is not required to cooperate with other users or a trusted third party to make his transactions; hence each participant produces a cover traffic independently.

4.1 Literature review

Our scheme relies on the cryptographic primitive called a *group signature*. First presented by D. Chaum and E. van Heyst [19], it allows a user to sign his message on behalf of the group. After signing the message the user provides (for verification purposes) not his own single public

¹This is so-called “soft limit” — the reference client restriction for creating new blocks. Hard maximum of possible blocksize was 1 MB

key, but the keys of all the users of his group. A verifier is convinced that the real signer is a member of the group, but cannot exclusively identify the signer.

The original protocol required a trusted third party (called the Group Manager), and he was the only one who could trace the signer. The next version called a *ring signature*, introduced by Rivest et al. in [34], was an autonomous scheme without Group Manager and anonymity revocation. Various modifications of this scheme appeared later: *linkable ring signature* [26, 27, 17] allowed to determine if two signatures were produced by the same group member, *traceable ring signature* [24, 23] limited excessive anonymity by providing possibility to trace the signer of two messages with respect to the same metainformation (or “tag” in terms of [24]).

A similar cryptographic construction is also known as a *ad-hoc group signature* [16, 38]. It emphasizes the arbitrary group formation, whereas group/ring signature schemes rather imply a fixed set of members.

For the most part, our solution is based on the work “Traceable ring signature” by E. Fujisaki and K. Suzuki [24]. In order to distinguish the original algorithm and our modification we will call the latter a *one-time ring signature*, stressing the user’s capability to produce only one valid signature under his private key. We weakened the traceability property and kept the linkability only to provide one-timeness: the public key may appear in many foreign verifying sets and the private key can be used for generating a unique anonymous signature. In case of a double spend attempt these two signatures will be linked together, but revealing the signer is not necessary for our purposes.

4.2 Definitions

4.2.1 Elliptic curve parameters

As our base signature algorithm we chose to use the fast scheme EdDSA, which is developed and implemented by D.J. Bernstein et al. [18]. Like Bitcoin’s ECDSA it is based on the elliptic curve discrete logarithm problem, so our scheme could also be applied to Bitcoin in future.

Common parameters are:

q : a prime number; $q = 2^{255} - 19$;

d : an element of \mathbb{F}_q ; $d = -121665/121666$;

E : an elliptic curve equation; $-x^2 + y^2 = 1 + dx^2y^2$;

G : a base point; $G = (x, -4/5)$;

l : a prime order of the base point; $l = 2^{252} + 27742317777372353535851937790883648493$;

\mathcal{H}_s : a cryptographic hash function $\{0, 1\}^* \rightarrow \mathbb{F}_q$;

\mathcal{H}_p : a deterministic hash function $E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q)$.

4.2.2 Terminology

Enhanced privacy requires a new terminology which should not be confused with Bitcoin entities.

private ec-key is a standard elliptic curve private key: a number $a \in [1, l - 1]$;

public ec-key is a standard elliptic curve public key: a point $A = aG$;

one-time keypair is a pair of private and public ec-keys;

private user key is a pair (a, b) of two different private ec-keys;

tracking key is a pair (a, B) of private and public ec-key (where $B = bG$ and $a \neq b$);

public user key is a pair (A, B) of two public ec-keys derived from (a, b) ;

standard address is a representation of a public user key given into human friendly string with error correction;

truncated address is a representation of the second half (point B) of a public user key given into human friendly string with error correction.

The transaction structure remains similar to the structure in Bitcoin: every user can choose several independent incoming payments (transactions outputs), sign them with the corresponding private keys and send them to different destinations.

Contrary to Bitcoin's model, where a user possesses unique private and public key, in the proposed model a sender generates a one-time public key based on the recipient's address and some random data. In this sense, an incoming transaction for the same recipient is sent to a one-time public key (not directly to a unique address) and only the recipient can recover the corresponding private part to redeem his funds (using his unique private key). The recipient can spend the funds using a ring signature, keeping his ownership and actual spending anonymous. The details of the protocol are explained in the next subsections.

4.3 Unlinkable payments

Classic Bitcoin addresses, once being published, become unambiguous identifier for incoming payments, linking them together and tying to the recipient's pseudonyms. If someone wants to receive an "untied" transaction, he should convey his address to the sender by a private channel. If he wants to receive different transactions which cannot be proven to belong to the same owner he should generate all the different addresses and never publish them in his own pseudonym.

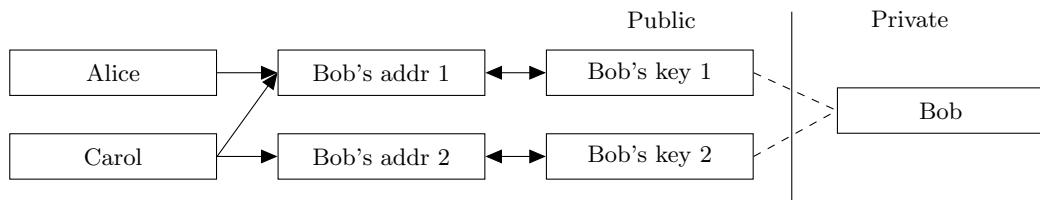


Fig. 2. Traditional Bitcoin keys/transactions model.

We propose a solution allowing a user to publish **a single address** and receive unconditional unlinkable payments. The destination of each CryptoNote output (by default) is a public key, derived from recipient's address and sender's random data. The main advantage against Bitcoin is that every destination key is unique by default (unless the sender uses the same data for each of his transactions to the same recipient). Hence, there is no such issue as "address reuse" by design and no observer can determine if any transactions were sent to a specific address or link two addresses together.

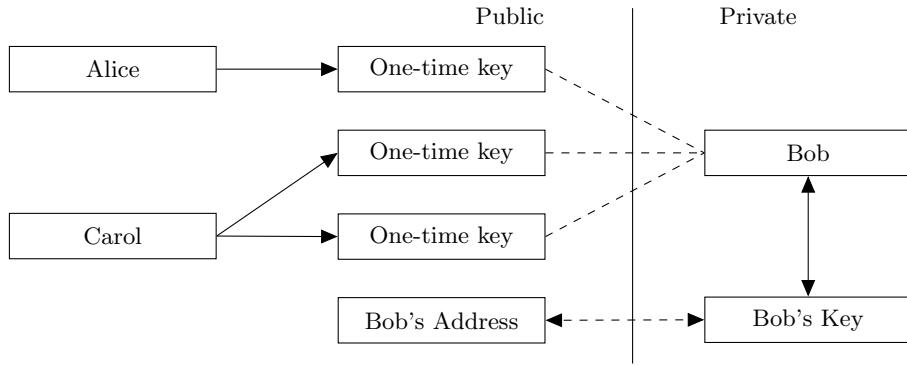


Fig. 3. CryptoNote keys/transactions model.

First, the sender performs a Diffie-Hellman exchange to get a shared secret from his data and half of the recipient's address. Then he computes a one-time destination key, using the shared secret and the second half of the address. Two different ec-keys are required from the recipient for these two steps, so a standard CryptoNote address is nearly twice as large as a Bitcoin wallet address. The receiver also performs a Diffie-Hellman exchange to recover the corresponding secret key.

A standard transaction sequence goes as follows:

1. Alice wants to send a payment to Bob, who has published his standard address. She unpacks the address and gets Bob's public key (A, B) .
2. Alice generates a random $r \in [1, l - 1]$ and computes a one-time public key $P = \mathcal{H}_s(rA)G + B$.
3. Alice uses P as a destination key for the output and also packs value $R = rG$ (as a part of the Diffie-Hellman exchange) somewhere into the transaction. Note that she can create other outputs with unique public keys: different recipients' keys (A_i, B_i) imply different P_i even with the same r .

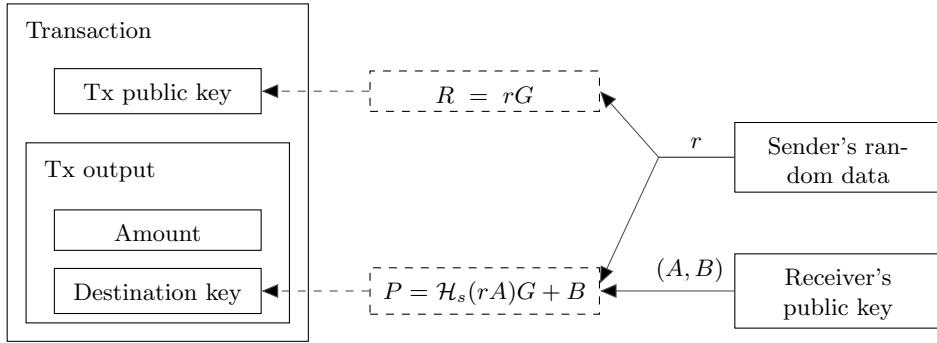


Fig. 4. Standard transaction structure.

4. Alice sends the transaction.
5. Bob checks every passing transaction with his private key (a, b) , and computes $P' = \mathcal{H}_s(aR)G + B$. If Alice's transaction for with Bob as the recipient was among them, then $aR = arG = rA$ and $P' = P$.

6. Bob can recover the corresponding one-time private key: $x = \mathcal{H}_s(aR) + b$, so as $P = xG$. He can spend this output at any time by signing a transaction with x .

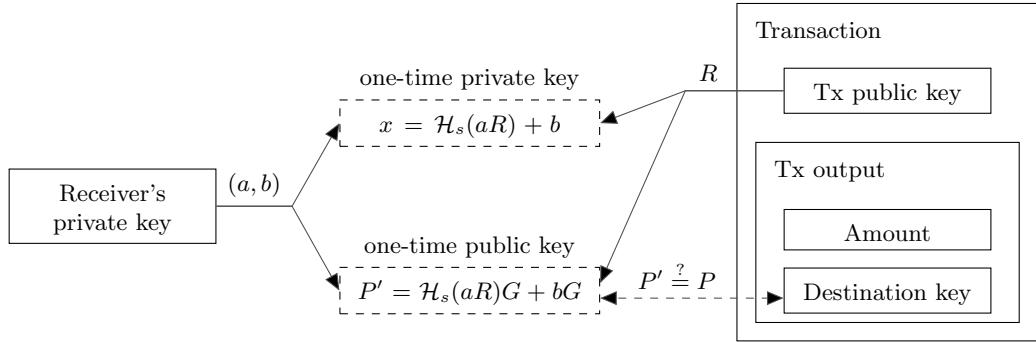


Fig. 5. Incoming transaction check.

As a result Bob gets incoming payments, associated with one-time public keys which are **unlinkable** for a spectator. Some additional notes:

- When Bob “recognizes” his transactions (see step 5) he practically uses only half of his private information: (a, B) . This pair, also known as the **tracking key**, can be passed to a third party (Carol). Bob can delegate her the processing of new transactions. Bob doesn’t need to explicitly trust Carol, because she can’t recover the one-time secret key b without Bob’s full private key (a, b) . This approach is useful when Bob lacks bandwidth or computation power (smartphones, hardware wallets etc.).
- In case Alice wants to prove she sent a transaction to Bob’s address she can either disclose r or use any kind of zero-knowledge protocol to prove she knows r (for example by signing the transaction with r).
- If Bob wants to have an audit compatible address where all incoming transaction are linkable, he can either publish his tracking key or use a **truncated address**. That address represent only one public ec-key B , and the remaining part required by the protocol is derived from it as follows: $a = \mathcal{H}_s(B)$ and $A = \mathcal{H}_s(B)G$. In both cases every person is able to “recognize” all of Bob’s incoming transaction, but, of course, none can spend the funds enclosed within them without the secret key b .

4.4 One-time ring signatures

A protocol based on one-time ring signatures allows users to achieve unconditional unlinkability. Unfortunately, ordinary types of cryptographic signatures permit to trace transactions to their respective senders and receivers. Our solution to this deficiency lies in using a different signature type than those currently used in electronic cash systems.

We will first provide a general description of our algorithm with no explicit reference to electronic cash.

A one-time ring signature contains four algorithms: (**GEN**, **SIG**, **VER**, **LNK**):

GEN: takes public parameters and outputs an ec-pair (P, x) and a public key I .

SIG: takes a message m , a set \mathcal{S}' of public keys $\{P_i\}_{i \neq s}$, a pair (P_s, x_s) and outputs a signature σ and a set $\mathcal{S} = \mathcal{S}' \cup \{P_s\}$.

VER: takes a message m , a set \mathcal{S} , a signature σ and outputs “true” or “false”.

LNK: takes a set $\mathcal{I} = \{I_i\}$, a signature σ and outputs “linked” or “indep”.

The idea behind the protocol is fairly simple: a user produces a signature which can be checked by a set of public keys rather than a unique public key. The identity of the signer is indistinguishable from the other users whose public keys are in the set until the owner produces a second signature using the same keypair.

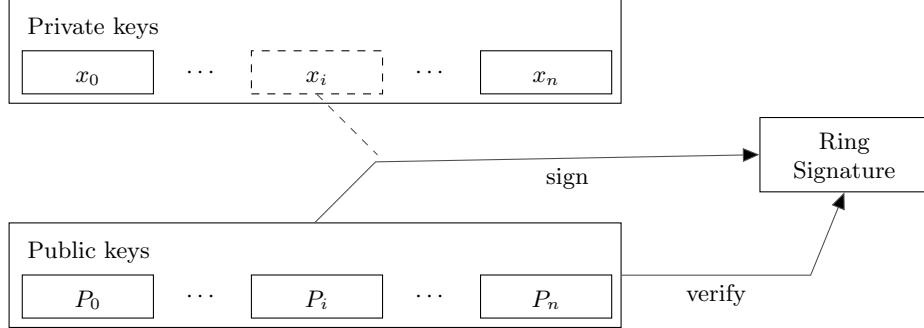


Fig. 6. Ring signature anonymity.

GEN: The signer picks a random secret key $x \in [1, l - 1]$ and computes the corresponding public key $P = xG$. Additionally he computes another public key $I = x\mathcal{H}_p(P)$ which we will call the “key image”.

SIG: The signer generates a one-time ring signature with a non-interactive zero-knowledge proof using the techniques from [21]. He selects a random subset \mathcal{S}' of n from the other users’ public keys P_i , his own keypair (x, P) and key image I . Let $0 \leq s \leq n$ be signer’s secret index in \mathcal{S} (so that his public key is P_s).

He picks a random $\{q_i \mid i = 0 \dots n\}$ and $\{w_i \mid i = 0 \dots n, i \neq s\}$ from $(1 \dots l)$ and applies the following *transformations*:

$$L_i = \begin{cases} q_i G, & \text{if } i = s \\ q_i G + w_i P_i, & \text{if } i \neq s \end{cases}$$

$$R_i = \begin{cases} q_i \mathcal{H}_p(P_i), & \text{if } i = s \\ q_i \mathcal{H}_p(P_i) + w_i I, & \text{if } i \neq s \end{cases}$$

The next step is getting the non-interactive *challenge*:

$$c = \mathcal{H}_s(m, L_1, \dots, L_n, R_1, \dots, R_n)$$

Finally the signer computes the *response*:

$$c_i = \begin{cases} w_i, & \text{if } i \neq s \\ c - \sum_{i=0}^n c_i \mod l, & \text{if } i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & \text{if } i \neq s \\ q_s - c_s x \mod l, & \text{if } i = s \end{cases}$$

The resulting signature is $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$.

VER: The verifier checks the signature by applying the inverse transformations:

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \end{cases}$$

Finally, the verifier checks if $\sum_{i=0}^n c_i \stackrel{?}{=} \mathcal{H}_s(m, L'_0, \dots, L'_n, R'_0, \dots, R'_n) \pmod{l}$

If this equality is correct, the verifier runs the algorithm **LNK**. Otherwise the verifier rejects the signature.

LNK: The verifier checks if I has been used in past signatures (these values are stored in the set \mathcal{I}). Multiple uses imply that two signatures were produced under the same secret key.

The meaning of the protocol: by applying L -transformations the signer proves that he knows such x that at least one $P_i = xG$. To make this proof non-repeatable we introduce the key image as $I = x\mathcal{H}_p(P)$. The signer uses the same coefficients (r_i, c_i) to prove almost the same statement: he knows such x that at least one $\mathcal{H}_p(P_i) = I \cdot x^{-1}$.

If the mapping $x \rightarrow I$ is an injection:

1. Nobody can recover the public key from the key image and identify the signer;
2. The signer cannot make two signatures with different I 's and the same x .

A full security analysis is provided in Appendix A.

4.5 Standard CryptoNote transaction

By combining both methods (unlinkable public keys and untraceable ring signature) Bob achieves new level of privacy in comparison with the original Bitcoin scheme. It requires him to store only one private key (a, b) and publish (A, B) to start receiving and sending anonymous transactions.

While validating each transaction Bob additionally performs only two elliptic curve multiplications and one addition per output to check if a transaction belongs to him. For his every output Bob recovers a one-time keypair (p_i, P_i) and stores it in his wallet. Any inputs can be *circumstantially proved* to have the same owner only if they appear in a single transaction. In fact this relationship is much harder to establish due to the one-time ring signature.

With a ring signature Bob can effectively hide every input among somebody else's; all possible spenders will be equiprobable, even the previous owner (Alice) has no more information than any observer.

When signing his transaction Bob specifies n foreign outputs with the same amount as his output, mixing all of them without the participation of other users. Bob himself (as well as anybody else) does not know if any of these payments have been spent: an output can be used in thousands of signatures as an ambiguity factor and never as a target of hiding. The double spend check occurs in the **LNK** phase when checking against the used key images set.

Bob can choose the ambiguity degree on his own: $n = 1$ means that the probability he has spent the output is 50% probability, $n = 99$ gives 1%. The size of the resulting signature increases linearly as $O(n+1)$, so the improved anonymity costs to Bob extra transaction fees. He also can set $n = 0$ and make his ring signature to consist of only one element, however this will instantly reveal him as a spender.

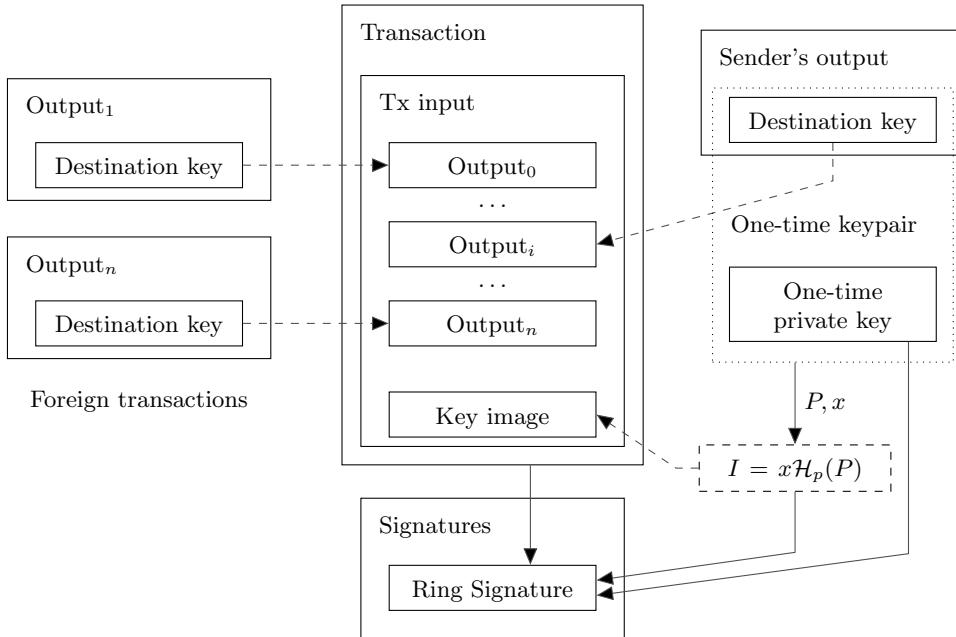


Fig. 7. Ring signature generation in a standard transaction.

5 Egalitarian Proof-of-work

In this section we propose and ground the new proof-of-work algorithm. Our primary goal is to close the gap between CPU (majority) and GPU/FPGA/ASIC (minority) miners. It is appropriate that some users can have a certain advantage over others, but their investments should grow at least linearly with the power. More generally, producing special-purpose devices has to be as less profitable as possible.

5.1 Related works

The original Bitcoin proof-of-work protocol uses the CPU-intensive pricing function SHA-256. It mainly consists of basic logical operators and relies solely on the computational speed of processor, therefore is perfectly suitable for multicore/conveyer implementation.

However, modern computers are not limited by the number of operations per second alone, but also by memory size. While some processors can be substantially faster than others [8], memory sizes are less likely to vary between machines.

Memory-bound price functions were first introduced by Abadi et al and were defined as “functions whose computation time is dominated by the time spent accessing memory” [15]. The main idea is to construct an algorithm allocating a large block of data (“scratchpad”) within memory that can be accessed relatively slowly (for example, RAM) and “accessing an unpredictable sequence of locations” within it. A block should be large enough to make preserving the data more advantageous than recomputing it for each access. The algorithm also should prevent internal parallelism, hence N simultaneous threads should require N times more memory at once.

Dwork et al [22] investigated and formalized this approach leading them to suggest another variant of the pricing function: “Mbound”. One more work belongs to F. Coelho [20], who

proposed the most effective solution: “Hokkaido”.

To our knowledge the last work based on the idea of pseudo-random searches in a big array is the algorithm known as “scrypt” by C. Percival [32]. Unlike the previous functions it focuses on key derivation, and not proof-of-work systems. Despite this fact scrypt can serve our purpose: it works well as a pricing function in the partial hash conversion problem such as SHA-256 in Bitcoin.

By now scrypt has already been applied in Litecoin [14] and some other Bitcoin forks. However, its implementation is not really memory-bound: the ratio “memory access time / overall time” is not large enough because each instance uses only 128 KB. This permits GPU miners to be roughly 10 times more effective and continues to leave the possibility of creating relatively cheap but highly-efficient mining devices.

Moreover, the scrypt construction itself allows a *linear* trade-off between memory size and CPU speed due to the fact that every block in the scratchpad is derived only from the previous. For example, you can store every second block and recalculate the others in a lazy way, i.e. only when it becomes necessary. The pseudo-random indexes are assumed to be uniformly distributed, hence the expected value of the *additional* blocks’ recalculations is $\frac{1}{2} \cdot N$, where N is the number of iterations. The overall computation time increases less than by half because there are also time independent (constant time) operations such as preparing the scratchpad and hashing on every iteration. Saving $2/3$ of the memory costs $\frac{1}{3} \cdot N + \frac{1}{3} \cdot 2 \cdot N = N$ additional recalculations; $9/10$ results in $\frac{1}{10} \cdot N + \dots + \frac{1}{10} \cdot 9 \cdot N = 4.5N$. It is easy to show that storing only $\frac{1}{s}$ of all blocks increases the time less than by a factor of $\frac{s-1}{2}$. This in turn implies that a machine with a CPU 200 times faster than the modern chips can store only 320 bytes of the scratchpad.

5.2 The proposed algorithm

We propose a new memory-bound algorithm for the proof-of-work pricing function. It relies on random access to a slow memory and emphasizes latency dependence. As opposed to scrypt every new block (64 bytes in length) depends on *all* the previous blocks. As a result a hypothetical “memory-saver” should increase his calculation speed exponentially.

Our algorithm requires about 2 Mb per instance for the following reasons:

1. It fits in the L3 cache (per core) of modern processors, which should become mainstream in a few years;
2. A megabyte of internal memory is an almost unacceptable size for a modern ASIC pipeline;
3. GPUs may run hundreds of concurrent instances, but they are limited in other ways: GDDR5 memory is slower than the CPU L3 cache and remarkable for its bandwidth, not random access speed.
4. Significant expansion of the scratchpad would require an increase in iterations, which in turn implies an overall time increase. “Heavy” calls in a trust-less p2p network may lead to serious vulnerabilities, because nodes are obliged to check every new block’s proof-of-work. If a node spends a considerable amount of time on each hash evaluation, it can be easily DDoSed by a flood of fake objects with arbitrary work data (nonce values).

6 Further advantages

6.1 Smooth emission

The upper bound for the overall amount of CryptoNote digital coins is: $\text{MSupply} = 2^{64} - 1$ atomic units. This is a natural restriction based only on implementation limits, not on intuition such as “ N coins ought to be enough for anybody”.

To ensure the smoothness of the emission process we use the following formula for block rewards:

$$\text{BaseReward} = (\text{MSupply} - A) \gg 18,$$

where A is amount of previously generated coins.

6.2 Adjustable parameters

6.2.1 Difficulty

CryptoNote contains a targeting algorithm which changes the difficulty of every block. This decreases the system’s reaction time when the network hashrate is intensely growing or shrinking, preserving a constant block rate. The original Bitcoin method calculates the relation of actual and target time-span between the last 2016 blocks and uses it as the multiplier for the current difficulty. Obviously this is unsuitable for rapid recalculations (because of large inertia) and results in oscillations.

The general idea behind our algorithm is to sum all the work completed by the nodes and divide it by the time they have spent. The measure of work is the corresponding difficulty values in each block. But due to inaccurate and untrusted timestamps we cannot determine the exact time interval between blocks. A user can shift his timestamp into the future and the next time intervals might be improbably small or even negative. Presumably there will be few incidents of this kind, so we can just sort the timestamps and cut-off the outliers (i.e. 20%). The range of the rest values is the time which was spent for 80% of the corresponding blocks.

6.2.2 Size limits

Users pay for storing the blockchain and shall be entitled to vote for its size. Every miner deals with the trade-off between balancing the costs and profit from the fees and sets his own “soft-limit” for creating blocks. Also the core rule for the maximum block size is necessary for preventing the blockchain from being flooded with bogus transaction, however this value should not be hard-coded.

Let M_N be the median value of the last N blocks sizes. Then the “hard-limit” for the size of accepting blocks is $2 \cdot M_N$. It averts the blockchain from bloating but still allows the limit to slowly grow with time if necessary.

Transaction size does not need to be limited explicitly. It is bounded by the size of a block; and if somebody wants to create a huge transaction with hundreds of inputs/outputs (or with the high ambiguity degree in ring signatures), he can do so by paying sufficient fee.

6.2.3 Excess size penalty

A miner still has the ability to stuff a block full of his own zero-fee transactions up to its maximum size $2 \cdot M_b$. Even though only the majority of miners can shift the median value, there is still a

possibility to bloat the blockchain and produce an additional load on the nodes. To discourage malevolent participants from creating large blocks we introduce a penalty function:

$$NewReward = BaseReward \cdot \left(\frac{BlkSize}{M_N} - 1 \right)^2$$

This rule is applied only when $BlkSize$ is greater than minimal free block size which should be close to $\max(10\text{kb}, M_N \cdot 110\%)$. Miners are permitted to create blocks of “usual size” and even exceed it with profit when the overall fees surpass the penalty. But fees are unlikely to grow quadratically unlike the penalty value so there will be an equilibrium.

6.3 Transaction scripts

CryptoNote has a very minimalist scripting subsystem. A sender specifies an expression $\Phi = f(x_1, x_2, \dots, x_n)$, where n is the number of destination public keys $\{P_i\}_{i=1}^n$. Only five binary operators are supported: `min`, `max`, `sum`, `mul` and `cmp`. When the receiver spends this payment, he produces $0 \leq k \leq n$ signatures and passes them to transaction input. The verification process simply evaluates Φ with $x_i = 1$ to check for a valid signature for the public key P_i , and $x_i = 0$. A verifier accepts the proof iff $\Phi > 0$.

Despite its simplicity this approach covers every possible case:

- **Multi-/Threshold signature.** For the Bitcoin-style “M-out-of-N” multi-signature (i.e. the receiver should provide at least $0 \leq M \leq N$ valid signatures) $\Phi = x_1 + x_2 + \dots + x_N \geq M$ (for clarity we are using common algebraic notation). The weighted threshold signature (some keys can be more important than other) could be expressed as $\Phi = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_N \cdot x_N \geq w_M$. And scenario where the master-key corresponds to $\Phi = \max(M \cdot x, x_1 + x_2 + \dots + x_N) \geq M$. It is easy to show that any sophisticated case can be expressed with these operators, i.e. they form basis.
- **Password protection.** Possession of a secret password s is equivalent to the knowledge of a private key, deterministically derived from the password: $k = \text{KDF}(s)$. Hence, a receiver can prove that he knows the password by providing another signature under the key k . The sender simply adds the corresponding public key to his own output. Note that this method is much more secure than the “transaction puzzle” used in Bitcoin [13], where the password is explicitly passed in the inputs.
- **Degenerate cases.** $\Phi = 1$ means that anybody can spend the money; $\Phi = 0$ marks the output as not spendable forever.

In the case when the output script combined with public keys is too large for a sender, he can use special output type, which indicates that the recipient will put this data in his input while the sender provides only a hash of it. This approach is similar to Bitcoin’s “pay-to-hash” feature, but instead of adding new script commands we handle this case at the data structure level.

7 Conclusion

We have investigated the major flaws in Bitcoin and proposed some possible solutions. These advantageous features and our ongoing development make new electronic cash system CryptoNote a serious rival to Bitcoin, outclassing all its forks.

Nobel prize laureate Friedrich Hayek in his famous work proves that the existence of concurrent independent currencies has a huge positive effect. Each currency issuer (or developer in our case) is trying to attract users by improving his product. Currency is like a commodity: it can have unique benefits and shortcomings and the most convenient and trusted currency has the greatest demand. Suppose we had a currency excelling Bitcoin: it means that Bitcoin would develop faster and become better. The biggest support as an open source project would come from its own users, who are interested in it.

We do not consider CryptoNote as a full replacement to Bitcoin. On the contrary, having two (or more) strong and convenient currencies is better than having just one. Running two and more different projects in parallel is the natural flow of electronic cash economics.

A Security

We shall give a proof for our one-time ring signature scheme. At some point it coincides with the parts of the proof in [24], but we decided to rewrite them with a reference rather than to force a reader to rush about from one paper to another.

These are the properties to be established:

- **Linkability.** Given all the secret keys $\{x_i\}_{i=1}^n$ for a set \mathcal{S} it is impossible to produce $n+1$ valid signatures $\sigma_1, \sigma_2, \dots, \sigma_{n+1}$, such that all of them pass the **LNK** phase (i.e. with $n+1$ different key images I_i). This property implies the double spending protection in the context of CryptoNote.
- **Exculpability.** Given set \mathcal{S} , at most $n-1$ corresponding private keys x_i (excluding $i=j$) and the image I_j of the keys x_j it is impossible to produce a valid signature σ with I_j . This property implies theft protection in the context of CryptoNote.
- **Unforgeability.** Given only a public keys set \mathcal{S} it is impossible to produce a valid signature σ .
- **Anonymity.** Given a signature σ and the corresponding set \mathcal{S} it is impossible to determine the secret index j of the signer with a probability $p > \frac{1}{n}$.

Linkability

Theorem 1. *Our one-time ring signature scheme is linkable under the random oracle model.*

Proof. Suppose an adversary can produce $n+1$ valid signatures σ_i with key images $I_i \neq I_j$ for any $i, j \in [1 \dots n]$. Since $\#\mathcal{S} = n$, at least one $I_i \neq x_i \mathcal{H}_p(P_i)$ for every i . Consider the corresponding signature $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$. $\mathbf{VER}(\sigma) = \text{"true"}$, this means that

$$\begin{cases} L'_i = r_i G + c_i P_i \\ R'_i = r_i \mathcal{H}_p(P_i) + c_i I \\ \sum_{i=1}^n c_i = \mathcal{H}_s(m, L'_1, \dots, L'_n, R'_1, \dots, R'_n) \mod l \end{cases}$$

The first two equalities imply

$$\begin{cases} \log_G L'_i = r_i + c_i x_i \\ \log_{\mathcal{H}_p(P_i)} R'_i = r_i + c_i \log_{\mathcal{H}_p(P_i)} I \end{cases}$$

where $\log_A B$ informally denotes the discrete logarithm of B to the base A .

As in [24] we note that $\nexists i : x_i = \log_{\mathcal{H}_p(P_i)} I$ implies that all c_i 's are uniquely determined. The third equality forces the adversary to find a pre-image of \mathcal{H}_s to succeed in the attack, an event whose probability is considered to be negligible. \square

Exculpability

Theorem 2. *Our one-time ring signature scheme is exculpable under the discrete logarithm assumption in the random oracle model.*

Proof. Suppose an adversary can produce a valid signature $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$ with $I = x_j \mathcal{H}_P(P_j)$ with given $\{x_i \mid i = 1, \dots, j-1, j+1, \dots, n\}$. Then, we can construct an algorithm \mathcal{A} which solves the discrete logarithm problem in $E(\mathbb{F}_q)$.

Suppose $\text{inst} = (G, P) \in E(\mathbb{F}_q)$ is a given instance of the DLP and the goal is to get s , such that $P = sG$. Using the standard technique (as in [24]), \mathcal{A} simulates the random and signing oracles and makes the adversary produce two valid signatures with $P_j = P$ in the set \mathcal{S} : $\sigma = (I, c_1, \dots, c_n, r_1, \dots, r_n)$ and $\sigma' = (I, c'_1, \dots, c'_n, r'_1, \dots, r'_n)$.

Since $I = x_j \mathcal{H}_p(P_j)$ in both signatures we compute $x_j = \log_{\mathcal{H}_p(P_j)} I = \frac{r_j - r'_j}{c'_j - c_j} \pmod{l}$

\mathcal{A} outputs x_j because $L_j = r_j G + c_j P_j = r'_j G + c'_j P_j$ and $P_j = P$. \square

Unforgeability

It has been shown in [24] that unforgeability is just an implication of both linkability and exculpability.

Theorem 3. *If a one-time ring signature scheme is linkable and exculpable, then it is unforgeable.*

Proof. Suppose an adversary can forge a signature for a given set \mathcal{S} : $\sigma_0 = (I_0, \dots)$. Consider all valid signatures (produced by the honest signers) for the same message m and the set \mathcal{S} : $\sigma_1, \sigma_2, \dots, \sigma_n$. There are two possible cases:

1. $I_0 \in \{I_i\}_{i=1}^n$. Which contradicts exculpability.
2. $I_0 \notin \{I_i\}_{i=1}^n$. Which contradicts linkability. \square

Anonymity

Theorem 4. *Our one-time ring signature scheme is anonymous under the decisional Diffie-Hellman assumption in the random oracle model.*

Proof. Suppose an adversary can determine the secret index j of the Signer with a probability $p = \frac{1}{n} + \epsilon$. Then, we can construct algorithm \mathcal{A} which solves the decisional Diffie-Hellman problem in $E(\mathbb{F}_q)$ with the probability $\frac{1}{2} + \frac{\epsilon}{2}$.

Let $\text{inst} = (G_1, G_2, Q_1, Q_2) \in E(\mathbb{F}_q)$ be the instance of DDH and the goal to determine if $\log_{G_1} Q_1 = \log_{G_2} Q_2$. \mathcal{A} feeds the adversary with valid signature $\sigma_0 = (I, \dots)$, where $P_j = x_j G_1 = Q_1$ and $I = Q_2$ and simulates oracle \mathcal{H}_p , returning G_2 for query $\mathcal{H}_p(P_j)$.

The adversary returns k as his guess for the index i : $I = x_i \mathcal{H}_P(P_i)$. If $k = j$, then \mathcal{A} returns 1 (for “yes”) otherwise a random $r \in \{1, 0\}$. The probability of the right choice is computed as in [24]: $\frac{1}{2} + \Pr(1 \mid \text{inst} \in \text{DDH}) - \Pr(1 \mid \text{inst} \notin \text{DDH}) = \frac{1}{2} + \Pr(k = j \mid \text{inst} \in \text{DDH}) + \Pr(k \neq j \mid \text{inst} \in \text{DDH}) \cdot \Pr(r = 1) - \Pr(k = j \mid \text{inst} \notin \text{DDH}) - \Pr(k \neq j \mid \text{inst} \notin \text{DDH}) \cdot \Pr(r = 0) = \frac{1}{2} + \frac{1}{n} + \epsilon + (\frac{n-1}{n} - \epsilon) \cdot \frac{1}{2} - \frac{1}{n} - \frac{n-1}{n} \cdot \frac{1}{2} = \frac{1}{2} + \frac{\epsilon}{2}$

In fact, the result should be reduced by the probability of collision in \mathcal{H}_s , but this value is considered to be negligible. \square

Notes on the hash function \mathcal{H}_p

We defined \mathcal{H}_p as deterministic hash function $E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q)$. None of the proofs demands \mathcal{H}_p to be an ideal cryptographic hash function. It’s main purpose is to get a pseudo-random base for image key $I = x \mathcal{H}_p(xG)$ in some determined way.

With fixed base ($I = xG_2$) the following scenario is possible:

1. Alice sends two standard transactions to Bob, generating one-time tx-keys: $P_2 = \mathcal{H}_s(r_1A)G + B$ and $P_1 = \mathcal{H}_s(r_2A)G + B$.
2. Bob recovers corresponding one-time private tx-keys x_1 and x_2 and spends the outputs with valid signatures and images keys $I_1 = x_1G_2$ and $I_2 = x_2G_2$.
3. Now Alice can link these signatures, checking the equality $I_1 - I_2 \stackrel{?}{=} (\mathcal{H}_s(r_1A) - \mathcal{H}_s(r_2A))G_2$.

The problem is that Alice knows the linear correlation between public keys P_1 and P_2 and in case of fixed base G_2 she also gets the same correlation between key images I_1 and I_2 . Replacing G_2 with $\mathcal{H}_p(xG_2)$, which does not preserve linearity, fixes that flaw.

For constructing deterministic \mathcal{H}_p we use algorithm presented in [37].

References

- [1] <http://bitcoin.org>.
- [2] https://en.bitcoin.it/wiki/Category:Mixing_Services.
- [3] <http://blog.ezyang.com/2012/07/secure-multiparty-bitcoin-anonymization>.
- [4] <https://bitcointalk.org/index.php?topic=279249.0>.
- [5] <http://msrvideo.vo.msecnd.net/rmcvideos/192058/dl/192058.pdf>.
- [6] <https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki#Specification>.
- [7] https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki#Backwards_Compatibility.
- [8] https://en.bitcoin.it/wiki/Mining_hardware_comparison.
- [9] <https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki>.
- [10] <http://luke.dashjr.org/programs/bitcoin/files/charts/branches.html>.
- [11] <https://bitcointalk.org/index.php?topic=196259.0>.
- [12] <https://en.bitcoin.it/wiki/Contracts>.
- [13] <https://en.bitcoin.it/wiki/Script>.
- [14] <http://litecoin.org>.
- [15] Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard, memory-bound functions. In *NDSS*, 2003.
- [16] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Ad-hoc-group signatures from hijacked keypairs. In *in DIMACS Workshop on Theft in E-Commerce*, 2005.
- [17] Man Ho Au, Sherman S. M. Chow, Willy Susilo, and Patrick P. Tsang. Short linkable ring signatures revisited. In *EuroPKI*, pages 101–115, 2006.
- [18] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89, 2012.
- [19] David Chaum and Eugène van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.
- [20] Fabien Coelho. Exponential memory-bound functions for proof of work protocols. *IACR Cryptology ePrint Archive*, 2005:356, 2005.
- [21] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.
- [22] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *CRYPTO*, pages 426–444, 2003.
- [23] Eiichiro Fujisaki. Sub-linear size traceable ring signatures without random oracles. In *CT-RSA*, pages 393–415, 2011.

- [24] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *Public Key Cryptography*, pages 181–200, 2007.
- [25] Jezz Garzik. Peer review of “quantitative analysis of the full bitcoin transaction graph”. <https://gist.github.com/3901921>, 2012.
- [26] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In *ACISP*, pages 325–335, 2004.
- [27] Joseph K. Liu and Duncan S. Wong. Linkable ring signatures: Security models and new schemes. In *ICCSA (2)*, pages 614–623, 2005.
- [28] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Symposium on Security and Privacy*, pages 397–411, 2013.
- [29] Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future internet*, 5(2):237–250, 2013.
- [30] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *CRYPTO*, pages 324–337, 1991.
- [31] Marc Santamaria Ortega. The bitcoin transaction graph — anonymity. Master’s thesis, Universitat Oberta de Catalunya, June 2013.
- [32] Colin Percival. Stronger key derivation via sequential memory-hard functions. Presented at BSDCan’09, May 2009.
- [33] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. *CoRR*, abs/1107.4524, 2011.
- [34] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001.
- [35] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. *IACR Cryptology ePrint Archive*, 2012:584, 2012.
- [36] Meni Rosenfeld. Analysis of hashrate-based double-spending. 2012.
- [37] Maciej Ulas. Rational points on certain hyperelliptic curves over finite fields. *Bulletin of the Polish Academy of Sciences. Mathematics*, 55(2):97–104, 2007.
- [38] Qianhong Wu, Willy Susilo, Yi Mu, and Fangguo Zhang. Ad hoc group signatures. In *IWSEC*, pages 120–135, 2006.

The Tangle

Serguei Popov*

April 30, 2018. Version 1.4.3

Abstract

In this paper we analyze the mathematical foundations of IOTA, a cryptocurrency for the Internet-of-Things (IoT) industry. The main feature of this novel cryptocurrency is the *tangle*, a directed acyclic graph (DAG) for storing transactions. The tangle naturally succeeds the blockchain as its next evolutionary step, and offers features that are required to establish a machine-to-machine micropayment system.

An essential contribution of this paper is a family of Markov Chain Monte Carlo (MCMC) algorithms. These algorithms select attachment sites on the tangle for a transaction that has just arrived.

1 Introduction and description of the system

The rise and success of Bitcoin during the last six years proved that blockchain technology has real-world value. However, this technology also has a number of drawbacks that prevent it from being used as a generic platform for cryptocurrencies across the globe. One notable drawback is the concept of a transaction fee for transactions of any value. The importance of micropayments will increase in the rapidly developing IoT industry, and paying a fee that is *larger* than the amount of value being transferred is not logical. Furthermore, it is not easy to get rid of fees in the blockchain infrastructure since they serve as an incentive for the creators of blocks. This leads to another issue with existing cryptocurrency technology, namely the heterogeneous nature of the system. There are two distinct types of participants in the system, those who issue transactions, and those who approve transactions. The design of this system creates unavoidable discrimination of some participants, which in turn creates

*a.k.a. `mthcl`; author's contact information: `serguei.popov@iota.org`

conflicts that make all elements spend resources on conflict resolution. The aforementioned issues justify a search for solutions essentially different from blockchain technology, the basis for Bitcoin and many other cryptocurrencies.

In this paper we discuss an innovative approach that does not incorporate blockchain technology. This approach is currently being implemented as a cryptocurrency called *iota* [1], which was designed specifically for the IoT industry. The purpose of this paper is to focus on general features of the tangle, and to discuss problems that arise when one attempts to get rid of the blockchain and maintain a distributed ledger. The concrete implementation of the iota protocol is not discussed.

In general, a tangle-based cryptocurrency works in the following way. Instead of the global blockchain, there is a DAG that we call the *tangle*. The transactions issued by nodes constitute the site set of the tangle graph, which is the ledger for storing transactions. The edge set of the tangle is obtained in the following way: when a new transaction arrives, it must *approve* two¹ previous transactions. These approvals are represented by directed edges, as shown in Figure 1². If there is not a directed edge between transaction *A* and transaction *B*, but there is a directed path of length at least two from *A* to *B*, we say that *A* *indirectly approves* *B*. There is also the “genesis” transaction, which is approved either directly or indirectly by all other transactions (Figure 2). The genesis is described in the following way. In the beginning of the tangle, there was an address with a balance that contained all of the tokens. The genesis transaction sent these tokens to several other “founder” addresses. Let us stress that all of the tokens were created in the genesis transaction. No tokens will be created in the future, and there will be no mining in the sense that miners receive monetary rewards “out of thin air”.

A quick note on terminology: *sites* are transactions represented on the tangle graph. The network is composed of *nodes*; that is, nodes are entities that issue and validate transactions.

The main idea of the tangle is the following: to issue a transaction, users must work to approve other transactions. Therefore, users who issue a transaction are contributing to the network’s security. It is assumed that the nodes check if the approved transactions are not conflicting. If a node finds that a transaction is in conflict with the tangle history, the node will not approve the conflicting transaction in either a direct or indirect manner³.

¹This is the simplest approach. One may also study similar systems where transactions must approve k other transactions for a general $k \geq 2$, or have an entirely different set of rules.

²Time always increases from left to right in each figure.

³If a node issues a new transaction that approves conflicting transactions, then it risks that other nodes will not approve its new transaction, which will fall into oblivion.

As a transaction receives additional approvals, it is accepted by the system with a higher level of confidence. In other words, it will be difficult to make the system accept a double-spending transaction. It is important to observe that we do not *impose* any rules for choosing which transactions a node will approve. Instead, we argue that if a large number of nodes follow some “reference” rule, then for any fixed node it is better to stick to a rule of the same kind⁴. This seems to be a reasonable assumption, especially in the context of IoT, where nodes are specialized chips with pre-installed firmware.

In order to issue a transaction, a node does the following:

- The node chooses two other transactions to approve according to an algorithm. In general, these two transactions may coincide.
- The node checks if the two transactions are not conflicting, and does not approve conflicting transactions.
- For a node to issue a valid transaction, the node must solve a cryptographic puzzle similar to those in the Bitcoin blockchain. This is achieved by finding a nonce such that the hash of that nonce concatenated with some data from the approved transaction has a particular form. In the case of the Bitcoin protocol, the hash must have at least a predefined number of leading zeros.

It is important to observe that the iota network is asynchronous. In general, nodes do not necessarily see the same set of transactions. It should also be noted that the tangle may contain conflicting transactions. The nodes do not have to achieve consensus on which valid⁵ transactions have the right to be in the ledger, meaning all of them can be in the tangle. However, in the case where there are conflicting transactions, the nodes need to decide which transactions will become orphaned⁶. The main rule that the nodes use for deciding between two conflicting transactions is the following: a node runs the tip selection algorithm⁷ (cf. Section 4.1) many times, and sees which of the two transactions is more likely to be indirectly approved by the selected tip. For example, if a transaction was selected 97 times during 100 runs of the tip selection algorithm, we say that it is confirmed with 97% confidence.

Let us also comment on the following question (cf. [4]): what motivates the nodes to propagate transactions? Every node calculates some statistics, one of which is

⁴We comment more on this at the end of Section 4.1

⁵Transactions that are issued according to the protocol.

⁶Orphaned transactions are not indirectly approved by incoming transactions anymore

⁷As mentioned above, there is a good reason to assume that other nodes would follow the same algorithm for tip selection.

how many new transactions are received from a neighbor. If one particular node is “too lazy”, it will be dropped by its neighbors. Therefore, even if a node does not issue transactions, and hence has no direct incentive to share new transactions that approve its own transaction, it still has incentive to participate.

After introducing some notation in Section 2, we discuss algorithms for choosing the two transactions to approve, the rules for measuring the transaction’s overall approval (Section 3, especially Section 3.1), and possible attack scenarios (Section 4). Also, in the unlikely event that the reader is scared by the formulas, they can jump directly to the “conclusions” at the end of each section.

It should be noted that the idea of using DAGs in the cryptocurrency space has been around for some time, see [3, 6, 7, 9, 12]. Specifically, [7] introduces the GHOST protocol, which proposes a modification of the Bitcoin protocol by making the main ledger a tree instead of a blockchain. It is shown that such a modification reduces confirmation times and improves the overall security of the network. In [9] the authors consider a DAG-based cryptocurrency model. Their model is different than our model for the following reasons: the sites of their DAG are blocks instead of individual transactions; the miners in their system compete for transaction fees; and new tokens may be created by block miners. Also, observe that a solution somewhat similar to ours was proposed in [6], although it does not discuss any particular tip approval strategies. After the first version of this paper was published, several other works that aim to create a DAG-based distributed ledger have appeared, e.g. [8]. We also reference another approach [2, 10] that aims to make Bitcoin micropayments possible by establishing peer-to-peer payment channels.

2 Weights and more

In this section we define the weight of a transaction, and related concepts. The weight of a transaction is proportional to the amount of work that the issuing node invested into it. In the current implementation of iota, the weight may only assume values 3^n , where n is a positive integer that belongs to some nonempty interval of acceptable values⁸. In fact, it is irrelevant to know how the weight was obtained in practice. It is only important that every transaction has a positive integer, its weight, attached to it. In general, the idea is that a transaction with a larger weight is more “important” than a transaction with a smaller weight. To avoid spamming and other attack styles, it is assumed that no entity can generate an abundance of transactions with “acceptable” weights in a short period of time.

⁸This interval should also be finite — see the “large weight attack” in Section 4.

One of the notions we need is the *cumulative weight* of a transaction: it is defined as the own weight of a particular transaction plus the sum of own weights of all transactions that directly or indirectly approve this transaction. The algorithm for cumulative weight calculation is illustrated in Figure 1. The boxes represent transactions, the small number in the SE corner of each box denotes own weight, and the bold number denotes the cumulative weight. For example, transaction F is directly or indirectly approved by transactions A, B, C, E . The cumulative weight of F is $9 = 3 + 1 + 3 + 1 + 1$, which is the sum of the own weight of F and the own weights of A, B, C, E .

Let us define “tips” as unapproved transactions in the tangle graph. In the top tangle snapshot of Figure 1, the only tips are A and C . When the new transaction X arrives and approves A and C in the bottom tangle snapshot, X becomes the only tip. The cumulative weight of all other transactions increases by 3, the own weight of X .

We need to introduce two additional variables for the discussion of approval algorithms. First, for a transaction site on the tangle, we introduce its

- *height*: the length of the longest oriented path to the genesis;
- *depth*: the length of the longest reverse-oriented path to some tip.

For example, G has height 1 and depth 4 in Figure 2 because of the reverse path F, D, B, A , while D has height 3 and depth 2. Also, let us introduce the notion of the *score*. By definition, the score of a transaction is the sum of own weights of all transactions approved by this transaction plus the own weight of the transaction itself. In Figure 2, the only tips are A and C . Transaction A directly or indirectly approves transactions B, D, F, G , so the score of A is $1+3+1+3+1 = 9$. Analogously, the score of C is $1+1+1+3+1 = 7$.

In order to understand the arguments presented in this paper, one may safely assume that all transactions have an own weight equal to 1. *From now on, we stick to this assumption.* Under this assumption, the cumulative weight of transaction X becomes 1 plus the number of transactions that directly or indirectly approve X , and the score becomes 1 plus the number of transactions that are directly or indirectly approved by X .

Let us note that, among those defined in this section, the cumulative weight is (by far!) the most important metric, although height, depth, and score will briefly enter some discussions as well.

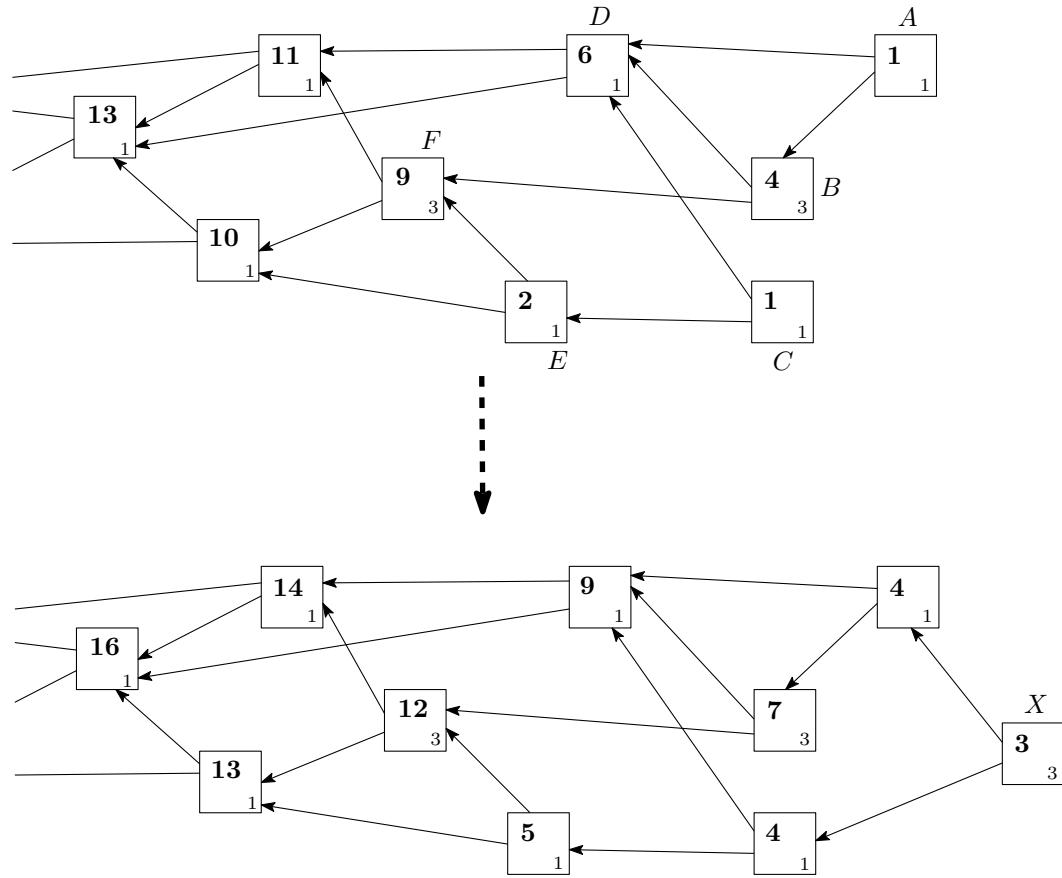


Figure 1: DAG with weight assignments before and after a newly issued transaction, X . The boxes represent transactions, the small number in the SE corner of each box denotes own weight, and the bold number denotes the cumulative weight.

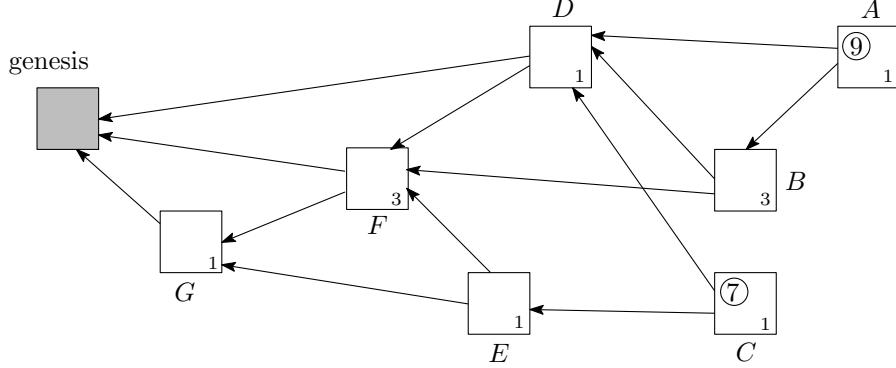


Figure 2: DAG with own weights assigned to each site, and scores calculated for sites A and C .

3 Stability of the system, and cutsets

Let $L(t)$ be the total number of tips in the system at time t . One expects that the stochastic process $L(t)$ remains *stable*⁹. More precisely, one expects the process to be *positive recurrent*, see Sections 4.4 and 6.5 of [11] for formal definitions. In particular, positive recurrence implies that the limit of $\mathbb{P}[L(t) = k]$ as $t \rightarrow \infty$ should exist and be positive for all $k \geq 1$. Intuitively, we expect that $L(t)$ should fluctuate around a constant value, and not escape to infinity. If $L(t)$ were to escape to infinity, many unapproved transactions would be left behind.

To analyze the stability properties of $L(t)$, we need to make some assumptions. One assumption is that transactions are issued by a large number of roughly independent entities, so the process of incoming transactions can be modeled by a Poisson point process (cf. e.g. Section 5.3 of [11]). Let λ be the rate of that Poisson process. For simplicity, let us assume that this rate remains constant in time. Assume that all devices have approximately the same computing power, and let h be the average time a device needs to perform calculations that are required to issue a transaction. Then, let us *assume* that all nodes behave in the following way: to issue a transaction, a node chooses two tips at random and approves them. It should be observed that, in general, it is *not* a good idea for the “honest nodes” to adopt this strategy because it has a number of practical disadvantages. In particular, it does not offer enough protection against “lazy” or malicious nodes (see Section 4.1 below). On the other hand, we still consider this model since it is simple to analyze, and may provide insight into the system’s behavior for more complicated tip selection strategies.

⁹Under an additional assumption that the process is time-homogeneous.

Next, we make a further simplifying assumption that any node, at the moment when it issues a transaction, observes not the actual state of the tangle, but the one exactly h time units ago. This means, in particular, that a transaction attached to the tangle at time t only becomes visible to the network at time $t+h$. We also assume that the number of tips remains roughly stationary in time, and is concentrated around a number $L_0 > 0$. In the following, we will calculate L_0 as a function of λ and h .

Observe that, at a given time t we have roughly λh “hidden tips” (which were attached in the time interval $[t-h, t)$ and so are not yet visible to the network); also, assume that typically there are r “revealed tips” (which were attached before time $t-h$ and remain tips at time t), so $L_0 = r + \lambda h$. By stationarity, we may then assume that at time t there are also around λh sites that were tips at time $t-h$, but are not tips anymore. Now, think about a new transaction that comes at this moment; then, a transaction it chooses to approve is a tip with probability $r/(r + \lambda h)$ (since there are around r tips known to the node that issued the transaction, and there are also around λh transactions which are not tips anymore, although that node thinks they are), so the mean number of chosen tips is $2r/(r + \lambda h)$. The key observation is now that, in the stationary regime, this mean number of chosen tips should be equal to 1, since, in average, a newcomer transaction should not change the number of tips. Solving the equation $2r/(r + \lambda h) = 1$ with respect to r , we obtain $r = \lambda h$, and so

$$L_0 = 2\lambda h. \quad (1)$$

We also note that, if the rule is that a new transaction references k transactions instead of 2, then a similar calculation gives

$$L_0^{(k)} = \frac{k\lambda h}{k-1}. \quad (2)$$

This is, of course, consistent with the fact that $L_0^{(k)}$ should tend to λh as $k \rightarrow \infty$ (basically, the only tips would be those still unknown to the network).

Also (we return to the case of two transactions to approve) the expected time for a transaction to be approved for the first time is approximately $h + L_0/2\lambda = 2h$. This is because, by our assumption, during the first h units of time a transaction cannot be approved, and after that the Poisson flow of approvals to it has rate approximately $2\lambda/L_0$. (Recall Proposition 5.3 of [11], which says that if we independently classify each event of a Poisson process according to a list of possible subtypes, then the processes of events of each subtype are independent Poisson processes.)

Observe that¹⁰ at any fixed time t the set of transactions that were tips at some

¹⁰At least in the case where the nodes *try* to approve tips.

moment $s \in [t, t + h(L_0, N)]$ typically constitutes a *cutset*. Any path from a transaction issued at time $t' > t$ to the genesis must pass through this set. It is important that the size of a new cutset in the tangle occasionally becomes small. One may then use the small cutsets as checkpoints for possible DAG pruning and other tasks.

It is important to observe that the above “purely random” approval strategy is not very good in practice because it does not encourage approving tips. A “lazy” user could always approve a fixed pair of very old transactions, therefore not contributing to the approval of more recent transactions, without being punished for such behavior¹¹. Also, a malicious entity can artificially inflate the number of tips by issuing many transactions that approve a fixed pair of transactions. This would make it possible for future transactions to select these tips with very high probability, effectively abandoning the tips belonging to “honest” nodes. To avoid issues of this sort, one has to adopt a strategy that is biased towards the “better” tips. One example of such a strategy is presented in Section 4.1 below.

Before starting the discussion about the expected time for a transaction to receive its first approval, note that we can distinguish two regimes (Figure 3).

- Low load: the typical number of tips is small, and frequently becomes 1. This may happen when the flow of transactions is so small that it is not probable that several different transactions approve the same tip. Also, if the network latency is very low and devices compute fast, it is unlikely that many tips would appear. This even holds true in the case when the flow of transactions is reasonably large. Moreover, we have to assume that there are no attackers that try to artificially inflate the number of tips.
- High load: the typical number of tips is large. This may happen when the flow of transactions is large, and computational delays together with network latency make it likely that several different transactions approve the same tip.

This division is rather informal, and there is no clear borderline between the two regimes. Nevertheless, we find that it may be instructive to consider these two different extremes.

The situation in the low load regime is relatively simple. The first approval happens on an average timescale of order λ^{-1} since one of the first few incoming transactions will approve a given tip.

Let us now consider the high load regime, the case where L_0 is large. As mentioned above, one may assume that the Poisson flows of approvals to different tips are

¹¹We remind the reader that we do not try to *enforce* any particular tip selection strategy. An attacker can choose tips in any way they find convenient.

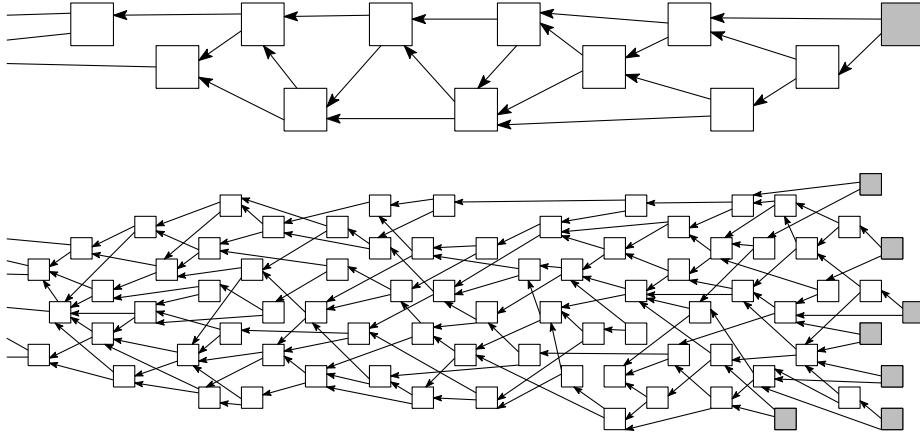


Figure 3: Low load (top) and high load (bottom) regimes of incoming transaction flow. White squares represent verified sites, while gray squares represent tips.

independent and have an approximate rate of $2\lambda/L_0$. Therefore, the expected time for a transaction to receive its first approval is around $L_0/(2\lambda) = h$ (recall (1)).

However, it is worth noting that for more elaborate approval strategies¹², it may not be a good idea to passively wait a long time until a transaction is approved by the others. This is due to the fact that “better” tips will keep appearing and will be preferred for approval. Rather, in the case when a transaction is waiting for approval over a time interval much larger than $L_0/2\lambda$, a good strategy would be to promote this latent transaction with an additional empty transaction¹³. In other words, a node can issue an empty transaction that approves its previous transaction together with one of the “better” tips to increase the probability that the empty transaction receives approval.

It turns out that the approval strategies based on heights and scores may be vulnerable to a specific type of attacks, see Section 4.1. We will discuss more elaborate strategies¹⁴ to defend against such attacks in that section. In the meantime, it is still

¹²That favor “better” quality tips in future implementations of iota.

¹³An empty transaction is a transaction that does not involve any token transfer, but still has to approve two other transactions. It should be noted that generating an empty transaction contributes to the network’s security.

¹⁴In fact, the author’s feeling is that the tip approval strategy is *the* most important ingredient for constructing a tangle-based cryptocurrency. It is there that many attack vectors are hiding. Also, since there is usually no way to *enforce* a particular tip approval strategy, it must be such that the nodes would voluntarily choose to follow it knowing that at least a good proportion of other nodes does so.

worth considering the simple tip selection strategy where an incoming transaction approves two random tips. This strategy is the easiest to analyze, and therefore may provide some insight into the qualitative and quantitative behavior of the tangle.

Conclusions:

1. We distinguish between two regimes, low load and high load (Figure 3).
2. There are only a few tips in the low load regime. A tip gets approved for the first time in $\Theta(\lambda^{-1})$ time units, where λ is the rate of the incoming flow of transactions.
3. In the high load regime the typical number of tips depends on the tip approval strategy employed by the new transaction.
4. If a transaction uses the strategy of approving two random tips, the typical number of tips is given by (1). It can be shown that this strategy is optimal with respect to the typical number of tips. However, it is not practical to adopt this strategy because it does not encourage approving tips.
5. More elaborate strategies are needed to handle attacks and other network issues. A family of such strategies is discussed in Section 4.1.
6. The typical time for a tip to be approved is $\Theta(h)$ in the high load regime, where h is the average computation/propagation time for a node. However, if the first approval does not occur in the above time interval, it is a good idea for the issuer and/or receiver to promote that transaction with an additional empty transaction.

3.1 How fast does the cumulative weight typically grow?

Assume that the network is in the low load regime. After a transaction gets approved several times, its cumulative weight will grow with speed λ because all new transactions will indirectly reference this transaction¹⁵.

In the case where the network is in the high load regime, an old transaction with a large cumulative weight will experience weight growth with speed λ because essentially all new transactions will indirectly reference it. Moreover, when the transaction

¹⁵Recall that we assumed that the own weights of all transactions are equal to 1, so the cumulative weight is just the number of transactions that directly or indirectly reference a transaction plus 1.

is first added to the tangle it may have to wait for some time to be approved. In this time interval, the transaction's cumulative weight behaves in a random fashion. To characterize the speed with which the cumulative weight grows after the transaction receives several approvals, let us define $H(t)$ as the expected cumulative weight at time t (for simplicity, we start counting time at the moment when our transaction was revealed to the network, i.e., h time units after it was created) and $K(t)$ as the expected number of tips that approve the transaction at time t . Let us also abbreviate $h := h(L_0, N)$. We make a simplifying assumption that the number of tips remains roughly constant at a value of L_0 over time. We work with the “approve two random tips” strategy in this section. It is expected that the qualitative behavior will be roughly the same for other reasonable strategies.

Recall that a transaction entering the network at time t typically chooses two tips to approve based on the state of the system at time $t - h$ because the node must do some calculations and verifications before actually issuing the transaction. It is not difficult to see that (assuming, though, that $K(\cdot)$ is the *actual* number of tips, not just expected number) the probability of the transaction approving at least one of “our” tips in the tangle is $1 - (1 - \frac{K(t-h)}{L_0})^2 = \frac{K(t-h)}{L_0}(2 - \frac{K(t-h)}{L_0})$ ¹⁶. Analogous to Example 6.4 of [11], we can write for small $\delta > 0$

$$H(t + \delta) = H(t) + \lambda\delta \frac{K(t-h)}{L_0} \left(2 - \frac{K(t-h)}{L_0}\right) + o(\delta),$$

and thus deduce the following differential equation

$$\frac{dH(t)}{dt} = \lambda \frac{K(t-h)}{L_0} \left(2 - \frac{K(t-h)}{L_0}\right). \quad (3)$$

In order to be able to use (3), we need to first calculate $K(t)$. This is not a trivial task since a tip at time $t - h$ may not be a tip at time t , and the overall number of tips approving the original transaction increases by 1 in the case where an incoming transaction approves such a tip. The crucial observation is that the probability that a tip at time $t - h$ remains a tip at time t is approximately 1/2. (To verify this, recall the discussion from Section 3: the typical number of tips is $2\lambda h$, and during the interval of length h new λh tips will substitute a half of old ones.) Therefore, at time t approximately one half $K(t-h)$ tips remain in the unconfirmed tip state, while the other half will have received at least one approval. Let A denote the set of $K(t-h)/2$ tips at time $t - h$ that are still tips at time t , and let B denote the

¹⁶The expression on the left-hand side is 1 minus the probability that the two approved tips are not ours.

remaining set of $K(t-h)/2$ tips that were already approved by time t . Let p_1 be the probability that a new transaction approves at least 1 transaction from B and does not approve any transactions from A . Furthermore, let p_2 be the probability that both approved transactions belong to A . In other words, p_1 and p_2 are the probabilities that the current number of “our” tips increases or decreases by 1 upon arrival of the new transaction. We have

$$p_1 = \left(\frac{K(t-h)}{2L_0}\right)^2 + 2 \times \frac{K(t-h)}{2L_0} \left(1 - \frac{K(t-h)}{L_0}\right),$$

$$p_2 = \left(\frac{K(t-h)}{2L_0}\right)^2.$$

To obtain the first expression, observe that p_1 equals the probability that both approved tips belong to B plus twice the probability that the first tip belongs to B and the second tip does not belong to $A \cup B$. Analogous to (3), the differential equation for $K(t)$ is:

$$\frac{dK(t)}{dt} = (p_1 - p_2)\lambda = \lambda \frac{K(t-h)}{L_0} \left(1 - \frac{K(t-h)}{L_0}\right). \quad (4)$$

It is difficult to solve (4) exactly, so we make further simplifying assumptions. First of all, we observe that after the time when $K(t)$ reaches level εL_0 for a fixed $\varepsilon > 0$, it will grow very quickly to $(1-\varepsilon)L_0$. Now, when $K(t)$ is small with respect to L_0 , we can drop the last factor in the right-hand side of (4)¹⁷. We obtain a simplified version of (4) by recalling that $\frac{\lambda h}{L_0} = \frac{1}{2}$:

$$\frac{dK(t)}{dt} \approx \frac{1}{2h} K(t-h), \quad (5)$$

with boundary condition $K(0) = 1$. We look for a solution of the form $K(t) = \exp(c\frac{t}{h})$; after substituting this into (5), we obtain

$$\frac{c}{h} \exp\left(c\frac{t}{h}\right) \approx \frac{1}{2h} \exp\left(c\frac{t}{h} - c\right),$$

therefore

$$K(t) = \exp\left(W\left(\frac{1}{2}\right)\frac{t}{h}\right) \approx \exp\left(0.352\frac{t}{h}\right) \quad (6)$$

is an approximate solution, where $W(\cdot)$ is the so-called Lambert W -function.¹⁸ Taking the logarithm of both sides in (6), we find that the time when $K(t)$ reaches εL_0 is

¹⁷It would be a constant close to 1, so the right-hand side would be equivalent to $\lambda \frac{K(t-h)}{L_0}$.

¹⁸Also known as the omega function or product logarithm; for $x \in [0, +\infty)$ it is characterized by the relation $x = W(x) \exp(W(x))$.

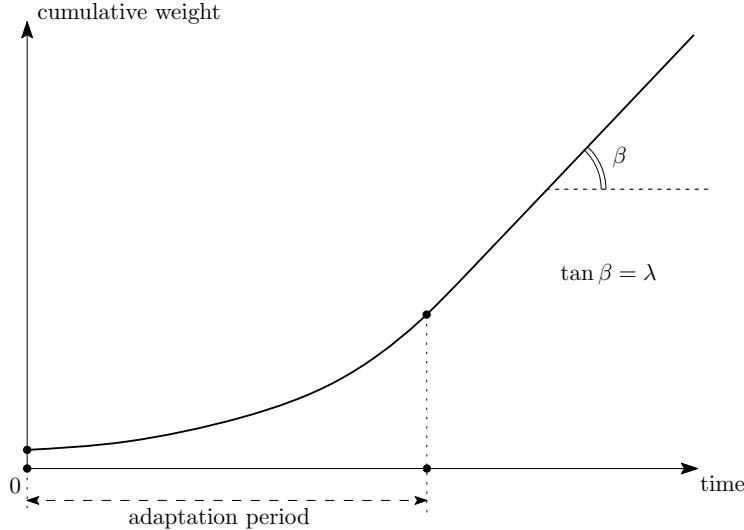


Figure 4: Plot of cumulative weight vs. time for the high load regime.

roughly

$$t_0 \approx \frac{h}{W\left(\frac{1}{2}\right)} \times (\ln L_0 - \ln \varepsilon^{-1}) \lesssim 2.84 \cdot h \ln L_0. \quad (7)$$

Returning to (3) and dropping the last term on the right-hand side, we obtain that during the “adaptation period” (i.e., $t \leq t_0$ with t_0 as in (7)), it holds that

$$\begin{aligned} \frac{dH(t)}{dt} &\approx \frac{2\lambda}{L_0} K(t-h) \\ &\approx \frac{1}{h \exp(W(\frac{1}{2}))} \exp\left(W(\frac{1}{2})\frac{t}{h}\right) \\ &= \frac{2W(\frac{1}{2})}{h} \exp\left(W(\frac{1}{2})\frac{t}{h}\right) \end{aligned}$$

and therefore

$$H(t) \approx 2 \exp\left(W(\frac{1}{2})\frac{t}{h}\right) \approx 2 \exp\left(0.352\frac{t}{h}\right). \quad (8)$$

Let us also remind the reader that after the adaptation period, the cumulative weight $H(t)$ grows linearly with speed λ . We stress that the “exponential growth” in (8) does not mean that the cumulative weight grows “very quickly” during the adaptation period. Rather, the behavior is as depicted in Figure 4.

Conclusions:

1. After a transaction gets approved multiple times in the low load regime, its cumulative weight will grow with speed λw , where w is the mean weight of a generic transaction.
2. In the high load regime, there are two distinct growth phases. First, a transaction's cumulative weight $H(t)$ grows with increasing speed during the *adaptation period* according to (8). After the adaptation period is over, the cumulative weight grows with speed λw (Figure 4). In fact, for *any* reasonable strategy the cumulative weight will grow with this speed after the end of the adaptation period because all incoming transactions will indirectly approve the transaction of interest.
3. One can think of the adaptation period of a transaction as the time until most of the current tips indirectly approve that transaction. The typical length of the adaptation period is given by (7).

4 Possible attack scenarios

We start by discussing an attack scenario where the attacker tries to “outpace” the network alone:

1. An attacker sends a payment to a merchant and receives the goods after the merchant decides the transaction has a sufficiently large cumulative weight.
2. The attacker issues a double-spending transaction.
3. The attacker uses their computing power to issue many small transactions that approve the double-spending transaction, but do not approve the original transaction that they sent to the merchant either directly or indirectly.
4. It is possible for the attacker to have a plethora of Sybil identities which are not required to approve tips.
5. An alternative method to item 3 would be for the attacker to issue a big double-spending transaction using all of their computing power. This transaction would have a very large own weight¹⁹, and would approve transactions prior to the legitimate transaction used to pay the merchant.

¹⁹Here we assume that the own weight of a transaction may vary. It will become clear in the discussion below why it is a good idea to let the own weight vary.

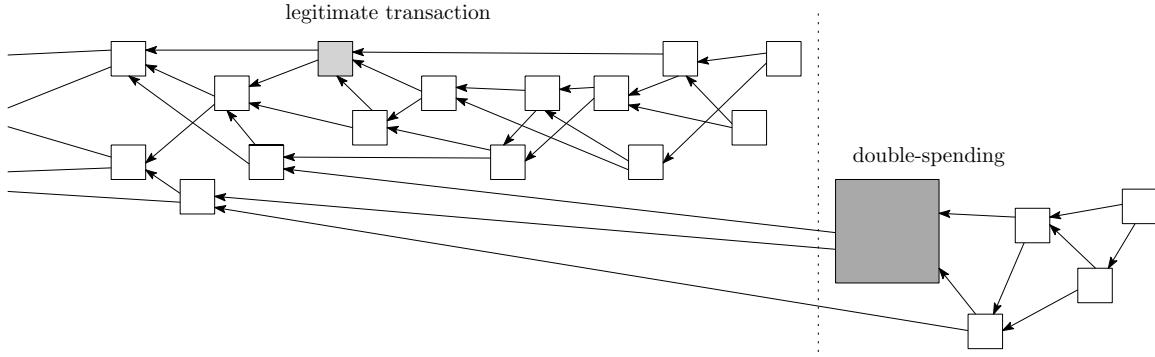


Figure 5: The “large weight” attack

6. The attacker hopes that their dishonest subtangle outpaces the honest subtangle. If this happens, the main tangle continues growing from the double-spending transaction, and the legitimate branch with the original payment to the merchant is orphaned (Figure 5).

In fact, it can be shown that the strategy of one large double-spending transaction increases the attacker’s chances of being successful. In the “ideal” situation of this mathematical model, this attack *always* succeeds.

Let $W^{(n)}$ be the time needed to obtain a nonce that gives the double-spending transaction a weight of at least 3^n . One may assume that $W^{(n)}$ is an exponentially distributed random variable with parameter²⁰ $\mu 3^{-n}$, where μ represents the computing power of the attacker.

Assume that the merchant accepts the legitimate transaction when its cumulative weight becomes at least w_0 , which happens t_0 time units after the original transaction. It is reasonable to expect that the cumulative weight grows with linear speed λw , where λ is the overall arrival rate of transactions issued on the network by honest nodes, and w is the mean weight of a generic transaction. The typical total weight of the legitimate branch at that time is $w_1 = \lambda w t_0$.

Let $\lceil x \rceil$ be the smallest integer greater than or equal to x , define $n_0 = \lceil \frac{\ln w_1}{\ln 3} \rceil$, so that $3^{n_0} \geq w_1$ ²¹. If the attacker managed to obtain a nonce that gives the double-spending transaction a weight of at least 3^{n_0} during the time interval of length t_0 , then the attack succeeds. The probability of this event is

$$\mathbb{P}[W^{(n_0)} < t_0] = 1 - \exp(-t_0 \mu 3^{-n_0}) \approx 1 - \exp(-t_0 \mu w_1^{-1}) \approx \frac{t_0 \mu}{w_1}.$$

²⁰With expectation $\mu^{-1} 3^n$.

²¹In fact, $3^{n_0} \approx w_1$ if w_1 is large.

This approximation is true in the case where $\frac{t_0\mu}{w_1}$ is small, which is a reasonable assumption. If this “immediate” attack does not succeed, the attacker may continue to look for the nonce that gives weight 3^n for $n > n_0$, and hope that at the moment they find it, the total weight of the legitimate branch is smaller than 3^n . The probability of this event occurring is

$$\mathbb{P}[\lambda w W^{(n)} < 3^n] = 1 - \exp(-\mu 3^{-n_0} \times (3^{n_0}/\lambda w)) = 1 - \exp(-\mu/\lambda w) \approx \frac{\mu}{\lambda w}.$$

That is, although $\frac{\mu}{\lambda w}$ should typically be a small number, at each “level” n the attack succeeds with a constant probability. Therefore, it will a.s. succeed. The typical time until it succeeds is roughly $3^{\frac{\lambda w}{\mu}}$. Although this quantity may be very large, the probability that the “first”²² attack succeeds is not negligible. Therefore, we need countermeasures. One such countermeasure would be limiting the own weight from above, or even setting it to a constant value. As mentioned in Section 3, the latter may not be the best solution because it does not offer enough protection from spam.

Now, let us discuss the situation where the maximum own weight is capped at a value of 1, and estimate the probability that the attack succeeds.

Assume that a given transaction gained cumulative weight w_0 in t_0 time units after the moment when it was issued, and that the adaptation period for that transaction is over. In this situation, the transaction’s cumulative weight increases linearly with speed λ . Now, imagine that the attacker wants to double-spend on this transaction. To do so, the attacker secretly prepares the double-spending transaction, and starts generating *nonsense* transactions that approve the double-spending transaction at the time²³ when the *original* transaction was issued to the merchant. If the attacker’s subtangle outpaces the legitimate subtangle at some moment after the merchant decides to accept the legitimate transaction, then the double-spending attack would be successful. If that does not happen, then the double-spending transaction would not be approved by others because the legitimate transaction would acquire more cumulative weight and essentially all new tips would indirectly approve it. The double-spending transaction would be orphaned in this scenario.

As before, let μ stand for the computing power of the attacker. We also make a simplifying assumption that the transactions propagate instantly. Let G_1, G_2, G_3, \dots denote i.i.d. exponential random variables with parameter μ ²⁴, and define $V_k = \mu G_k$, $k \geq 1$. It follows that V_1, V_2, V_3, \dots are i.i.d. exponential random variables with parameter 1.

²²During the time t_0 .

²³Or even before; we discuss this case later.

²⁴With expected value $1/\mu$.

Suppose that at time t_0 the merchant decides to accept the transaction with cumulative weight w_0 . Let us estimate the probability that the attacker successfully double-spends. Let $M(\theta) = (1 - \theta)^{-1}$ be the moment generating function of the exponential distribution with parameter 1 (Section 7.7 of [14]). It is known²⁵ that for $\alpha \in (0, 1)$ it holds that

$$\mathbb{P}\left[\sum_{k=1}^n V_k \leq \alpha n\right] \approx \exp(-n\varphi(\alpha)), \quad (9)$$

where $\varphi(\alpha) = -\ln \alpha + \alpha - 1$ is the Legendre transform of $\ln M(\theta)$. As a general fact, it holds that $\varphi(\alpha) > 0$ for $\alpha \in (0, 1)$. Recall that the expectation of an exponential random variable with parameter 1 also equals 1.

Assume that $\frac{\mu t_0}{w_0} < 1$, otherwise the probability that the attacker's subtangle eventually outpaces the legitimate subtangle would be close to 1. Now, to outweigh w_0 at time t_0 , the attacker needs to be able to issue at least w_0 transactions with maximum own weight m during time t_0 . Therefore, using (9), we find the probability that the double-spending transaction has more cumulative weight at time t_0 is roughly

$$\begin{aligned} \mathbb{P}\left[\sum_{k=1}^{w_0/m} G_k < t_0\right] &= \mathbb{P}\left[\sum_{k=1}^{w_0} V_k < \mu t_0\right] \\ &= \mathbb{P}\left[\sum_{k=1}^{w_0} V_k < w_0 \times \frac{\mu t_0}{w_0}\right] \\ &\approx \exp(-w_0\varphi(\frac{\mu t_0}{w_0})). \end{aligned} \quad (10)$$

For the above probability to be small, $\frac{w_0}{m}$ needs to be large and $\varphi(\frac{\mu t_0}{w_0})$ cannot be very small.

Note that, at time $t \geq t_0$, the cumulative weight of the legitimate transaction is roughly $w_0 + \lambda(t - t_0)$ because we assumed that the adaptation period is over, so the cumulative weight grows with speed λ . Analogous to (10), one finds the probability that the double-spending transaction has more cumulative weight at time $t \geq t_0$ is roughly

$$\exp(-(w_0 + \lambda(t - t_0))\varphi(\frac{\mu t}{w_0 + \lambda(t - t_0)})). \quad (11)$$

Then, it must be true that we have $\frac{\mu t_0}{w_0} \geq \frac{\mu}{\lambda}$ since the cumulative weight grows with speed less than λ during the adaptation period. It can be shown that the probability

²⁵This is a consequence of the so-called Large Deviation Principle. See the general book [13], and Proposition 5.2 in Section 8.5 of [14] for a simple and instructive derivation of the upper bound, and Section 1.9 of [5] for the (not so simple) derivation of the lower bound.

of achieving a successful double spend is of order

$$\exp\left(-w_0\varphi\left(\max\left(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}\right)\right)\right). \quad (12)$$

For example, let $\mu = 2$, $\lambda = 3$ so that the attacker's power is only a bit less than that of the rest of the network. Assume that the transaction has a cumulative weight of 32 by time 12. Then, $\max\left(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}\right) = \frac{3}{4}$, $\varphi\left(\frac{3}{4}\right) \approx 0.03768$, and (12) then gives the upper bound approximately 0.29. If one assumes that $\mu = 1$ and keeps all other parameters intact, then $\max\left(\frac{\mu t_0}{w_0}, \frac{\mu}{\lambda}\right) = \frac{3}{8}$, $\varphi\left(\frac{3}{8}\right) \approx 0.3558$, and (12) gives approximately 0.00001135, quite a drastic change.

From the above discussion it is important to recognize that the inequality $\lambda > \mu$ should be true for the system to be secure. In other words, the input flow of "honest" transactions should be large compared to the attacker's computational power. Otherwise, the estimate (12) would be useless. This indicates the need for additional security measures, such as checkpoints, during the early days of a tangle-based system.

When choosing a strategy for deciding which one of two conflicting transactions is valid, one has to be careful when using cumulative weight as a decision metric. This is due to the fact that cumulative weight can be subject to an attack similar to the one described in Section 4.1, namely the attacker may prepare a double-spending transaction well in advance, build a secret subtangle referencing it, and then broadcast that subtangle after the merchant accepts the legitimate transaction. A better method for deciding between two conflicting transactions might be the one described in the next section: run the tip selection algorithm and see which of the two transactions is indirectly approved by the selected tip.

4.1 A parasite chain attack and a new tip selection algorithm

Consider the following attack (Figure 6): an attacker secretly builds a subtangle that occasionally references the main tangle to gain a higher score. Note that the score of honest tips is roughly the sum of all own weights in the main tangle, while the score of the attacker's tips also contains the sum of all own weights in the parasite chain. Since network latency is not an issue for an attacker who builds a subtangle alone²⁶, they might be able to give more height to the parasite tips if they use a computer that is sufficiently strong. Moreover, the attacker can artificially increase their tip count at the moment of the attack by broadcasting many new transactions

²⁶This is due to the fact that an attacker can always approve their own transactions without relying on any information from the rest of the network.

that approve transactions that they issued earlier on the parasite chain (Figure 6). This will give the attacker an advantage in the case where the honest nodes use some selection strategy that involves a simple choice between available tips.

To defend against this attack style, we are going to use the fact that the main tangle is supposed to have more active hashing power than the attacker. Therefore, the main tangle is able to produce larger increases in cumulative weight for more transactions than the attacker. The idea is to use a MCMC algorithm to select the two tips to reference.

Let \mathcal{H}_x be the current cumulative weight of a site. Recall that we assumed all own weights are equal to 1. Therefore, the cumulative weight of a tip is always 1, and the cumulative weight of other sites is at least 2.

The idea is to place some particles, a.k.a. random walkers, on sites of the tangle and let them walk towards the tips in a random²⁷ way. The tips “chosen” by the walks are then the candidates for approval. The algorithm is described in the following way:

1. Consider all sites on the interval $[W, 2W]$, where W is reasonably large²⁸.
2. Independently place N particles on sites in that interval²⁹.
3. Let these particles perform independent discrete-time random walks “towards the tips”, meaning that a transition from x to y is possible if and only if y approves x
4. The two random walkers that reach the tip set first will sit on the two tips that will be approved. However, it may be wise to modify this rule in the following way: first discard those random walkers that reached the tips *too fast* because they may have ended on one of the “lazy tips”.
5. The transition probabilities of the walkers are defined in the following way: if y approves x ($y \rightsquigarrow x$), then the transition probability P_{xy} is proportional to

²⁷There is not a “canonical” source of randomness. The nodes just use their own (pseudo)random number generators to simulate the random walks.

²⁸The idea is to place the particle “deep” into the tangle so that it will not arrive at a tip straight away. However, the particle should not be placed “too deep” because it needs to find a tip in a reasonable time. Also, the interval $[W, 2W]$ is arbitrary. One could chose $[W, 5W]$, etc. There are also other ways to select the walkers’ starting points. For example, a node can simply take a random transaction received between t_0 and $2t_0$ time units in the past, where t_0 is some fixed time point.

²⁹This choice is largely arbitrary. We use several particles instead of just two for additional security. The idea is that if a particle were to accidentally jump to the attacker’s chain, which is supposed to be long, then it would spend a lot of time there and other tips will be chosen first.

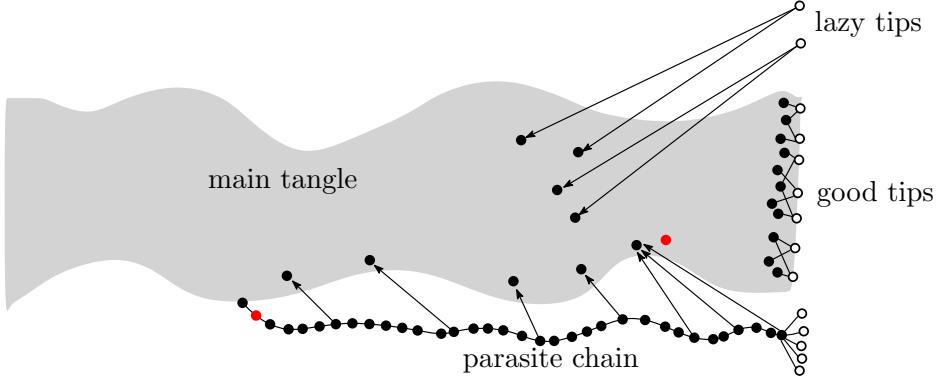


Figure 6: Visual representation of the tip selection algorithm for honest tips, as well as the parasite chain. The two red circles indicate an attempted double-spend by an attacker.

$\exp(-\alpha(\mathcal{H}_x - \mathcal{H}_y))$, that is

$$P_{xy} = \exp(-\alpha(\mathcal{H}_x - \mathcal{H}_y)) \left(\sum_{z: z \rightsquigarrow x} \exp(-\alpha(\mathcal{H}_x - \mathcal{H}_z)) \right)^{-1}, \quad (13)$$

where $\alpha > 0$ is a parameter to be chosen³⁰.

Note that this algorithm is “local”, meaning one does not need to traverse the tangle back to the genesis to perform relevant calculations. In particular, observe that one does not need to calculate the cumulative weights for the whole tangle. At most one needs to calculate the cumulative weights for the sites that indirectly approve the starting point of the walker.

To check that the algorithm works as intended, first consider the “lazy tips”. These tips intentionally approve some old transactions to avoid doing verification work (Figure 6). Even if the particle is on a site approved by a lazy tip, it is not probable that the lazy tip would be selected because the difference between cumulative weights would be very large and P_{xy} would be small.

Next, consider this alternate attack style: the attacker secretly builds a chain containing a transaction that empties their account balance to another account under their control, indicated as the leftmost red circle in Figure 6. Then, the attacker issues a transaction on the main tangle, represented by the rightmost red circle, and waits for the merchant to accept it. The parasite chain occasionally references the main

³⁰One can start with $\alpha = 1$.

tangle. However, the cumulative weight is not very large in the parasite chain. It should be noted that the parasite chain cannot reference the main tangle after the merchant’s transaction. Furthermore, the attacker might try to artificially inflate the number of tips in their parasite chain at the moment of the attack (Figure 6). The attacker’s idea is to make the nodes issuing new transactions reference the parasite chain so that the honest branch of the tangle will be orphaned.

It is easy to see why the MCMC selection algorithm will not select one of the attacker’s tips with high probability. The reasoning is identical to the lazy tip scenario: the sites on the parasite chain will have a cumulative weight that is much smaller than the sites that they reference on the main tangle. Therefore, it is not probable that the random walker will ever jump to the parasite chain unless it begins there, and this event is not very probable either because the main tangle contains more sites.

As an additional protecting measure, we can first run a random walk with a large α (so that it is in fact “almost deterministic”) to choose a “model tip”; then, use random walks with small α for actual tip selection, but verify if the (indirectly) referenced transactions are consistent with the model tip.

Observe also that, for a random walk that *always* moves towards the tips it is very simple and rapid to calculate the exit probability distribution using a straightforward recursion; this is something that we *do not* want the nodes to do. However, it is possible to modify our approach in the following way: on each step, the random walk may backtrack (i.e., go 1 step away from the tips) with probability (say) $\frac{1}{3}$ (and divide the remaining $\frac{2}{3}$ as before). The walk will reach the tips very quickly anyway (because it has a drift towards the tips), but it will not be so easy to calculate the exit measure.

Let us comment on why the nodes would follow this algorithm. Recall from Section 1 that it is reasonable to assume that at least a “good” proportion of the nodes will follow the *reference* algorithm. Also, because of computational and network delays, the tip selection algorithm would rather work with a past snapshot of the tangle with respect to the moment when a transaction is issued. *It may be a good idea to intentionally move this snapshot to a time point further in the past*³¹ in the reference algorithm for the reasons that we explain in the sequel. Imagine a “selfish” node that just wants to maximize the chances of their transaction being approved quickly. The MCMC algorithm of this section, which is adopted by a considerable proportion of nodes, defines a probability distribution on the set of tips. It is clear that

³¹First the random walker finds a former tip with respect to that snapshot, and then it continues to walk towards the “actual” tips on the current tangle.

a natural first choice for a selfish node would be to choose the tips where the maximum of that distribution is attained. However, if many other nodes also behave in a selfish way and use the same strategy, which is a reasonable assumption, then they all will lose. *Many* new transactions will approve the same two tips at roughly the same time, therefore generating too much competition between them for subsequent approval. It should also be clear that nodes will not immediately “feel” the cumulative weight increase caused by this mass approval of the same two tips since the nodes are using a past snapshot. For this reason, even a selfish node would have to use some random tip approval algorithm³² with a probability distribution for tip selection that is close to the default probability distribution produced by the reference tip selection algorithm. We do not claim that this “aggregated” probability distribution would be equal to the default probability distribution in the presence of selfish nodes. However, the above argument shows that it should be close to it. This means that the probability of many nodes attempting to verify the same “bad” tips would remain small. In any case, there is not a large incentive for the nodes to be selfish because possible gains only amount to a slight decrease in confirmation time. This is inherently different from other decentralized constructs, such as Bitcoin. The important fact is that nodes do not have reasons to abandon the MCMC tip selection algorithm.

We would like to mention that the definition of transition probabilities, as given in (13), has not been set in stone. Instead of the exponent, one can use a different function that decreases rapidly, such $f(s) = s^{-3}$. There is also freedom for choosing W and N as well. At this point in time, it is unclear if there are any theoretical arguments that show exactly in which way these parameters should be chosen. In sum, we feel that the main contribution of this section is the idea of using MCMC for tip selection.

4.2 Splitting attack

Aviv Zohar suggested the following attack scheme against the proposed MCMC algorithm. In the high-load regime, an attacker can try to split the tangle into two branches and maintain the balance between them. This would allow both branches to continue to grow. The attacker must place at least two conflicting transactions

³²as noticed before, for a backtracking walk there seem to be no easy way to discover which tips are better (that is, more likely to be selected by “honest” nodes) other than running the MCMC many times. However, running MCMC many times requires time and other resources; after one spends some time on it, the state of the tangle will already change, so one would possibly even have to start anew. This explains why nodes do not have reasons to abandon the MCMC tips selection strategy in favor of something else, at least if they assume that a considerable proportion of the other nodes follow the default tips selection strategy.

at the beginning of the split to prevent an honest node from effectively joining the branches by referencing them both simultaneously. Then, the attacker hopes that roughly half of the network would contribute to each branch so that they would be able to “compensate” for random fluctuations, even with a relatively small amount of personal computing power. If this technique works, the attacker would be able to spend the same funds on the two branches.

To defend against such an attack, one needs to use a “sharp-threshold” rule that makes it too hard to maintain the balance between the two branches. An example of such a rule is selecting the longest chain on the Bitcoin network. Let us translate this concept to the tangle when it is undergoing a splitting attack. Assume that the first branch has total weight 537, and the second branch has total weight 528. If an honest node selects the first branch with probability very close to $1/2$, then the attacker would probably be able to maintain the balance between the branches. However, if an honest node selects the first branch with probability much larger than $1/2$, then the attacker would probably be unable to maintain the balance. The inability to maintain balance between the two branches in the latter case is due to the fact that after an inevitable random fluctuation, the network will quickly choose one of the branches and abandon the other. In order to make the MCMC algorithm behave this way, one has to choose a very rapidly decaying function f , and initiate the random walk at a node with large depth so that it is highly probable that the walk starts before the branch bifurcation. In this case, the random walk would choose the “heavier” branch with high probability, even if the difference in cumulative weight between the competing branches is small.

It is worth noting that the attacker’s task is very difficult because of network synchronization issues: they may not be aware of a large number of recently issued transactions³³. Another effective method for defending against a splitting attack would be for a sufficiently powerful entity to instantaneously publish a large number of transactions on one branch, thus rapidly changing the power balance and making it difficult for the attacker to deal with this change. If the attacker manages to maintain the split, the most recent transactions will only have around 50% confirmation confidence (Section 1), and the branches will not grow. In this scenario, the “honest” nodes may decide to start selectively giving their approval to the transactions that occurred before the bifurcation, bypassing the opportunity to approve the conflicting transactions on the split branches.

One may consider other versions of the tip selection algorithm. For example, if a node sees two big subtangles, then it chooses the one with a larger sum of own

³³The “real” cumulative weights may be quite different from what they believe.

weights before performing the MCMC tip selection algorithm outlined above.

The following idea may be worth considering for future implementations. One could make the transition probabilities defined in (13) depend on both $\mathcal{H}_x - \mathcal{H}_y$ and \mathcal{H}_x in such a way that the next step of the Markov chain is almost deterministic when the walker is deep in the tangle, yet becomes more random when the walker is close to tips. This will help avoid entering the weaker branch while assuring sufficient randomness when choosing the two tips to approve.

Conclusions:

1. We considered attack strategies for when an attacker tries to double-spend by “outpacing” the system.
2. The “large weight” attack means that, in order to double-spend, the attacker tries to give a very large weight to the double-spending transaction so that it would outweigh the legitimate subtangle. This strategy would be a menace to the network in the case where the allowed own weight is unbounded. As a solution, we may limit the own weight of a transaction from above, or set it to a constant value.
3. In the situation where the maximal own weight of a transaction is m , the best attack strategy is to generate transactions with own weight m that reference the double-spending transaction. When the input flow of “honest” transactions is large enough compared to the attacker’s computational power, the probability that the double-spending transaction has a larger cumulative weight can be estimated using the formula (12) (see also examples below (12)).
4. The attack method of building a “parasite chain” makes approval strategies based on height or score obsolete since the attacker’s sites will have higher values for these metrics when compared to the legitimate tangle. On the other hand, the MCMC tip selection algorithm described in Section 4.1 seems to provide protection against this kind of attack.
5. The MCMC tip selection algorithm also offers protection against the lazy nodes as a bonus.

5 Resistance to quantum computations

It is known that a sufficiently large quantum computer³⁴ could be very efficient for handling problems that rely on trial and error to find a solution. The process of finding a nonce in order to generate a Bitcoin block is a good example of such a problem. As of today, one must check an average of 2^{68} nonces to find a suitable hash that allows a new block to be generated. It is known (see e.g. [15]) that a quantum computer would need $\Theta(\sqrt{N})$ operations to solve a problem that is analogous to the Bitcoin puzzle stated above. This same problem would need $\Theta(N)$ operations on a classical computer. Therefore, a quantum computer would be around $\sqrt{2^{68}} = 2^{34} \approx 17$ billion times more efficient at mining the Bitcoin blockchain than a classical computer. Also, it is worth noting that if a blockchain does not increase its difficulty in response to increased hashing power, there would be an increased rate of orphaned blocks.

For the same reason, a “large weight” attack would also be much more efficient on a quantum computer. However, capping the weight from above, as suggested in Section 4, would effectively prevent a quantum computer attack as well. This is evident in iota because the number of nonces that one needs to check in order to find a suitable hash for issuing a transaction is not unreasonably large. On average, it is around 3^8 . The gain of efficiency for an “ideal” quantum computer would therefore be of order $3^4 = 81$, which is already quite acceptable³⁵. More importantly, the algorithm used in the iota implementation is structured such that the time to find a nonce is not much larger than the time needed for other tasks that are necessary to issue a transaction. The latter part is much more resistant against quantum computing, and therefore gives the tangle much more protection against an adversary with a quantum computer when compared to the (Bitcoin) blockchain.

Acknowledgements

The author thanks Bartosz Kusmierz, Cyril Grünspan, Olivia Saa, Razvan Savu, Samuel Reid, Toru Kazama, Rafael Kallis, and Rodrigo Bueno who pointed out several errors in earlier drafts, and James Brogan for his contributions towards making this paper more readable.

³⁴Still a hypothetical construct as of today.

³⁵Note that $\Theta(\sqrt{N})$ could easily mean $10\sqrt{N}$.

References

- [1] Iota: a cryptocurrency for Internet-of-Things. See <http://www.iotatoken.com/>, and <https://bitcointalk.org/index.php?topic=1216479.0>
- [2] bitcoinj. Working with micropayment channels.
<https://bitcoinj.github.io/working-with-micropayments>
- [3] PEOPLE ON NXTFORUM.ORG (2014) DAG, a generalized blockchain.
<https://nxtforum.org/proof-of-stake-algorithm/dag-a-generalized-blockchain/> (registration at nxtforum.org required)
- [4] MOSHE BABAIOFF, SHAHAR DOBZINSKI, SIGAL OREN, AVIV ZOHAR (2012) On Bitcoin and red balloons. *Proc. 13th ACM Conf. Electronic Commerce*, 56–73.
- [5] RICHARD DURRETT (2004) Probability – Theory and Examples. *Duxbury advanced series*.
- [6] SERGIO DEMIAN LERNER (2015) DagCoin: a cryptocurrency without blocks.
<https://bitslog.wordpress.com/2015/09/11/dagcoin/>
- [7] YONATAN SOMPOLINSKY, AVIV ZOHAR (2013) Accelerating Bitcoin’s Transaction Processing. Fast Money Grows on Trees, Not Chains.
<https://eprint.iacr.org/2013/881.pdf>
- [8] YONATAN SOMPOLINSKY, YOAD LEWENBERG, AVIV ZOHAR (2016) SPECTRE: Serialization of Proof-of-work Events: Confirming Transactions via Recursive Elections. <https://eprint.iacr.org/2016/1159.pdf>
- [9] YOAD LEWENBERG, YONATAN SOMPOLINSKY, AVIV ZOHAR (2015) Inclusive Block Chain Protocols.
http://www.cs.huji.ac.il/~avivz/pubs/15/inclusive_btc.pdf
- [10] JOSEPH POON, THADDEUS DRYJA (2016) The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.
<https://lightning.network/lightning-network-paper.pdf>
- [11] SHELDON M. ROSS (2012) *Introduction to Probability Models*. 10th ed.
- [12] DAVID VORICK (2015) Getting rid of blocks. slides.com/davidvorick/braids

- [13] AMIR DEMBO, OFER ZEITOUNI (2010) *Large Deviations Techniques and Applications*. Springer.
- [14] SHELDON M. ROSS (2009) *A First Course in Probability*. 8th ed.
- [15] GILLES BRASSARD, PETER HYER, ALAIN TAPP (1998) Quantum cryptanalysis of hash and claw-free functions. *Lecture Notes in Computer Science* **1380**, 163–169.

Dash: A Payments-Focused Cryptocurrency

Evan Duffield - evan@dash.org

Daniel Diaz - daniel@dash.org

Abstract. A cryptocurrency based on Bitcoin, the work of Satoshi Nakamoto, with various improvements such as a two-tier incentivized network, known as the masternode network. Included are other improvements such as PrivateSend, for increasing fungibility, and InstantSend, which allows instant transaction confirmation without a centralized authority.

1 Introduction

Bitcoin [1] is a cryptocurrency that has emerged as a popular medium of exchange and is the first digital currency that has attracted a substantial number of users [2]. Since its inception in 2009, Bitcoin has been rapidly growing in mainstream adoption and merchant usage [3]. A main issue with the acceptance of Bitcoin in point-of-sale (POS) situations is the time required to wait for the network to confirm the transaction made is valid. Some payment processors have created methods to allow vendors to take zero-confirmation transactions, but these solutions utilize a trusted counterparty to mediate the transaction outside of the protocol.

Bitcoin provides pseudonymous transactions in a public ledger, with a one-to-one relationship between sender and receiver. This provides a permanent record of all transactions that have ever taken place on the network [5]. Bitcoin is widely known in academic circles to provide a low level of privacy, although with this limitation many people still entrust their financial history to its blockchain.

Dash is the first cryptocurrency based on the work of Satoshi Nakamoto with built-in privacy functions. In this paper we propose a series of improvements to Bitcoin resulting in a decentralized, strongly anonymous cryptocurrency, with tamper-proof instant transactions and a secondary peer-to-peer (P2P) network incentivized to provide services to the Dash Network.

2 Masternode Network

Full nodes are servers running on a P2P network that allow peers to use them to receive updates about the events on the network. These nodes utilize significant amounts of traffic and other resources that incur a substantial cost. As a result, a steady decrease in the amount of these nodes has been observed for some time on the Bitcoin network [7] and as a result, block propagation times have been upwards of 40 seconds [14]. Many solutions have been proposed such as a new reward scheme by Microsoft Research [4] and the Bitnodes incentive program [6].

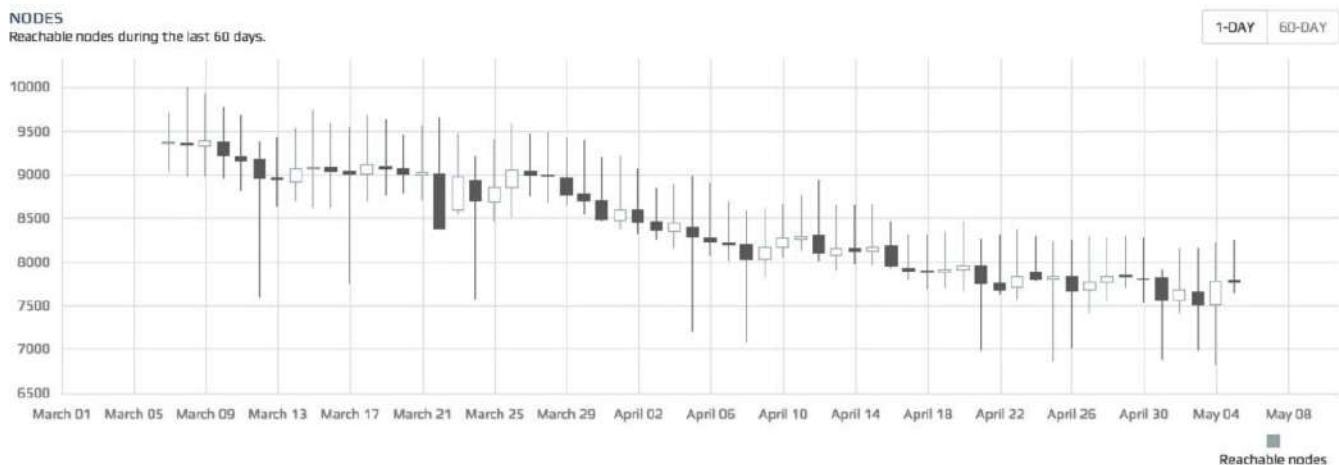


Figure 1: Bitcoin Full nodes in the spring of 2014

These nodes are very important to the health of the network. They provide clients with the ability to synchronize and facilitate quick propagation of messages throughout the network. We propose adding a secondary network, known as the Dash masternode network. These nodes will have high availability and provide a required level of service to the network in order to take part in the Masternode Reward Program.

2.1 Masternode Reward Program - Cost and Payments

Much of the reason for the decrease of full nodes on the Bitcoin network is the lack of incentive to run one. Over time, the cost of running a full node increases as the network gets used more, creating more bandwidth and costing the operator more money. As the cost rises, operators consolidate their services to be cheaper to run, or run a light client which does not help the network at all.

Masternodes are full nodes, just like in the Bitcoin network, except they must provide a level of service to the network and have a bond of collateral to participate. The collateral is never forfeit and is safe while the masternode is operating. This permits masternode operators to provide a service to the network, earn payment for their services and reduce the volatility of the currency.

To run a masternode, the operator must demonstrate control over 1,000 DASH. When active, masternodes provide services to clients on the network, and in return receive regular payment from the block reward. Like miners, masternodes are all paid from the block reward, 45% of which is dedicated to this program.

Due to the fact that the masternode rewards program is a fixed percentage and the masternode network nodes are fluctuating, expected masternode rewards will vary according to the current total count of active masternodes. Payments for a standard day for running a masternode can be calculated by using the following formula:

$$(n / t) * r * b * a$$

Where:

n is the number of masternodes an operator controls

t is the total number of masternodes

r is the current block reward (presently averaging about 3.3 DASH)

b is blocks in an average day. For the Dash network this usually is 576.

a is the average masternode payment (45% of the average block reward)

The cost associated with running a masternode creates a hard and soft limit of active nodes on the network. Currently with 8.2 million DASH in circulation, only 8,200 nodes could possibly be running on the network. The soft limit is imposed by the price it costs to acquire a node and the limited liquidity on exchanges due to usage of Dash as a currency and not merely an investment.

2.2 Deterministic Ordering

A special deterministic algorithm is used to create a pseudo-random ordering of the masternodes. By using the hash from the proof-of-work for each block, security of this functionality will be provided by the mining network.

Pseudocode, for selecting a masternode:

```
For(mastenode in masternodes){  
    current_score = mastenode.CalculateScore();  
  
    if(current_score > best_score){  
        best_score = current_score;  
        winning_node = mastenode;  
    }  
}  
  
CMasterNode::CalculateScore(){  
    pow_hash = GetProofOfWorkHash(nBlockHeight); // get the hash of this block  
    pow_hash_hash = Hash(pow_hash); //hash the POW hash to increase the entropy  
    difference = abs(pow_hash_hash - mastenode_vin);  
    return difference;  
}
```

The example code can be extended further to provide rankings of masternodes also, a “second”, “third”, “fourth” masternode in the list to be selected.

2.3 Trustless Quorums

Currently the Dash network has ~4,800 active masternodes [8]. By requiring 1,000DASH collateral to become an active masternode, we create a system in which no one can control the entire network of masternodes. For example, if someone wanted to control 50% of the masternode network, they would have to buy 4,800,000 DASH from the open market. This would raise the price substantially and it would become impossible to acquire the needed DASH.

With the addition of the masternode network and the collateral requirements, we can use this secondary network to do highly sensitive tasks in a trustless way, where no single entity can control the outcome. By selecting N pseudo random masternodes from the total pool to perform the same task, these nodes can act as an oracle, without having the whole network do the task.

As an example, implementation of a trustless quorum (see InstantSend [9]), which uses quorums to approve transactions and lock the inputs or the proof-of-service implementation [10].

Another example use for trustless quorums can include utilizing the masternode network as a decentralized oracle for financial markets, making secure decentralized contracts a possibility. As an example contract, if Apple Stock (AAPL) is over \$300 on Dec 31, 2018 pay public key A, otherwise pay public key B.

2.4 Roles and Proof-Of-Service

Masternodes can provide any number of extra services to the network. As a proof-of-concept, our first implementation included PrivateSend and InstantSend. By utilizing what we call proof-of-service, we can require that these nodes are online, responding and even at the correct block height.

Bad actors could also run masternodes, but not provide any of the quality service that is required of the rest of the network. To reduce the possibility of people using the system to their advantage nodes must ping the rest of the network to ensure they remain active. This work is done by the masternode network by selecting 2 quorums per block. Quorum A checks the service of Quorum B each block. Quorum A are the closest nodes to the current block hash, while Quorum B are the furthest nodes from said hash.

Masternode A (1) checks Masternode B (rank 2300)

Masternode A (2) checks Masternode B (rank 2299)

Masternode A (3) checks Masternode B (rank 2298)

All work done to check the network to prove that nodes are active is done by the masternode network itself. Approximately 1% of the network will be checked each block. This results in the entire network being checked about six times per day. In order to keep this system trustless, we select nodes randomly via the Quorum system, then we also require a minimum of six violations in order to deactivate a node.

In order to trick this system, an attacker will need to be selected six times in a row. Otherwise, violations will be cancelled out by the system as other nodes are selected by the quorum system.

Attacker Controlled Masternodes / Total Masternodes	Required Picked Times In A Row	Probability of success $(n/t)^r$	DASH Required
1/2300	6	6.75e-21	1,000DASH
10/2300	6	6.75e-15	10,000DASH
100/2300	6	6.75e-09	100,000DASH
500/2300	6	0.01055%	500,000DASH
1000/2300	6	0.6755%	1,000,000DASH

Table 1. The probability of tricking the system representing one individual masternode as failing proof-of-service

Where:

n is the total number of nodes controlled by the attacker

t is the total number of masternodes in the network

r is the depth of the chain

The selection of masternodes is pseudo random based on the Quorum system

2.5 Masternode Protocol

The masternodes are propagated around the network using a series of protocol extensions including a masternode announce message and masternode ping message. These two messages are all that is needed to make a node active on the network, beyond these there are other messages for executing a proof-of-service request, PrivateSend and InstantSend.

Masternodes are originally formed by sending 1,000 DASH to a specific address in a wallet that will “activate” the node making it capable of being propagated across the network. A secondary private key is created that is used for signing all further messages. The latter key allows the wallet to be completely locked when running in a standalone mode.

A cold mode is made possible by utilizing the secondary private key on two separate machines. The primary “hot” client signs the 1,000 DASH input including the secondary signing private key in the message. Soon after the “cold” client sees a message including its secondary key and activates as a masternode. This allows the “hot” client to be deactivated (client turned off) and leaves no possibility of an attacker gaining access to the 1,000 DASH by gaining access to the masternode after activation.

Upon starting, a masternode sends a “Masternode Announce” message to the network, containing:

Message: (1K DASH Input, Reachable IP Address, Signature, Signature Time, 1K Dash Public Key, Secondary Public Key, Donation Public Key, Donation Percentage)

Every 15 minutes thereafter, a ping message is sent proving the node is still alive.

Message: (1K DASH Input, Signature (using secondary key), Signature Time, Stop)

After a time-to-live has expired the network will remove an inactive node from the network, causing the node to not be used by clients or paid. Nodes can also ping the network constantly, but if they do not have their ports open, they will eventually be flagged as inactive and not be paid.

2.6 Propagation of the Masternode List

New clients entering the Dash network must be made aware of the currently active masternodes on the network to be able to utilize their services. As soon as they join the mesh network, a command is sent to their peers asking for the known list of masternodes. A cache object is used for clients to record masternodes and their current status, so when clients restart they will simply load this file rather than asking for the full list of masternodes.

2.7 Payments via Mining and Enforcement

To ensure that each masternode is paid its fair share of the block reward, the network must enforce that blocks pay the correct masternode. If a miner is non-compliant their blocks must be rejected by the network, otherwise cheating will be incentivized.

We propose a strategy where masternodes form quorums, select a winning masternode and broadcast their message. After N messages have been broadcast to select the same target payee, a consensus will be formed and that block in question will be required to pay that masternode.

When mining on the network, pool software (websites that merge the efforts of individual miners) use the RPC API interface to get information about how to make a block. To pay the masternodes, this interface must be extended by adding a secondary payee to GetBlockTemplate. Pools then propagate their successfully mined blocks, with a split payment between themselves and a masternode.

3 PrivateSend

We believe it is important to have a standard trustless implementation for improving the privacy of its users in the reference client that provides a high degree of privacy. Other clients such as Electrum, Android and iOS will also have the same anonymity layer implemented directly and utilize the protocol extensions. This allows users a common experience anonymizing funds using a well understood system.

PrivateSend is an improved and extended version of the CoinJoin. In addition to the core concept of CoinJoin, we employ a series of improvements such as decentralization, strong anonymity by using a chaining approach, denominations and passive ahead-of-time mixing.

The greatest challenge when improving privacy and fungibility of a cryptocurrency is doing it in a way that does not obscure the entire blockchain. In Bitcoin based crypto currencies, one can tell which outputs are unspent and which are not, commonly called UTXO (unspent transaction output). This results in a public ledger that allows any user to act as guarantor of the integrity of transactions. The Bitcoin protocol is designed to function without the participation of trusted counterparties, and in their absence, it is critical that auditing capabilities remain readily accessible to the users through the public blockchain. Our goal is to improve privacy and fungibility without losing these key elements that we believe make a successful currency.

By having a decentralized mixing service within the currency we gain the ability to keep the currency itself perfectly fungible. Fungibility is an attribute of money, that dictates that all units of a currency should remain equal. When you receive money within a currency, it should not come with any history from the previous users of the currency or the users should have an easy way to disassociate themselves from that history, thus keeping all coins equal. At the same time, any user should be able to act as an auditor to guarantee the financial integrity of the public ledger without compromising others privacy.

To improve the fungibility and keep the integrity of the public blockchain, we propose using an ahead-of-time decentralized trustless mixing strategy. To be effective at keeping the currency fungible, this service is directly built into the currency, easy to use and safe for the average user.

3.1 Tracing CoinJoin By Amounts

A common strategy in existing Bitcoin implementations of CoinJoin is simply merging transactions together. This exposes the users to various methods of following the the users coins through these joined transaction.



Figure 2: An example CoinJoin transaction with 2 users [11][12]

In this transaction, 0.05BTC was sent through the mixer. To identify the source of the money, one simply has to add up the values on the right until they match one of the values on the left.

Breaking apart the transaction:

- $0.05 + 0.0499 + 0.0001(\text{fee}) = 0.10\text{BTC}$.
- $0.0499 + 0.05940182 + 0.0001(\text{fee}) = 0.10940182\text{BTC}$.

This gets exponentially more difficult as more users are added to the mixer. However, these sessions can be retroactively de-anonymized at any point in the future.

3.2 Through Linking and Forward Linking

In other proposed implementations of CoinJoin, it is possible that a user anonymizes money then eventually sends change from that transaction to an exchange or other entity that knows the user's identity. This breaks the anonymity and allows the entity to walk backwards through that user's transactions. We call this type of attack "Forward Linking":

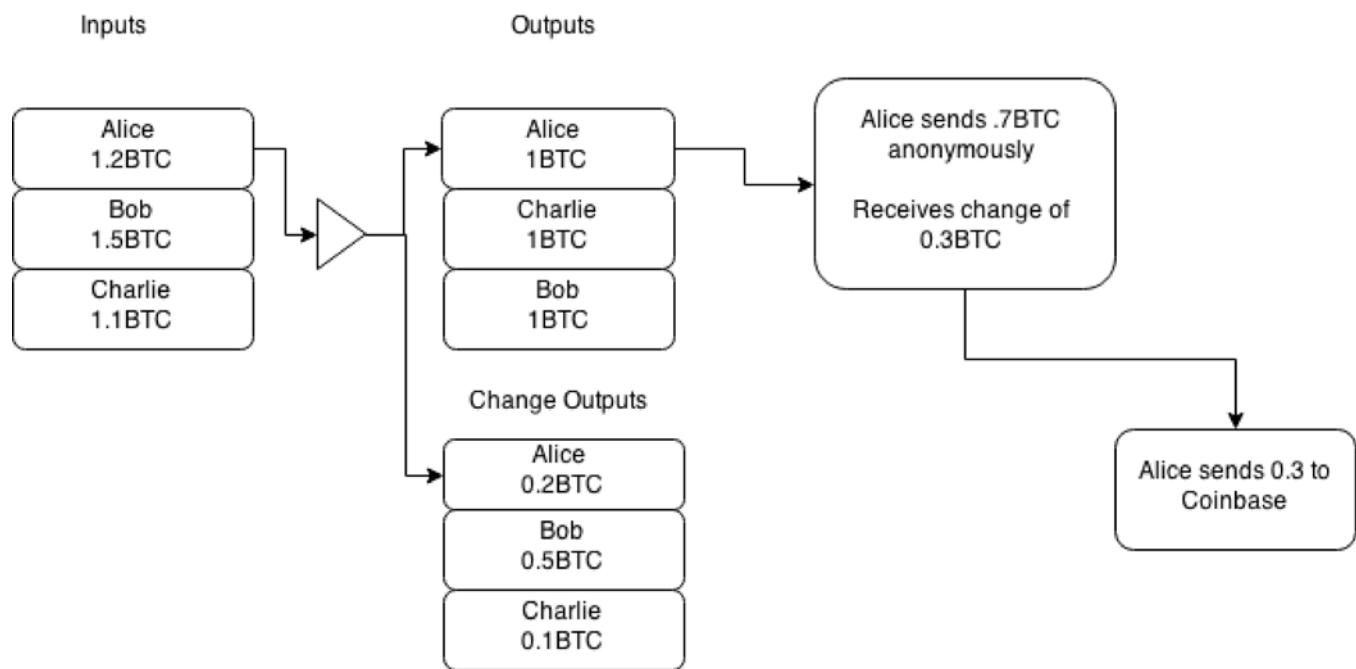


Figure 3: Forward Change Linking

In this example, Alice anonymizes 1.2 BTC, which goes to two outputs, 1 BTC and 0.2 BTC. She then spends 0.7 BTC from the 1 BTC output, receiving change of 0.3 BTC. That 0.3 BTC then goes to an identifiable source, confirming Alice also spent the 0.7 BTC in the prior transaction.

To identify the sender of the anonymous transaction, start at the "exchange" transaction and go backwards in the blockchain till you get to the "Alice sends 0.7 BTC anonymously". As the exchange, you know it was your user who just recently bought something anonymously, thus breaking the anonymity completely. We call this type of attack "Through Change Linking".

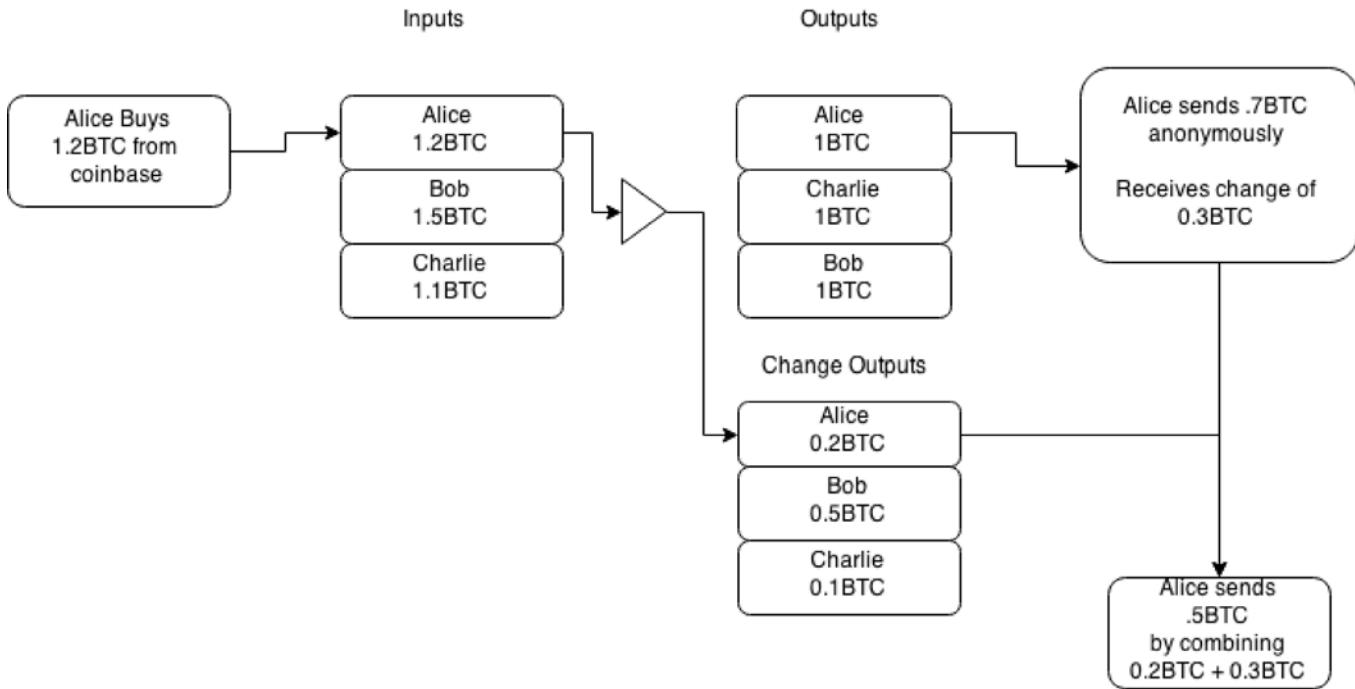


Figure 4: Through Change Linking

In the second example, Alice buys 1.2 BTC from coinbase, then anonymizes this amount into a 1 BTC output. She then spends the 1 BTC, receives change in the amount of 0.3 BTC and then combines that with her 0.2 BTC earlier change.

By combining the change from the anonymous transaction (0.3 BTC) and the change she received from the CoinJoin transaction, you can link the entire history before and after, completely breaking the anonymity.

3.3 Improved Privacy and Denial-of-Service (DOS) Resistance

PrivateSend uses the fact that a transaction can be formed by multiple parties and made out to multiple parties to merge funds together in a way where they cannot be uncoupled thereafter. Given that all PrivateSend transactions are setup for users to pay themselves, the system is highly secure against theft and users coins always remain safe. Currently, PrivateSend mixing requires at least three participants.

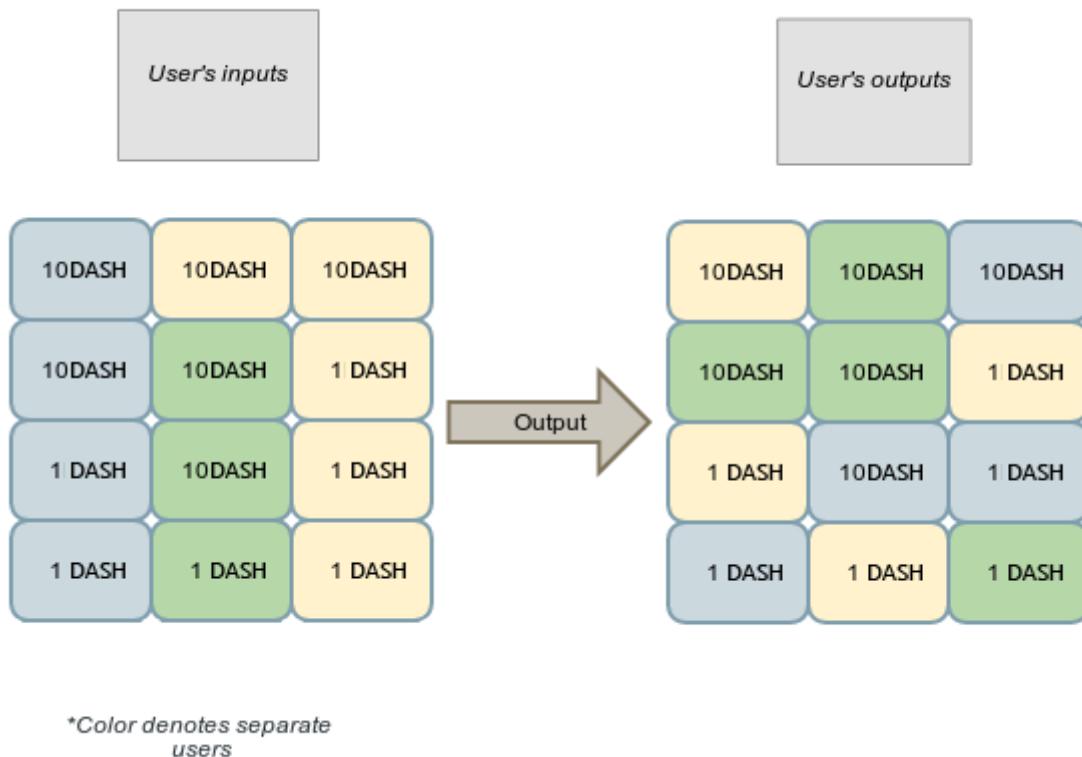


Figure 5: Three users submit denominated funds into a common transaction. Users pay themselves back in the form of new outputs, which are randomly ordered.

To improve the privacy of the system as a whole we propose using common denominations of 0.1DASH, 1DASH, 10DASH AND 100DASH. In each mixing session, all users should submit the same denominations as inputs and outputs. In addition to denominations, fees should be removed from the transactions and charged in bulk in separate, sporadic unlinkable transactions.

To address the possible DOS attacks, we propose all users submit a transaction as collateral to the pool when joining. This transaction will be made out to themselves and will pay a high fee to miners. In the case when a user submits a request to the mixing pool, they must provide collateral at the beginning of this exchange. If at any point any user fails to cooperate, by refusing to sign for example, the collateral transaction will be broadcast automatically. This will make it expensive to carry out a sustained attack on the privacy network.

3.4 Passive Anonymization of funds and chaining

PrivateSend is limited to 1,000DASH per session and requires multiple sessions to thoroughly anonymize significant amounts of money. To make the user experience easy and timing attacks very difficult, PrivateSend runs in a passive mode. At set intervals, a user's client will request to join with other clients via a masternode. Upon entry into the masternode, a queue object is propagated throughout the network detailing the denominations the user is looking to anonymize, but no information that can be used to identify the user.

Each PrivateSend session can be thought of as an independent event increasing the anonymity of user's funds. However each session is limited to three clients, so an observer has a one in three chance of being able to follow a transaction. To increase the quality of anonymity provided, a chaining approach is employed, which funds are sent through multiple masternodes, one after another.

Depth Of The Chain	Possible Users $(n)^r$
2	9
4	81
8	6561

Table 2. How many users could possibly be involved in N mixing sessions.

3.5 Security Considerations

As transactions are merged, masternodes can possibly "snoop" on users funds as they pass through. This is not considered a serious limitation due to the requirement for masternode's to hold 1,000DASH and the fact that users utilize random masternodes that they select to host their joins. The probability of following a transaction throughout a chaining event can be calculated as follows:

Attacker Controlled Masternodes / Total Masternodes	Depth Of The Chain	Probability of success $(n/t)^r$	DASH Required
10/1010	2	9.80e-05	10,000DASH
10/1010	4	9.60e-09	10,000DASH
10/1010	8	9.51e-11	10,000DASH
100/1100	2	8.26e-03	100,000DASH
100/1100	4	6.83e-05	100,000DASH
100/1100	8	4.66e-09	100,000DASH
1000/2000	2	25%	1,000,000DASH
1000/2000	4	6.25%	1,000,000DASH
1000/2000	8	0.39%	1,000,000DASH
2000/3000	2	44.4%	2,000,000DASH
2000/3000	4	19.75%	2,000,000DASH
2000/3000	8	3.90%	2,000,000DASH

Table 3. The probability of follow a PrivateSend transaction on the network given the attacker controls N Nodes.

Where:

n is the total number of nodes controlled by the attacker

t is the total number of masternodes in the network

r is the depth of the chain

The selection of masternodes is random.

Considering the limited supply of DASH (8.2 million at the time of writing, August 2018) and the low liquidity available on the market, it becomes an impossibility to attain a large enough number of masternodes to succeed at such an attack.

Extending the system by blinding masternodes to the transactions taking place on their node will also greatly enhance the security of the system.

3.6 Masternode Blinding via Relay System

In Section 3.4 we describe the probabilities of following a single transaction through multiple sessions of PrivateSend mixing. This can further be addressed by blinding masternodes, so they cannot see which inputs/outputs belong to which users. To do this we propose a simple relay system that users can use to protect their identity.

Instead of a user submitting the inputs and outputs directly into the pool, they will pick a random masternode from the network and request that it relays the inputs/outputs/signatures to the target masternode. This means that the masternode will receive N sets of inputs/outputs and N sets of signatures. Each set belongs to one of the users, but the masternode can't know which belongs to which.

4 Instant Transactions via InstantSend

By utilizing masternode quorums, users are able to send and receive instant irreversible transactions. Once a quorum has been formed, the inputs of the transaction are locked to only be spendable in a specific transaction, a transaction lock takes about four seconds to be set currently on the network. If consensus is reached on a lock by the masternode network, all conflicting transactions or conflicting blocks would be rejected thereafter, unless they matched the exact transaction ID of the lock in place.

This will allow vendors to use mobile devices in place of traditional POS systems for real world commerce and users to quickly settle face-to-face non commercial transactions as with traditional cash. This is done without a central authority. An extensive overview of this feature can be found in the InstantSend white paper [9].

5 Additional Improvements

5.1 X11 hashing algorithm

X11 is a widely used hashing algorithm, which takes a different approach, known as algorithm chaining. X11 consists of all 11 SHA3 contestants [13], each hash is calculated then submitted to the next algorithm in the chain. By utilizing multiple algorithms, the likelihood that an ASIC is created for the currency is minimal until a later part of its life cycle.

In the life cycle of Bitcoin, mining began with hobbyists which used Central Processing Units (CPUs) to mine the currency, then shortly after Graphics Processing Units (GPUs) software was created, which quickly replaced the CPUs. Years after the GPUs cycle, ASICs or Application Specific Integrated Circuits were created, which quickly replaced the GPUs.

Due to the complexity and die size required to create an ASIC to mine X11, we expect that it will take considerably longer than it did in Bitcoin, allowing for hobbyists to take part in the mining for a longer period of time. We believe this is highly important for good distribution and growth of a cryptocurrency.

Another benefit of the chaining hashing approach is high end CPUs give an average return similar to that of GPUs. Also GPUs have been reported to run 30-50% cooler, with less wattage than the Scrypt algorithm used by most current cryptocurrencies.

5.2 Mining Supply

A different approach to restricting the inflation of mining is taken in Dash, using a 7% reduction of the supply per year. This is done as opposed to halving implemented by other currencies. In addition supply each block is directly tied to the amount of miners on the network; more miners result in lower mining rewards.

Production of Dash is scheduled to carry on throughout this century and onto the next, slowly grinding down until finally near the year 2150, production will cease.

Dash Currency Supply and Mining Reward Schedule

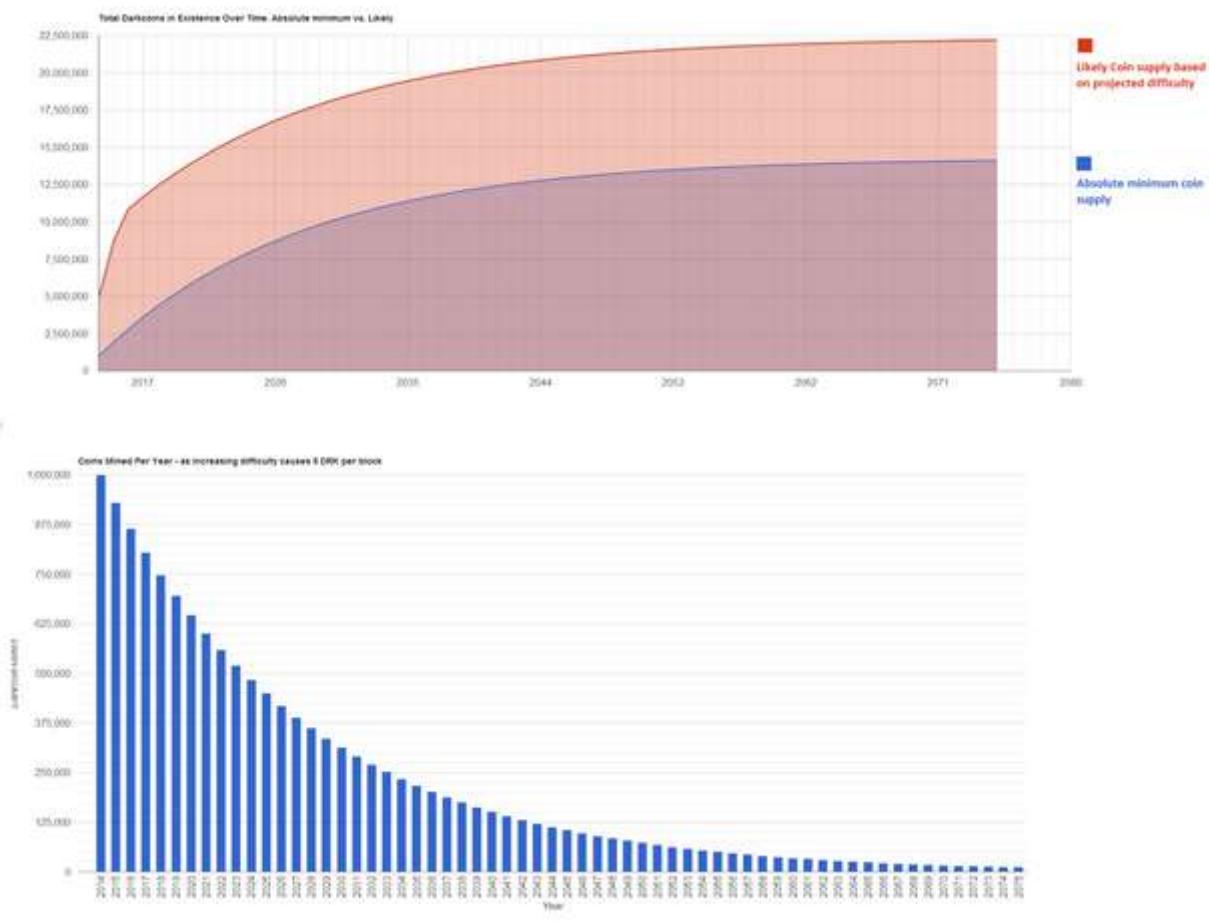


Figure 6: Mining Reward Schedule

6 Conclusion

This paper introduces various concepts to improve the design of bitcoin resulting in improved privacy and fungibility for the average user, less price volatility and quicker message propagation throughout the network. This is all accomplished by utilizing an incentivized two-tier model, rather than the existing single-tier model in other cryptocurrencies such as Bitcoin. By utilizing this alternative network design it becomes possible to add many types of services such as decentralized mixing of coins, instant transactions and decentralized oracles using masternode quorums.

References

1. [A peer-to-peer electronic cash system \(2008\)](#)
2. http://eprints.qut.edu.au/69169/1/Boyen_accepted_draft.pdf
3. <https://www.cryptocoinsnews.com/3-solutions-instant-bitcoin-confirmations/>
4. <http://research.microsoft.com/pubs/156072/bitcoin.pdf>
5. <http://www0.cs.ucl.ac.uk/staff/s.meiklejohn/files/imc13.pdf>
6. <https://getaddr.bitnodes.io/nodes/incentive/>
7. <https://medium.com/zapchain-magazine/why-don-t-people-run-bitcoin-nodes-anymore-d4da0b45aae5>
8. <https://dashninja.pl/>
9. <https://www.dash.org/wp-content/uploads/2014/09/InstantTX.pdf>
10. <https://github.com/dashpay/dash/blob/master/src/Masternode-pos.cpp>
11. <https://blockchain.info/tx/4eb3b2f9fe597d0aef6e43b58bbba7b8fb727e645fa89f922952f3e57ee6d603>
12. <https://blockchain.info/tx/1694122b34c8543d01ad422ce600d59f8d8fde495ac9ddd894edc7139aed7617>
13. http://en.wikipedia.org/wiki/NIST_hash_function_competition#Finalists
14. http://www.tik.ee.ethz.ch/file/49318d3f56c1d525aabf7fd78b23fc0/P2P2013_041.pdf

NEO White Paper

A distributed network for the Smart Economy

NEO design goals: Smart Economy

NEO is the use of blockchain technology and digital identity to digitize assets, the use of smart contracts for digital assets to be self-managed, to achieve "smart economy" with a distributed network.

Digital Assets

Digital assets are programmable assets that exist in the form of electronic data. With blockchain technology, the digitization of assets can be decentralized, trustful, traceable, highly transparent, and free of intermediaries. On the NEO blockchain, users are able to register, trade, and circulate multiple types of assets. Proving the connection between digital and physical assets is possible through digital identity. Assets registered through a validated digital identity are protected by law.

NEO has two forms of digital assets: global assets and contract assets. Global assets can be recorded in the system space and can be identified by all smart contracts and clients. Contract assets are recorded in the private storage area of the smart contract and require a compatible client to recognize them. Contract assets can adhere to certain standards in order to achieve compatibility with most clients.

Digital Identity

Digital identity refers to the identity information of individuals, organizations, and other entities that exist in electronic form. The more mature digital identity system is based on the PKI (Public Key Infrastructure) X.509 standard. In NEO, we will implement a set of X.509 compatible digital identity standards. This set of digital identity standards, in addition to compatible X.509 level certificate issuance model, will also support Web Of Trust point-to-point certificate issuance model. Our verification of identity when issuing or using digital identities includes the use of facial features, fingerprint, voice, SMS and other multi-factor authentication methods. At the same time, we will also use the

blockchain to replace the Online Certificate Status Protocol (OCSP) to manage and record the X.509 Certificate Revocation List (CRL).

Smart Contract

The smart contract was first proposed by the cryptographer Nick Szabo in 1994, only five years after the creation of the World Wide Web. According to Szabo's definition: When a pre-programmed condition is triggered, the smart contract will execute the corresponding contract terms. Blockchain technology provides us with a decentralized, tamper-resistant, highly reliable system in which smart contracts are very useful. NEO has an independent smart contract system: NeoContract.

The NeoContract smart contract system is the biggest feature of the seamless integration of the existing developer ecosystem. Developers do not need to learn a new programming language but use C#, Java and other mainstream programming languages in their familiar IDE environments (Visual Studio, Eclipse, etc.) for smart contract development, debugging and compilation. NEO's Universal Lightweight Virtual Machine, NeoVM, has the advantages of high certainty, high concurrency, and high scalability. The NeoContract smart contract system will allow millions of developers around the world to quickly carry out the development of smart contracts. NeoContract will have a separate white paper describing the implementation details.

Application and Ecosystem

Ecosystem is the vitality of the open source community. In order to achieve the goal of an intelligent economic network, NEO will be committed to the development of its ecosystem, providing mature development tools, improving development of documents, organizing education and training activities, and providing financial support. We plan to support the following NEO-based applications and ecology and to reward improvements to the design of the experience:

- ◆ Node Program

- A fully functioning Full node PC program
- A light node PC program with a better user experience
- Web / Android / iOS clients that do not need to synchronize with the blockchain
- Hardware wallet

- ◆ Blockchain Explorer
- ◆ SDK Development Kit
 - Support Java / Kotlin, .NET C # / VB, JavaScript / Typescript, Python, Go
- ◆ Smart Contract Compiler and IDE Plugin
 - C# / VB.Net / F#, Visual Studio
 - Java / Kotlin, Eclipse
 - C / C++ / GO
 - JavaScript / TypeScript
 - Python / Ruby
- ◆ Decentralized Applications
 - Smart fund
 - AI-assisted legal smart contract
 - Social networking
 - Automated tokens liquidity providers
 - Decentralized exchange
 - Secure communication protocol
 - Data exchange market
 - Intellectual property trading market
 - Prediction market
 - Advertising market
 - Hashpower market
 - NeoGas market

NEO Management Model

Economic Model

NEO has two native tokens, NEO (abbreviated symbol NEO) and NeoGas (abbreviated symbol GAS).

NEO, with a total of 100 million tokens, represents the right to manage the network. Management rights include voting for bookkeeping, NEO network parameter changes, and so on. The minimum unit of NEO is 1 and tokens cannot be subdivided.

GAS is the fuel token for the realization of NEO network resource control, with a maximum total limit of 100 million. The NEO network charges for the operation and storage of tokens and smart contracts, thereby creating economic incentives for bookkeepers and preventing the abuse of resources. The minimum unit of GAS is 0.00000001.

In the genesis block of the NEO network, 100 million NEOs are generated, GAS has not yet been generated. 100 million GAS, corresponding to the 100 million NEO, will be generated through a decay algorithm in about 22 years time to address holding NEO. If NEO is transferred to a new address, the subsequent GAS generated will be credited to the new address.

The NEO network will set a threshold by voting to exempt GAS from a certain amount of transfer transactions and smart contract operations to enhance the user experience. When a large amount of spam transactions occur, NeoID can be used to prioritize transactions and smart contracts with qualified identities. Transactions and smart contracts with no qualifying digital identities can get priority by paying GAS.

Distribution Mechanism

NEO distribution:

NEO's 100 million tokens is divided into two portions. The first portion is 50 million tokens distributed proportionally to supporters of NEO during the crowdfunding. This portion has been distributed.

The second portion is 50 million NEO managed by the NEO Council to support NEO's long-term development, operation and maintenance and ecosystem. The NEO in this portion has a lockout period of 1 year and is unlocked only after October 16, 2017. This portion will not enter the exchanges and is only for long-term support of NEO projects.

The plans for it are as below:

- ◆ 10 million tokens (10% total) will be used to motivate NEO developers and members of the NEO Council

- ◆ 10 million tokens (10% total) will be used to motivate developers in the NEO ecosystem
- ◆ 15 million tokens (15% total) will be used to cross-invest in other block-chain projects, which are owned by the NEO Council and are used only for NEO projects
- ◆ 15 million (15% total) will be retained as contingency
- ◆ The annual use of NEO in principle shall not exceed 15 million tokens

GAS distribution:

GAS is generated with each new block. The initial total amount of GAS is zero. With the increasing rate of new block generation, the total limit of 100 million GAS will be achieved in about 22 years. The interval between each block is about 15-20 seconds, and 2 million blocks are generated in about one year.

Each year around 2 million blocks will be generated and the initial generation will be 8 GAS per block. There will be an annual reduction of 1 GAS per block, per year, to coincide with the passing of every 2 million blocks. The reduction will continue down to just 1 GAS per block and will be held at that rate for around 22 years. After the 44 millionth block the total GAS generated will have reached 100 million and from this point there will be no further generation of GAS from new blocks.

According to this release curve, 16% of the GAS will be created in the first year, 52% of the GAS will be created in the first four years, and 80% of the GAS will be created in the first 12 years. These GAS will be distributed proportionally in accordance with the NEO holding ratio, recorded in the corresponding addresses. NEO holders can initiate a claim transaction at any time and claim these GAS tokens at their holding addresses.

Governance mechanism

Chain governance: NEO token holders are the network owners and managers, managing the network through voting in the network, using the GAS generated from NEO to utilize the functions in the network. NEO tokens can be transferred.

Off-chain governance: NEO Council consists of the founding members of the NEO project, under which the management committee, technical committee and the secretariat, respectively, are responsible for strategic decision-making, technical decision-making and specific implementation. The NEO Council is responsible to the

NEO community for the promotion and development of NEO ecosystem as its primary objective.

NEO technology implementation

Consensus mechanism: dBFT

The dBFT is called the Delegated Byzantine Fault Tolerant, a Byzantine fault-tolerant consensus mechanism that enables large-scale participation in consensus through proxy voting. The holder of the NEO token can, by voting, pick the bookkeeper it supports. The selected group of bookkeepers, through BFT algorithm, reach a consensus and generate new blocks. Voting in the NEO network continues in real time, rather than in accordance with a fixed term.

The dBFT provides fault tolerance of $f = \lfloor (n-1) / 3 \rfloor$ for a consensus system consisting of n consensus nodes. This fault tolerance also includes both security and availability, resistant to general and Byzantine failures, and is suitable for any network environment. dBFT has good finality, meaning that once confirmations are final, the block can not be bifurcated, and the transaction will not be revoked or rolled back.

In the NEO dBFT consensus mechanism, taking about 15 to 20 seconds to generate a block, the transaction throughput is measured up to about 1,000TPS, which is excellent performance among the public chains. Through appropriate optimization, there is potential to reach 10,000TPS, allowing it to support large-scale commercial applications.

The dBFT combines digital identity technology, meaning the bookkeepers can be a real name of the individual or institution. Thus, it is possible to freeze, revoke, inherit, retrieve, and ownership transfer due to judicial decisions on them. This facilitates the registration of compliant financial assets in the NEO network. The NEO network plans to support such operations when necessary.

Smart contract system: NeoContract

NEO's smart contract system consists of three parts:

NeoVM - Universal Block Chain Virtual Machine:

NeoVM is a lightweight, general-purpose virtual machine whose architecture is very close to the JVM and .NET Runtime, similar to a virtual CPU that reads and executes instructions in the contract in sequence, performs process control based on the functionality of the instruction operations, logic operations and so on. It has a good start-up speed and versatility, is very suitable for small programs such as smart contracts, can also be ported to non-blockchain systems, or integrated with the IDE to provide an optimal development experience. NeoVM's functionality can be extended, like introducing a JIT (real-time compiler) mechanism, thereby enhancing the efficiency of the implementation.

InteropService - Interoperable Services:

Used to load the blockchain ledger, digital assets, digital identity, persistent storage area, NeoFS, and other underlying services. They are like virtual machines that are provided for virtual machines, enabling smart contracts to access these services at run time to achieve some advanced functionality. Through this low-coupling design, NeoVM can be ported to any blockchain or even non-blockchain system used, increasing the utility of the smart contracts.

DevPack - Compiler and IDE plugin:

DevPack includes the high-level language compiler and the IDE plug-in. Because NeoVM's architecture is very similar to JVM and .NET Runtime, the compilers in DevPack can compile Java byte code and .NET MSIL into NeoVM's instruction set. Java / Kotlin, C# developers do not need to learn new languages and will be able to immediately start developing smart contracts in VS, Eclipse and other familiar IDE environments. This greatly reduces the learning curve for developing smart contracts, allowing us to easily build a vibrant community around NeoContract.

NeoContract can create a smart contract call tree through static analysis before running a smart contract. Through the deterministic call tree, the NEO node can dynamically fragment the smart contract to achieve theoretically unlimited expansion, which overcomes the "jamming effect" caused by the static fragmentation of other block chain systems.

Cross-chain interoperability agreement: NeoX

NeoX is a protocol that implements cross-chain interoperability. NeoX is divided into two parts: "cross-chain assets exchange protocol" and "cross-chain distributed transaction protocol."

Cross-chain assets exchange agreement:

NeoX has been extended on existing double-stranded atomic assets exchange protocols to allow multiple participants to exchange assets across different chains and to ensure that all steps in the entire transaction process succeed or fail together. In order to achieve this function, we need to use NeoContract function to create a contract account for each participant. If other blockchains are not compatible with NeoContract, they can be compatible with NeoX as long as they can provide simple smart contract functionality.

Cross-chain distributed transaction protocol:

Cross-chain distributed transactions mean that multiple steps of a transaction are scattered across different blockchains and that the consistency of the entire transaction is ensured. This is an extension of cross-chain assets exchange, extending the behavior of assets exchange into arbitrary behavior. In layman's terms, NeoX makes it possible for cross-chain smart contracts where a smart contract can perform different parts on multiple chains, either succeeding or reverting as a whole. This gives excellent possibilities for cross-chain collaborations and we are exploring cross-chain smart contract application scenarios.

Distributed Storage Protocol: NeoFS

NeoFS is a distributed storage protocol that utilizes Distributed Hash Table (DHT) technology. NeoFS indexes the data through file content (Hash) rather than file path (URI). Large files will be divided into fixed-size data blocks that are distributed and stored in many different nodes.

The main problem with this type of system is the need to find a balance between redundancy and reliability. NeoFS plans to solve this contradiction by means of token incentives and the establishment of backbone nodes. Users can choose the reliability requirements of the file. Files with low reliability requirements can be stored and accessed for free or almost free. Stable and reliable services for files with high reliability requirement will be provided by backbone nodes.

NeoFS will serve as one of the InteropService interoperability services under the NeoContract system, enabling smart contracts to store large files on the blockchain and set access for those files. In addition, NeoFS can be combined with digital identity so that digital certificates used by digital identities can be assigned, sent, and revoked without a central server to manage them. In the future, the old block data can be stored in NeoFS, so that most of the full nodes can release the old data for better scalability and at the same time, ensure the integrity of historical data.

Anti-quantum cryptography mechanism: NeoQS

The emergence of quantum computers poses a major challenge to RSA and ECC-based cryptographic mechanisms. Quantum computers can solve the large number of decomposition problems (which RSA relies on) and the elliptic curve discrete logarithm (which ECC relies on) in a very short time. NeoQS (Quantum Safe) is a lattice-based cryptographic mechanism. At present, quantum computers do not have the ability to quickly solve the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP), which is considered to be the most reliable algorithm for resisting quantum computers.

Summary

NEO is a distributed network that combines digital assets, digital identities and smart contracts. The NEO system will use DBFT, NeoX, NeoFS, NeoQS and many other original technologies, as the infrastructure for the intelligent economy of the future.

The Dai Stablecoin System

Whitepaper

<https://makerdao.com/>

By the Maker Team

December 2017

Overview of the Dai Stablecoin System	3
Collateralized Debt Position Smart Contracts	3
The CDP interaction process	4
Single-Collateral Dai vs Multi-Collateral Dai	4
Pooled Ether (Temporary mechanism for Single-Collateral Dai)	5
Price Stability Mechanisms	5
Target Price	5
Target Rate Feedback Mechanism	6
Sensitivity Parameter	7
Global Settlement	7
Global Settlement: Step by Step	7
Risk Management of The Maker Platform	8
Risk Parameters	9
MKR Token Governance	10
MKR and Multi-Collateral Dai	11
Automatic Liquidations of Risky CDPs	11
Liquidity Providing Contract (Temporary mechanism for Single-Collateral Dai)	12
Debt and Collateral Auctions (Multi-Collateral Dai)	12
Key External Actors	13
Keepers	13
Oracles	14
Global Settlers	14
Examples	14
Addressable Market	16
Risks and their Mitigation	17
Malicious hacking attack against the smart contract infrastructure	17
Black swan event in one or more collateral assets	18
Competition and the importance of ease-of-use	18
Pricing errors, irrationality and unforeseen events	18
Failure of centralized infrastructure	19
Conclusion	19
Glossary of Terms	20
Links	21

Overview of the Dai Stablecoin System

Popular digital assets such as Bitcoin (BTC) and Ether (ETH) are too volatile to be used as everyday currency. The value of a bitcoin often experiences large fluctuations, rising or falling by as much as 25% in a single day and occasionally rising over 300% in a month.¹.

The Dai Stablecoin is a collateral-backed cryptocurrency whose value is stable relative to the US Dollar. We believe that stable digital assets like Dai Stablecoin are essential to realizing the full potential of blockchain technology.

Maker is a smart contract platform on Ethereum that backs and stabilizes the value of Dai through a dynamic system of Collateralized Debt Positions (CDPs), autonomous feedback mechanisms, and appropriately incentivized external actors.

Maker enables anyone to leverage their Ethereum assets to generate Dai on the Maker Platform. Once generated, Dai can be used in the same manner as any other cryptocurrency: it can be freely sent to others, used as payments for goods and services, or held as long term savings. Importantly, the generation of Dai also creates the components needed for a robust decentralized margin trading platform.

Collateralized Debt Position Smart Contracts

Anyone who has collateral assets can leverage them to generate Dai on the Maker Platform through Maker's unique smart contracts known as Collateralized Debt Positions.²

CDPs hold collateral assets deposited by a user and permit this user to generate Dai, but generating also accrues debt. This debt effectively locks the deposited collateral assets inside the CDP until it is later covered by paying back an equivalent amount of Dai, at which point the owner can again withdraw their collateral. Active CDPs are always collateralized in excess, meaning that the value of the collateral is higher than the value of the debt.

¹ David Ernst [Hard Problems in Cryptocurrency](#)

² <https://github.com/makerdao>

The CDP interaction process

- **Step 1: Creating the CDP and depositing collateral**

The CDP user first sends a transaction to Maker to create the CDP, and then sends another transaction to fund it with the amount and type of collateral that will be used to generate Dai. At this point the CDP is considered collateralized.

- **Step 2: Generating Dai from the collateralized CDP**

The CDP user then sends a transaction to retrieve the amount of Dai they want from the CDP, and in return the CDP accrues an equivalent amount of debt, locking them out of access to the collateral until the outstanding debt is paid.

- **Step 3: Paying down the debt and Stability Fee**

When the user wants to retrieve their collateral, they have to pay down the debt in the CDP, plus the Stability fee that continuously accrue on the debt over time. The Stability Fee can only be paid in MKR. Once the user sends the requisite Dai and MKR to the CDP, paying down the debt and Stability Fee, the CDP becomes debt free.

- **Step 4: Withdrawing collateral and closing the CDP**

With the Debt and Stability Fee paid down, the CDP user can freely retrieve all or some of their collateral back to their wallet by sending a transaction to Maker.

Single-Collateral Dai vs Multi-Collateral Dai

Dai will initially launch with support for only one type of collateral, Pooled Ether. In the next 6-12 months we plan to upgrade Single-Collateral Dai to Multi-Collateral Dai. The primary difference is that it will support any number of CDP types.³

³ Mechanics that are temporarily in place in the system during the Single-Collateral phase are marked in this white paper

Pooled Ether (Temporary mechanism for Single-Collateral Dai)

At first, Pooled Ether (PETH) will be the only collateral type accepted on Maker. Users who wish to open a CDP and generate Dai during the first phase of the Maker Platform need to first obtain PETH. This is done instantly and easily on the blockchain by depositing ETH into a special smart contract that pools the ETH from all users, and gives them corresponding PETH in return.

If there is a sudden market crash in ETH, and a CDP ends up containing more debt than the value of its collateral, the Maker Platform automatically dilutes the PETH to recapitalize the system. This means that the proportional claim of each PETH goes down.

After the Maker Platform is upgraded to support multiple collateral types, PETH will be removed and replaced by ETH alongside the other new collateral types.

Price Stability Mechanisms

Target Price

The Dai Target Price has two primary functions on the Maker Platform: 1) It is used to calculate the collateral-to-debt ratio of a CDP, and 2) It is used to determine the value of collateral assets Dai holders receive in the case of a global settlement.

The Target Price is initially denominated in USD and starts at 1, translating to a 1:1 USD soft peg.

Target Rate Feedback Mechanism

In the event of severe market instability, the Target Rate Feedback Mechanism (TRFM) can be engaged. Engaging the TRFM breaks the fixed peg of Dai, but maintains the same denomination.

The TRFM is the automatic mechanism by which the Dai Stablecoin System adjusts the Target Rate in order to cause market forces to maintain stability of the Dai market price around the Target Price. The Target Rate determines the change of the Target Price over time, so it can act either as an incentive to hold Dai (if the Target Rate is positive) or an incentive to borrow Dai (If the Target Rate is negative). When the TRFM is not engaged the target rate is fixed at 0%, so the target price doesn't change over time and Dai is pegged.

When the TRFM is engaged, both the Target Rate and the Target Price change dynamically to balance the supply and demand of Dai by automatically adjusting user incentives for generating and holding Dai. The feedback mechanism pushes the market price of Dai towards the variable Target Price, dampening its volatility and providing real-time liquidity during demand shocks.

With the TRFM engaged, when the market price of Dai is below the Target Price, the Target Rate increases. This causes the Target Price to increase at a higher rate, causing generation of Dai with CDPs to become more expensive. At the same time, the increased Target Rate causes the capital gains from holding Dai to increase, leading to a corresponding increase in demand for Dai. This combination of reduced supply and increased demand causes the Dai market price to increase, pushing it back up towards the Target Price.

The same mechanism works in reverse if the Dai market price is higher than the Target Price: the Target Rate decreases, leading to an increased demand for generating Dai and a decreased demand for holding it. This causes the Dai market price to decrease, pushing it down towards the Target Price.

This mechanism is a negative feedback loop: Deviation away from the Target Price in one direction increases the force in the opposite direction.

Sensitivity Parameter

The TRFM's Sensitivity Parameter is a parameter that determines the magnitude of Target Rate change in response to Dai target/market price deviation. This tunes the rate of feedback to the scale of the system. MKR voters can set the Sensitivity Parameter but when

the TRFM is engaged the Target Price and the Target Rate are determined by market dynamics, and not directly controlled by MKR voters.

The Sensitivity Parameter is also what is used to engage or disengage the TRFM. If the Sensitivity Parameter and the Target Rate are both zero, Dai is pegged to the current Target Price.

Global Settlement

Global settlement is a process that can be used as a last resort to cryptographically guarantee the Target Price to holders of Dai. It shuts down and gracefully unwinds the Maker Platform while ensuring that all users, both Dai holders and CDP users, receive the net value of assets they are entitled to. The process is fully decentralized, and MKR voters govern access to it to ensure that it is only used in case of serious emergencies. Examples of serious emergencies are long term market irrationality, hacking or security breaches, and system upgrades.

Global Settlement: Step by Step

- **Step 1: Global Settlement is activated**

If enough actors who have been designated as global settlers by Maker Governance believe that the system is subject to a serious attack, or if a global settlement is scheduled as part of a technical upgrade, they can active the Global Settlement function. This stops CDP creation and manipulation, and freezes the Price Feed at a fixed value that is then used to process proportional claims for all users.

- **Step 2: Global Settlement claims are processed**

After Global Settlement has been activated, a period of time is needed to allow keepers to process the proportional claims of all Dai and CDP holders based on the fixed feed value. After this processing is done, all Dai holders and CDP holders will be able to claim a fixed amount of ETH with their Dai and CDPs.

- **Step 3: Dai and CDP holders claim the collateral with their Dai and CDPs**

Each Dai and CDP holder can call a claim function on the Maker Platform to exchange their Dai and CDPs directly for a fixed amount of ETH that corresponds to the calculated value of their assets, based on the target price of Dai.

E.g. If the Dai Target Price is 1 U.S. Dollar, The ETH/USD Price is 200 and a user holds 1000 Dai when Global Settlement is activated, after the processing period they will be able to claim exactly 5 ETH from the Maker Platform. There is no time limit for when the final claim can be made.

Risk Management of The Maker Platform

The MKR token allows holders to vote to perform the following Risk Management actions:

- **Add new CDP type:** Create a new CDP type with a unique set of Risk Parameters. A CDP type can either be a new type of collateral, or a new set of Risk Parameters for an existing collateral type.
- **Modify existing CDP types:** Change the Risk Parameters of one or more existing CDP types that were already added
- **Modify Sensitivity Parameter:** Change the sensitivity of the Target Rate Feedback Mechanism
- **Modify Target Rate:** Governance can change the Target Rate. In practice modifying the Target Rate will only be done in one specific circumstance: When MKR voters want to peg the price of Dai to its current Target Price. It will always be done in conjunction with modifying the Sensitivity Parameter. By setting both Sensitivity Parameter and Target Rate to 0%, the TRFM becomes disabled and the Target Price of Dai becomes pegged to its current value.

- **Choose the set of trusted oracles:** The Maker Platform derives its internal prices for collateral and the market price of Dai from a decentralized oracle infrastructure, consisting of a wide set of individual oracle nodes. MKR voters control how many nodes are in the set of trusted oracles, and who those nodes are. Up to half of the oracles can be compromised or malfunction without causing a disruption to the continued safe operation of the system
- **Modify Price Feed Sensitivity:** Change the rules that determine the largest change that the price feeds can affect on the internal price values in the system.
- **Choose the set of global settlers:** Global settlement is a crucial mechanic that allows the Maker Platform to survive attacks against the oracles or the governance process. The governance process chooses a set of global settlers and determines how many settlers are needed to activate global settlement.

Risk Parameters

Collateralized Debt Positions have multiple Risk Parameters that enforce how they can be used. Each CDP type has its own unique set of Risk Parameters, and these parameters are determined based on the risk profile of the collateral used by the CDP type. These parameters are directly controlled by MKR holders through voting, with one MKR giving its holder one vote.

The key Risk Parameters for CDPs are:

- **Debt Ceiling:** The Debt Ceiling is the maximum amount of debt that can be created by a single type of CDP. Once enough debt has been created by a CDP of any given type, it becomes impossible to create more unless existing CDPs are closed. The debt ceiling is used to ensure sufficient diversification of the collateral portfolio.
- **Liquidation Ratio:** The Liquidation Ratio is the collateral-to-debt ratio at which a CDP becomes vulnerable to Liquidation. A low Liquidation Ratio means MKR voters expect low price volatility of the collateral, while a high Liquidation Ratio means high volatility is expected.

- **Stability Fee:** The Stability Fee is a fee paid by every CDP. It is an annual percentage yield that is calculated on top of the existing debt of the CDP and has to be paid by the CDP user. The Stability Fee is denominated in Dai, but can only be paid using the MKR token. The amount of MKR that has to be paid is calculated based on a Price Feed of the MKR market price. When paid, the MKR is burned, permanently removing it from the supply.
- **Penalty Ratio:** The Penalty Ratio is used to determine the maximum amount of Dai raised from a Liquidation Auction that is used to buy up and remove MKR from the supply, with excess collateral getting returned to the CDP user who owned the CDP prior to its liquidation. The Penalty Ratio is used to cover the inefficiency of the liquidation mechanism. During the phase of Single-Collateral Dai, the Liquidation Penalty goes to buy and burn of PETH, benefitting the PETH to ETH ratio.

MKR Token Governance

In addition to payment of the Stability Fee on active CDPs, the MKR token plays an important role in the governance of the Maker Platform.

Governance is done at the system level through election of an Active Proposal by MKR voters. The Active Proposal is the smart contract that has been empowered by MKR voting to gain root access to modify the internal governance variables of the Maker Platform. Proposals can be in two forms: Single Action Proposal Contracts [SAPC], and Delegating Proposal Contracts [DPC].

Single Action Proposal Contracts are proposals that can only be executed once after gaining root access, and after execution immediately applies its changes to the internal governance variables of the Maker Platform. After the one-time execution, the SAPC deletes itself and cannot be re-used. This type of proposal is what will be used during the first phases of the system, as it is not very complicated to use, but is less flexible.

Delegating Proposal Contracts are proposals that continuously utilize their root access through second layer governance logic that is codified inside the DPC. The second layer governance logic can be relatively simple, such as defining a protocol for holding a weekly

vote on updated risk parameters. It can also implement more advanced logic, such as restrictions on the magnitude of governance actions within defined time periods, or even delegating some or all of its permissions further to one or more third layer DPCs with or without restrictions.

Any Ethereum account can deploy valid proposal smart contracts. MKR voters can then use their MKR tokens to cast approval votes for one or more proposals that they want to elect as the Active Proposal. The smart contract that has the highest total number of approval votes from MKR voters is elected as the Active Proposal.

MKR and Multi-Collateral Dai

After the upgrade to Multi-Collateral Dai, MKR will take on a more significant role in the Dai Stablecoin System by replacing PETH as the the recapitalization resource. When CDPs become undercollateralized due to market crashes, the MKR supply is automatically diluted and sold off in order to raise enough funds to recapitalize the system.

Automatic Liquidations of risky CDPs

To ensure there is always enough collateral in the system to cover the value of all outstanding Debt (according to the Target Price), a CDP can be liquidated if it is deemed to be too risky. The Maker Platform determines when to liquidate a CDP by comparing the Liquidation Ratio with the current collateral-to-debt ratio of the CDP.

Each CDP type has its own unique Liquidation Ratio that is controlled by MKR voters and established based on the risk profile of the particular collateral asset of that CDP type.

Liquidation occurs when a CDP hits its Liquidation Ratio. The Maker Platform will automatically buy the collateral of the CDP and subsequently sell it off. There is a temporary mechanism in place for Single-Collateral Dai called a Liquidity Providing Contract. For Multi-Collateral Dai an auction mechanism will be used.

Liquidity Providing Contract (Temporary mechanism for Single-Collateral Dai)

During Single-Collateral Dai, the mechanism for liquidation is a Liquidity Providing Contract: a smart contract that trades directly with ethereum users and keepers according to the price feed of the system.

When a CDP is liquidated, it is immediately acquired by the system. The CDP owner receives the value of the leftover collateral minus the debt, Stability Fee and Liquidation Penalty.

The PETH collateral is set for sale in the Liquidity Providing Contract, and keepers can atomically purchase the PETH by paying Dai. All Dai paid this way are immediately removed from the Dai supply, until an amount equal to the CDP debt has been removed. If any Dai is paid in excess of the debt shortfall, the excess Dai is used to purchase PETH from the market and burn it, which positively changes the ETH to PETH ratio. This results in a net value gain for PETH holders.

If the PETH sell-off initially does not raise enough Dai to cover the entire debt shortfall, more PETH is continuously created and sold off. New PETH created this way negatively changes the ETH to PETH ratio, causing PETH holders to lose value.

Debt and Collateral Auctions (Multi-Collateral Dai)

During a liquidation, the Maker platform buys the collateral of a CDP and subsequently sells it in an automatic auction. This auction mechanism enables the system to settle CDPs even when price information is unavailable.

In order to take over the collateral of the CDP so that it can be sold, the system first needs to raise enough Dai to cover the CDP's debt. This is called a Debt Auction, and works by diluting the supply of the MKR token and selling it to bidders in an auction format.

In parallel, the collateral of the CDP is sold in a Collateral Auction where all proceeds (also denominated in Dai) up to the CDP debt amount plus a Liquidation Penalty (A Risk Parameter determined by MKR voting) is used to buy MKR and remove it from the supply. This directly counteracts the MKR dilution that happened during the Debt Auction. If enough Dai is bid to fully cover the CDP debt plus the Liquidation Penalty, the Collateral Auction switches to a reverse auction mechanism and tries to sell as little collateral as possible--any leftover collateral is returned to the original owner of the CDP.

Key External Actors

In addition to its smart contract infrastructure, the Maker Platform relies on certain external actors to maintain operations. Keepers are external actors who take advantage of the economic incentives presented by the Maker platform. Oracles and Global Settlers are external actors with special permissions in the system assigned to them by MKR voters.

Keepers

A keeper is an independent (usually automated) actor that is incentivized by profit opportunities to contribute to decentralized systems. In the context of the Dai Stablecoin System, keepers participate in the Debt Auctions and Collateral Auctions when CDPs are liquidated.

Keepers also trade Dai around the Target Price. Keepers sell Dai when the market price is higher than the Target Price and buy Dai when the market price is below the Target Price to profit from the expected long-term convergence towards the Target Price.

Oracles

The Maker Platform requires real time information about the market price of the assets used as collateral in CDPs in order to know when to trigger liquidations. The Maker Platform also needs information about the market price of Dai and its deviation from the Target Price in order to adjust the Target Rate when the TRFM is engaged. MKR voters choose a set of trusted oracles to feed this information to the Maker Platform through Ethereum transactions.

To protect the system from an attacker who gains control of a majority of the oracles, and from other forms of collusion, there is a global variable that determines the maximum change to the value of the price feed permitted by the system. This variable is known as the Price Feed Sensitivity Parameter.

As an example of how the Price Feed Sensitivity Parameter works, if the Price Feed Sensitivity Parameter is defined as “5% in 15 minutes”, the price feeds cannot change more than 5% within one 15 minute period, and changing ~15% would take 45 minutes. This restriction ensures there is enough time to trigger a global settlement in the event that an attacker gains control over a majority of the oracles.

Global Settlers

Global Settlers are external actors similar to price feed oracles and are the last line of defense for the Dai Stablecoin System in the event of an attack. The set of global settlers, selected by MKR voters, have the authority to trigger global settlement. Aside from this authority, these actors do not have any additional special access or control within the system.

Examples

The Dai Stablecoin System can be used by anyone without any restrictions or sign-up process.

- **Example 1:** Bob needs a loan, so he decides to generate 100 Dai. He locks an amount of ETH worth significantly more than 100 Dai into a CDP and uses it to generate 100 Dai. The 100 Dai is instantly sent directly to his Ethereum account. Assuming that the Stability Fee is 1% per year, Bob will need 101 Dai to cover the CDP if he decides to retrieve his ETH one year later.

One of the primary use cases of CDPs is margin trading by CDP users.

- **Example 2:** Bob wishes to go margin long on the ETH/Dai pair, so he generates 100 USD worth of Dai by posting 150 USD worth of ETH to a CDP. He then buys another 100 USD worth of ETH with his newly generated Dai, giving him a net 1.66x ETH/USD exposure. He's free to do whatever he wants with the 100 USD worth of ETH he obtained by selling the Dai. The original ETH collateral (150 USD worth) remains locked in the CDP until the debt plus the Stability Fee is covered.

Although CDPs are not fungible with each other, the ownership of a CDP is transferable. This allows CDPs to be used in smart contracts that perform more complex methods of Dai generation (for example, involving more than one actor).

- **Example 3:** Alice and Bob collaborate using an Ethereum OTC contract to issue 100 USD worth of Dai backed by ETH. Alice contributes 50 USD worth of ETH, while Bob contributes 100 USD worth. The OTC contract takes the funds and creates a CDP, thus generating 100 USD worth of Dai. The newly generated Dai are automatically sent to Bob. From Bob's point of view, he is buying 100 USD worth of Dai by paying the equivalent value in ETH. The contract then transfers ownership of the CDP to Alice. She ends up with 100 USD worth of debt (denominated in Dai) and 150 USD worth of collateral (denominated in ETH). Since she started with only 50 USD worth of ETH, she is now 3x leveraged long ETH/USD.

Liquidations ensure that in the event of a price crash of the collateral backing a CDP type, the system will automatically be able to close CDPs that become too risky. This ensures that the outstanding Dai supply remains fully collateralized.

- **Example 4:** Let's assume that there is an Ether CDP type with a Liquidation Ratio of 145%, a Penalty Ratio of 105%, and we have an Ether CDP with a collateral-to-debt ratio of 150%. The Ether price now crashes 10% against the Target Price, causing the collateral-to-debt ratio of the CDP to fall to ~135%. As it falls below the Liquidation Ratio, traders can trigger its Liquidation and begin bidding with Dai for buying MKR in the debt auction. Simultaneously, traders can begin bidding with Dai for buying the ~135 Dai worth of collateral in the collateral auction. Once there is at least 105 Dai being bid on the Ether collateral, traders reverse bid to take the least amount of collateral for 105 Dai. Any remaining collateral is returned to the CDP owner.

Addressable Market

As mentioned in the introduction, a cryptocurrency with price stability is a basic requirement for the majority of decentralized applications. As such, the potential market for Dai is at least as large as that of the entire blockchain industry. The following is a short, non-exhaustive list of some of the immediate markets (in both the blockchain and the wider industry) for the Dai Stablecoin System in its capacity as a cryptocurrency with price stability and its use case as a decentralized margin trading platform:

- **Prediction Markets & Gambling Applications:** When making an unrelated prediction, it is obvious not to want to increase one's risk by placing the bet using a volatile cryptocurrency. Long term bets become especially infeasible if the user has to also gamble on the future price of the volatile asset used to place the bet. Instead, a cryptocurrency with price stability like Dai will be the natural choice for prediction market and gambling users.
- **Financial Markets; Hedging, Derivatives, Leverage:** CDPs will allow for permissionless leveraged trading. Dai will also be useful as stable and reliable collateral in custom derivative smart contracts, such as options or CFD's.
- **Merchant receipts, Cross-border transactions and remittances:** Foreign exchange volatility mitigation and a lack of intermediaries means the transaction costs of international trade can be significantly reduced by using Dai.
- **Transparent accounting systems:** Charities, NGO's and Governments will all see increases in efficiency and lower levels of corruption by utilizing Dai.

Risks and their Mitigation

There are many potential risks facing the successful development, deployment, and operation of the Maker Platform. It is vital that the Maker community takes all necessary steps to mitigate these risks. The following is a list spells out some of the risks identified and the accompanying plan for risk mitigation:

Malicious hacking attack against the smart contract infrastructure

The greatest risk to the system during its early stages is the risk of a malicious programmer finding an exploit in the deployed smart contracts, and using it to break or steal from the system before the vulnerability can be fixed. In a worst case scenario, all decentralized digital assets that are held as collateral in The Maker Platform, such as Ether (ETH) or Augur Reputation (REP), could be stolen without any chance of recovery. *The part of the collateral portfolio that is not decentralized, such as Digix Gold IOU's, would not be stolen in such an event as they can be frozen and controlled through a centralized backdoor.*

Mitigation: Smart contract security and best security practices have been the absolute highest priority of the Dai development effort since its inception. The codebase has already undergone three independent security audits by some of the best security researchers in the blockchain industry.

In the very long term, the risk of getting hacked can theoretically be almost completely mitigated through formal verification of the code. This means mathematically proving that the code does exactly what it is intended to do. While complete formal verification is a very long term goal, significant work towards it has already been completed, including a full reference implementation of the Dai Stablecoin System in the functional programming language Haskell, which serves as a stepping stone towards more sophisticated formalizations that are currently under active research and development

Black swan event in one or more collateral assets

Another high impact risk is a potential Black Swan event on collateral used for the Dai. This could either happen in the early stages of Dai Stablecoin System, before MKR is robust enough to support inflationary dilutions, or after the Dai Stablecoin System supports a diverse portfolio of collateral.

Mitigation: CDP collateral will be limited to ETH in the early stages, with the debt ceiling initially limited and growing gradually over time.

Competition and the importance of ease-of-use

As mentioned previously, there is a large amount of money and brainpower working on cryptocurrency with price stability. By virtue of having “true decentralization”, the Dai Stablecoin System is by far the most complex model being contemplated in the blockchain industry. A perceived risk is a movement among cryptocurrency users where the ideals of decentralization are exchanged for the simplicity and marketing of centralized digital assets.

Mitigation: We expect that Dai will be very easy to use for a regular cryptocurrency user. Dai will be a standard Ethereum token adhering to the ERC-20 standard and will be readily available with high liquidity across the ecosystem. Dai has been designed in such a way that the average user need not understand the underlying mechanics of the system in order to use it.

The complexities of the Dai Stablecoin System will need to be understood primarily by Keepers and capital investment companies that use the Dai Stablecoin System for margin trading. These types of users have enough resources to onboard themselves as long as there is abundant and clear documentation of every aspect of the system's mechanics. The Maker community will ensure that this is the case.

Pricing errors, irrationality and unforeseen events

A number of unforeseen events could potentially occur, such as a problem with the price feed from the Oracles, or irrational market dynamics that cause variation in the value of Dai for an extended period of time. If confidence is lost in the system, the TRFM adjustments or even MKR dilution could reach extreme levels while still not bringing enough liquidity and stability to the market.

Mitigation: The Maker community will need to incentivize a sufficiently large capital pool to act as Keepers of the market in order to maximize rationality and market efficiency and allow the Dai supply to grow at a steady pace without major market shocks.

Failure of centralized infrastructure

The Maker Team plays a major role in the development and governance of the Maker Platform in its early days: budgeting for expenses, hiring new developers, seeking partnerships and institutional users, and interfacing with regulators and other key external stakeholders. Should the Maker Team fail in some capacity — for legal reasons, or due to internal problems with management — the future of Maker could be at risk without a proper backup plan.

Mitigation: The Maker community exists partly to act as the decentralized counterparty to the Maker Team. It is a loose collective of independent actors who are all aligned by holding the MKR token, giving them a strong incentive to see the Maker Platform succeed. During the early phases of MKR distribution, great care was taken to ensure that the most important core developers received a significant MKR stake. In the event that the Maker Team is no longer effectively able to lead the development of the Maker Platform, individual MKR holders will be incentivized to fund developers (or simply carry out development themselves) in an effort to protect their investment.

Conclusion

The Dai Stablecoin System was designed to solve the crucial problem of stable exchange of value in the Ethereum ecosystem and the wider blockchain economy. We believe that the mechanism through which Dai is created, transacted, and retired, along with the direct Risk Management role of MKR holders, will allow for self-interested Keepers to maintain the price stability of Dai over time in an efficient manner. The founders of the Maker community have established a prudent governance roadmap that is appropriate for the needs of agile development in the short term, but also coherent with the ideals of decentralization over time. The development roadmap is aggressive and focused on widespread adoption of Dai in a responsible fashion.

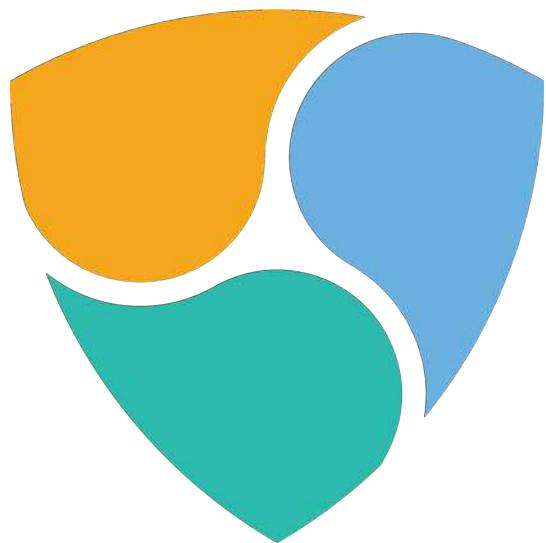
Glossary of Terms

- **Collateralized Debt Position (CDP):** A smart contract whose users receive an asset (Dai), which effectively operates as a debt instrument with an interest rate. The CDP user has posted collateral in excess of the value of the loan in order to guarantee their debt position.
- **Dai:** The cryptocurrency with price stability that is the asset of exchange in the Dai Stablecoin System. It is a standard Ethereum token adhering to the ERC20 standard.
- **Debt Auction:** The reverse auction selling MKR for Dai to cover Emergency Debt when a CDP becomes undercollateralized.
- **Collateral Auction:** The auction selling collateral from a CDP undergoing liquidation. It is designed to prioritize covering the debt owed by the CDP, and secondarily to give the CDP owner the best possible price for their excess collateral refund.
- **The Dai Foundation:** A decentralized team of smart contract developers committed to the development and successful launch of the Maker Platform.
- **Keepers:** Independent economic actors that trade Dai, CDPs and/or MKR; create Dai or close CDPs; and seek arbitrage on The Dai Stablecoin System. As a result, Keepers help maintain Dai market rationality and price stability.
- **MKR:** The ERC20 token used by MKR voters for voting. It also serves as a backstop in the case of insolvent CDPs.
- **MKR Voters:** MKR holders who actively manage the risk of the Dai Stablecoin System by voting on Risk Parameters.
- **Maker:** The name of the Decentralized Autonomous Organization that is made up of the Maker Platform technical infrastructure, and the community of MKR voters.

- **Oracles:** Ethereum accounts (either contracts or users) selected to provide price feeds into various components of Maker Platform.
- **Risk Parameters:** The variables that determine (among other things) when the Maker Platform automatically judges a CDP to be Risky, allowing Keepers to liquidate it.
- **Sensitivity Parameter:** The variable that determines how aggressively the Dai Stablecoin System automatically changes the Target Rate in response to Dai market price deviations.
- **Target Rate Feedback Mechanism (TRFM):** The automatic mechanism by which the Dai Stablecoin System adjusts the Target Rate in order to cause market forces to maintain stability of the Dai market price around the Target Price.

Links

- **Chat:** <https://chat.makerdao.com/> — Primary platform of community interaction
- **Forum:** <https://forum.makerdao.com/> — For debate and proposals
- **Subreddit:** <https://reddit.com/r/makerdao/> — Best place to get latest news and links
- **GitHub:** <https://github.com/makerdao/> — Repository of the public Maker code
- **TeamSpeak:** <https://ts.makerdao.com/> — For governance meeting conference calls
- **SoundCloud:** <https://soundcloud.com/makerdao/> — Governance meeting recordings
- **Oasis:** <https://oasisdex.com/> — MKR and Dai decentralized exchange
- **Sai:** <https://sai.makerdao.com/> — Experimental stablecoin



NEM

Technical Reference

Version 1.2.1

February 23, 2018

Contents

Preface	iii
1 Introduction	1
2 Accounts and Addresses	2
2.1 Account state	2
2.2 NEM addresses	3
2.3 Converting a public key into an address	4
2.4 Intentional address collision	6
3 Cryptography	7
3.1 Private and public key	7
3.2 Signing and verification of a signature	8
3.3 Encoding and decoding messages	8
4 Transactions	10
4.1 Transfer transactions	10
4.2 Importance transfer transactions	11
4.2.1 Activating	11
4.2.2 Deactivating	11
4.3 Multisig related transaction types	11
4.3.1 Aggregate modification transactions (multisig modification)	12
4.3.2 Multisig signature transactions	12
4.3.3 Multisig transactions	13
4.4 Unconfirmed transactions (spam filter)	13
5 Blocks and the block chain	16
5.1 Block difficulty	16
5.2 Block score	18
5.3 Block creation	18
5.4 Block chain synchronization	19
6 A reputation system for nodes	21

6.1	Node interactions	21
6.2	Local trust value	21
6.3	Aggregating local trust values	22
6.4	Enhancing the algorithm	23
6.5	Benefits of the reputation system	24
7	Proof-of-Importance	26
7.1	Eligibility for Entering the Importance Calculation	26
7.2	The outlink matrix	27
7.3	NCDawareRank	30
7.4	Clustering the transaction graph	32
7.5	Calculating Importance Scores	34
7.6	Resistance to Manipulation	35
7.6.1	Sybil Attack	35
7.6.2	Loop Attack	37
7.7	Nothing-at-Stake Problem	39
7.8	Comparing importance to stake	39
8	Time synchronization	44
8.1	Gathering samples	44
8.2	Applying filters to remove bad data	45
8.3	Calculation of the effective offset	46
8.4	Coupling and threshold	47
9	Network	49
9.1	Node Protocol	49
9.2	Node Startup	50
9.3	Node Discovery	50
9.3.1	Announcement	50
9.3.2	Refresh	51
9.4	Node Selection	51

Preface

“

”

You miss 100% of the shots you don't take.

- Wayne Gretzky

NEM is a movement that aims to empower individuals by creating a new economy based on the principles of decentralization, financial freedom, and equality of opportunity.

We would like to thank the contributors and the many people who have inspired us...

BloodyRookie gimre Jaguar0625 Makoto

1 Introduction

“ He'd say hello and introduce himself, but most of the cats turned a deaf ear, ,
pretending they couldn't hear him, or stare right through him.

- Haruki Murakami



EM, in its most basic form, is a crypto currency that is built on block chain technology. The NEM block chain is an improvement on existing block chain technologies. It integrates concepts from other cryptocurrencies (e.g. Bitcoin) and academic research in network theory.

NEM's primary contribution to the crypto currency landscape is a new consensus mechanism called Proof of Importance (PoI). Unlike Proof of Work (PoW), it is environmentally sustainable and does not require large scale computing resources in perpetuity. PoI is similar to Proof of Stake (PoS) except that it is not solely derived from the size of an account's balance. It incorporates other behaviors that are believed to be positive for the holistic economy. In this way, it attempts to reward active economy participants at the expense of inactive ones and dampens the rich getting richer effect that is inherent to PoS.

NEM's vision is to be the foundation of a vibrant crypto currency ecosystem that emphasizes security and trustless computing. NEM was launched with built-in support for multisig transactions and encrypted messages. Additionally, the peer-to-peer (P2P) NEM network implements a modified version of Eigentrust++ to identify and minimize the impact of malicious nodes.

NEM is evolving and this is just the beginning. Stay tuned for more things to come.

2 Accounts and Addresses

“

No wind serves him who addresses his voyage to no certain port.

- Michel de Montaigne

”



NEM uses elliptic curve cryptography to ensure confidentiality, authenticity and non-repudiability of all transactions. Each account is a private+public Ed25519 keypair ([section 3: Cryptography](#)) and is associated with a mutable state that is updated when transactions are accepted by the network. Accounts are identified by NEM addresses, which are derived in part from one way mutations of Ed25519 public keys.

2.1 Account state

The state associated with each account includes the following items:

- account balance
- number of harvested blocks (see [subsection 5.3: Block creation](#))
- height of the first transaction that referenced the account
- list of multisig accounts and list of cosignatories (see [subsection 4.3: Multisig related transaction types](#))
- information about delegated account status (see [subsection 4.2: Importance transfer transactions](#))
- importance and NCD aware rank (see [section 7: Proof-of-Importance](#))
- vested balance (crucial for PoI and NEM itself)

The underlying crypto currency of the NEM network is called XEM. Each account's XEM balance is split into two parts: vested and unvested.

Whenever an account receives XEM, the new XEM are added to the account's unvested balance. When an account sends XEM, XEMs are taken from both the vested and the unvested balance, to retain the vested to unvested ratio¹. Additionally, every 1440 blocks, $\frac{1}{10}$ of the unvested balance is moved to the vested part.

¹Of course that is not always possible

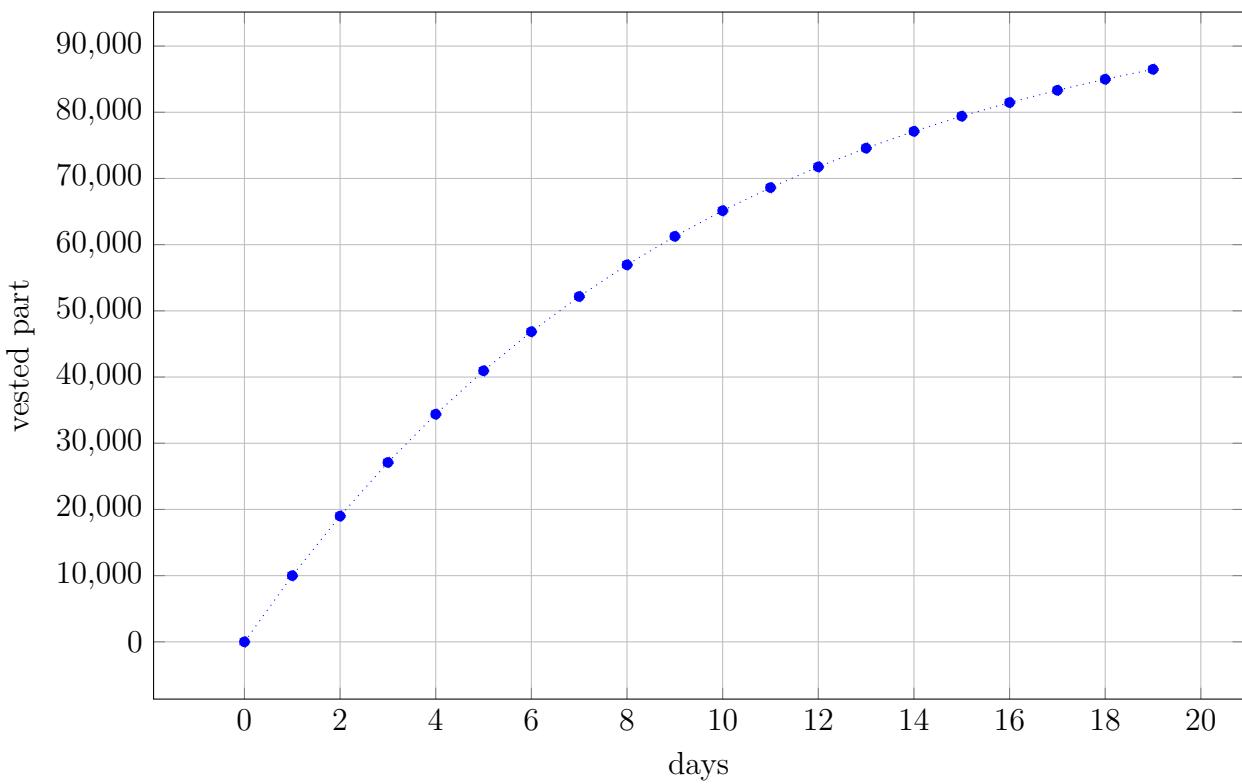


Figure 1: Vesting of 100,000 XEM

All accounts in the nemesis block² are fully vested.

2.2 NEM addresses

A *NEM address* is a base-32³ encoded triplet consisting of:

- network byte
- 160-bit hash of the account's public key
- 4 byte checksum

The checksum allows for quick recognition of mistyped addresses. It is possible to send XEM to any valid address even if the address has not previously participated in any

²first block in the NEM block chain

³<http://en.wikipedia.org/wiki/Base32>

transaction. If nobody owns the private key of the account to which the XEM is sent, the XEM is most likely lost forever.

2.3 Converting a public key into an address

In order to convert a public key to an address, the following steps are performed:

1. Perform 256-bit Sha3 on the public key
2. Perform 160-bit Ripemd of hash resulting from step 1.
3. Prepend version byte to Ripemd hash (either 0x68 or 0x98)
4. Perform 256-bit Sha3 on the result, take the first four bytes as a checksum
5. Concatenate output of step 3 and the checksum from step 4
6. Encode result using base32

Example:

1. public key
X: deb73ed7d0334e983701feba4599a37fb62e862e45368525b8d9fb9ab80aa57e
Y: 169318abc3e5b002059a396d4cf1c3d35ba022c675b15fb1c4943f7662eef268
Z: a90573bd221a3ae33fec5d4efc4fa137897a40347eeafe87bee5d67ae5b4f725
2. compressed public key:
c5247738c3a510fb6c11413331d8a47764f6e78ffcdb02b6878d5dd3b77f38ed
3. sha3-256:
70c9dcf696b2ad92dbb9b52ceb33ec0eda5bfdb7052df4914c0919caddb9dfcf
4. ripemd: 1f142c5ea4853063ed6dc3c13aaa8257cd7daf11
5. prepend version: 681f142c5ea4853063ed6dc3c13aaa8257cd7daf11
6. sha3-256 of above:
09132a5ea90ab7fa077847a699b4199691b4130f66876254eadd70ae459dbb53
7. 4-byte checksum: 09132a5e (first 4 bytes of the above)
8. binary address: 681f142c5ea4853063ed6dc3c13aaa8257cd7daf1109132a5e
9. base-32 encoding: NAPRILC6USCTAY7NNXB4COVKQJL427NPCEERGKS6
10. pretty-print: NAPRIL-C6USCT-AY7NNX-B4COVK-QJL427-NPCEER-GKS6

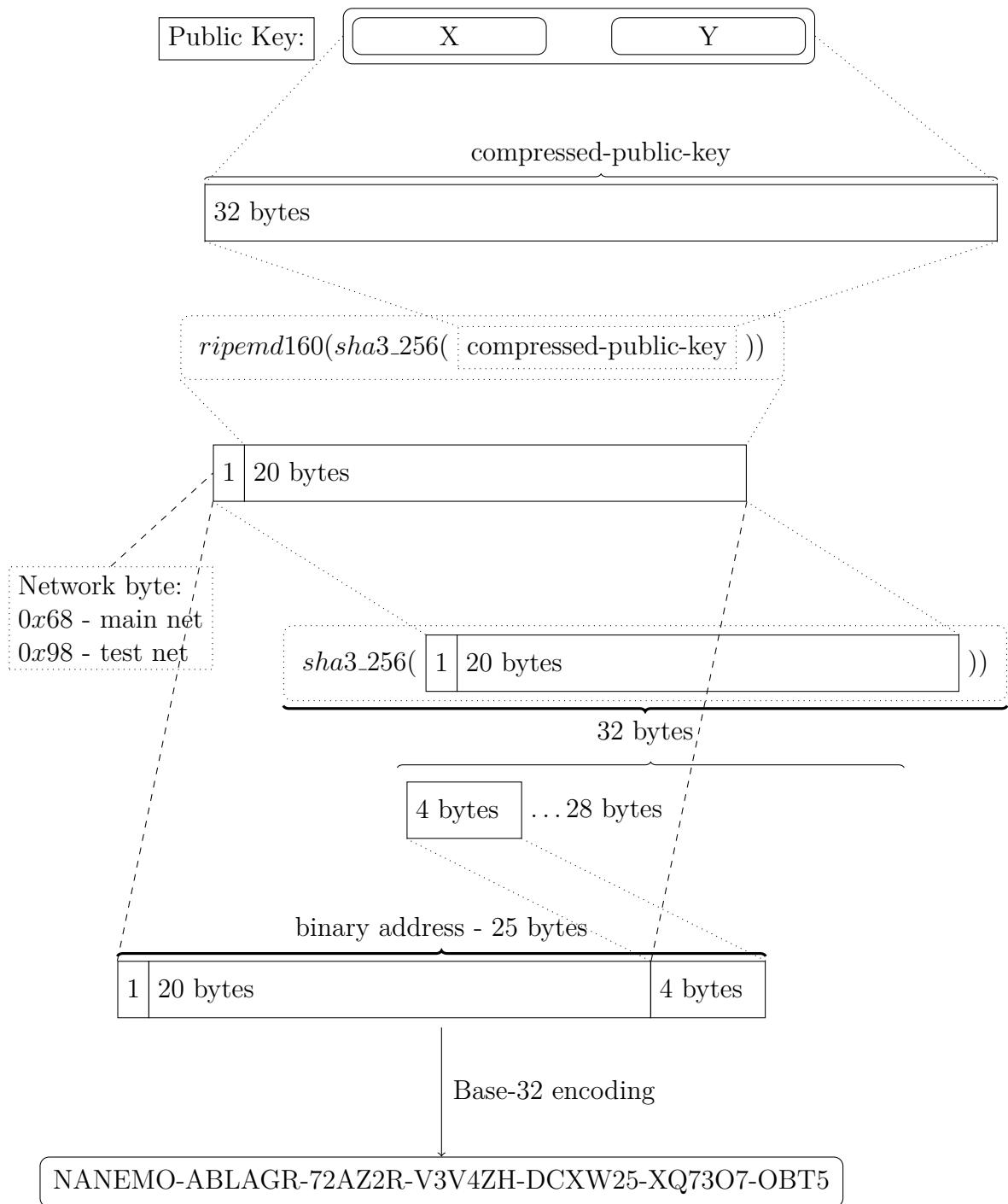


Figure 2: Address generation

2.4 Intentional address collision

It is possible that two different public keys will yield the same address. If such an address contains XEM it would be possible for an attacker to withdraw funds from such account.

In order for the attack to succeed, the attacker would need to find a private+public keypair such that the sha3_256 of the public key would **at the same time** be equal to the ripemd-160 preimage of 160-bit hash mentioned above. Since sha3_256 offers 128 bits of security, it's mathematically improbable for a single sha3_256 collision to be found. Due to similarities between NEM addresses and Bitcoin addresses, the probability of causing a NEM address collision is roughly the same as that of causing a Bitcoin address collision.

3 “ Cryptography

I understood the importance in principle of public key cryptography but it's all moved much faster than I expected. I did not expect it to be a mainstay of advanced communications technology. ”

- Whitfield Diffie



LOCK chain technology demands the use of some cryptographic concepts. NEM, like many other crypto currencies, is using cryptography based on Elliptic Curve Cryptography. The choice of the underlying curve is important in order to guarantee security and speed.

NEM has chosen to use the *Twisted Edwards curve*:

$$-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$$

over the finite field defined by the prime number $2^{255} - 19$ together with the digital signature algorithm called Ed25519. It was developed by D. J. Bernstein et al. and is one of the safest and fastest digital signature algorithms [2].

The base point for the corresponding group G is called B. The group has $q = 2^{252} - 27742317773723535851937790883648493$ elements. Every group element A can be encoded into a 256 bit integer A which can also be interpreted as 256-bit string and A can be decoded to receive A again. For details see [2].

For the hash function H mentioned in the paper, NEM uses the 512 bit SHA3 hash function.

3.1 Private and public key

The *private key* is a random 256-bit integer k . To derive the public key A from it, the following steps are taken:

$$H(k) = (h_0, h_1, \dots, h_{511}) \quad (1)$$

$$a = 2^{254} + \sum_{3 \leq i \leq 253} 2^i h_i \quad (2)$$

$$A = aB \quad (3)$$

Since A is a group element it can be encoded into a 256-bit integer A which serves as the public key.

3.2 Signing and verification of a signature

Given a message M , private key k and its associated public key \underline{A} , the following steps are taken to create a signature:

$$H(k) = (h_0, h_1, \dots, h_{511}) \quad (4)$$

$$r = H(h_{256}, \dots, h_{511}, M) \text{ where the comma means concatenation.} \quad (5)$$

$$R = rB \quad (6)$$

$$S = (r + H(\underline{R}, \underline{A}, M)a) \bmod q \quad (7)$$

Then $(\underline{R}, \underline{S})$ is the *signature* for the message M under the private key k . Note that only signatures where $S < q$ and $S > 0$ are considered as valid **to prevent** the problem of *signature malleability*.

To verify the signature $(\underline{R}, \underline{S})$ for the given message M and public key \underline{A} one checks $S < q$ and $S > 0$ and then calculates

$$\tilde{R} = SB - H(\underline{R}, \underline{A}, M)A$$

and verifies that

$$\tilde{R} = R \quad (8)$$

If S was computed as shown in (7) then

$$SB = rB + (H(\underline{R}, \underline{A}, M)a)B = R + H(\underline{R}, \underline{A}, M)A$$

so (8) will hold.

3.3 Encoding and decoding messages

NEM uses Bouncy Castle's AES block cipher implementation in CBC mode⁴ to encrypt and decrypt messages.

If Alice has the private key k_A and wants to encrypt a message for Bob who has the public key \underline{A}_B (with corresponding group element A_B) then the shared secret used when setting up the cipher is calculated as follows:

a_A is computed from k_A according to (2)

salt = 32 random bytes

$$G = a_A A_B$$

$$\text{shared secret} = \tilde{H}(G \vee \text{salt})$$

⁴http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#CBC

where \tilde{H} is the 256-bit SHA3 hash function.

Another 16 random bytes are used as IV data. Thus, the encrypted message payload consists of

1. the salt
2. the IV data
3. the encrypted message block

Decryption works in a similar manner. Bob has to know Alice's public key \underline{A}_A (and his own private key k_B) and the salt to derive the shared secret:

a_B is computed from k_B according to (2)

$$G = a_B \underline{A}_A$$

$$\text{shared secret} = \tilde{H}(G \vee \text{salt})$$

Supplying the shared secret and the IV data to the cipher engine decrypts the encoded message.

4 Transactions

To transact business with the girl who ran the gas-pump Dean merely threw on his T-shirt like a scarf and was curt and abrupt as usual and got back in the car and off we roared again.

- Jack Kerouac



TRANSACTIONS introduce dynamism into a cryptocurrency system. They are the only way of altering the state of an account. A newly created transaction that has not yet been included in a block is called an *unconfirmed transaction*. Unconfirmed transactions are not guaranteed to be included in any block. As a result, unconfirmed transactions have no effect on the account state. The account state is only updated when a transaction is included in a harvested block and thereby confirmed.

Different types of transactions exist. Each type has a specific purpose, e.g. transfer XEM from one account to another or convert an account to a multisig account. Since transactions consume resources of the p2p network there is a fee for each transaction. The fee depends on the transaction type and other parameters of the transaction.

Transactions have a deadline. If a transaction is not included in a block before its deadline, the transaction is considered expired and gets dropped by the network nodes.

The following sections describe the different transaction types.

4.1 Transfer transactions

A *transfer transaction* is used to transfer XEM from one account to another. A message of at most 1024 bytes can be attached to each transfer transaction. In the case of an encrypted message, only 960 bytes can contain custom data because the salt and the IV data are part of the encrypted message.

Fees for transfer transactions are divided into two parts:

- .05 XEM per 10,000 XEM transferred, capped at 1.25 XEM
- .05 XEM per commenced 32 message bytes ($\text{messageLength} / 32 + 1$).

Both fee parts are added to give the final fee.

4.2 Importance transfer transactions

NEM allows an account to lease its harvesting power to another account through an *importance transfer transaction*. This is known as *delegated harvesting*. This allows the original account to use its importance to harvest on a remote server (such as a virtual private server (VPS)) without needing to have its private key openly exposed on the server. In fact, this feature allows accounts in cold storage to harvest without putting any funds at risk.

The fee for an importance transfer transaction is .15 XEM.

4.2.1 Activating

An account can enable delegated harvesting by sending a special importance transfer transaction that specifies the delegated account.

After the importance transfer transaction is accepted by the network, **360 confirmations** are needed before delegated harvesting activation is complete.

During the activation period, only the original account can harvest but the delegated account cannot. After the activation period, only the delegated account can harvest but the original account cannot.

4.2.2 Deactivating

An account with delegated harvesting activated can disable delegated harvesting at any time by sending a special importance transfer transaction that specifies the delegated account.

After the importance transfer transaction is accepted by the network, **360 confirmations** are needed before delegated harvesting deactivation is complete.

During the deactivation period, only the delegated account can harvest but the original account cannot. After the deactivation period, only the original account can harvest but the delegated account cannot.

4.3 Multisig related transaction types

NEM natively supports m-of-n multisignature accounts.

NEM multisig transactions have been designed with flexibility in mind. Any other

transaction (as of now: importance transfer, transfer, aggregate modification), can be wrapped in a multisig transaction.

A Multisig transaction itself cannot be wrapped inside another *multisig transaction*.

4.3.1 Aggregate modification transactions (multisig modification)

Creating a multisig account An account can be converted into a *multisig account* by sending a special *aggregate modification transaction* that is signed by the multisig account. The transaction specifies information about the cosignatories. The number of cosignatories is limited to 32.

Subsequently, no transactions can be initiated from the multisig account.

Modifying a multisig account After a multisig account has been created, a cosignatory can be added or removed by wrapping an aggregate modification transaction in a multisig transaction. A single modification transaction can add one or more cosignatories but can remove at most one.

- When adding cosignatories, all existing cosignatories must co-sign the transaction.
- When removing a cosignatory, all existing cosignatories except for the one being removed must co-sign the transaction.

The fee for an aggregate modification transaction is a flat fee of 0.5 xem, no matter what modifications you make.

where the term 'modification' refers to the addition or deletion of a cosignatory.

4.3.2 Multisig signature transactions

NEM uses multisig signature transactions for letting cosignatories sign a multisig transaction.

The fee for such a signature transaction is always .15 XEM. The fee is deduced from the multisig account, not from the cosignatory's account.

A multisig signature transaction can only be included in the block chain if the corresponding multisig transaction has been included in the block chain. An orphaned multisig signature transaction will never be included in the block chain.

4.3.3 Multisig transactions

As mentioned earlier, any transaction can be wrapped in a *multisig transaction*.

To send XEM from a *multisig account* to another account, a transfer transaction must be wrapped. The multisig wrapper transaction has a fee of .15 XEM. This fee is added to the usual transfer transaction fee.

The following example shows the steps that must be taken in more detail:

Assume that a multisig account (**M**) has a balance of 1000 XEM and has three cosignatories (**A**, **B**, **C**) and **100 XEM** needs to be transferred from **M** to another account **X**.

Any of the three cosignatories can initiate the 100 XEM transfer. Assuming that B initiates the transfer and A and C cosign, the following steps must happen for the transaction to be accepted:

1. B creates a regular, unsigned transfer transaction that has the multisig account as the “signer” and the transfer amount as 100 XEM
2. B wraps the unsigned transfer transaction in a multisig transaction
3. B signs the multisig transaction and sends it to the NEM network
4. A and C are notified of the pending multisig transaction
5. A creates a multisig signature transaction by signing the hash of the unsigned transfer transaction and sends it to the network
6. C creates a multisig signature transaction by signing the hash of the unsigned transfer transaction and sends it to the network
7. Once all cosignatories (B implicitly and A and C explicitly) have signed the unsigned transfer transaction, the transaction is accepted by the network and 100 XEM is transferred from M to X

If A and/or C do not send a multisig signature transaction corresponding to the multisig transfer transaction before the transaction deadline, the multisig transfer transaction will be rejected by the network and no XEM will be transferred from M to X.

4.4 Unconfirmed transactions (spam filter)

When a new transaction is created and delivered to a node, the node

-
1. Puts the transaction into its unconfirmed transaction cache
 2. Broadcasts the transaction to other nodes (if the transaction is valid)

There is also a poll mechanism for unconfirmed transactions during each block chain synchronization round. This allows nodes that do not have their port 7890 open (and therefore cannot receive broadcasts) to learn about new unconfirmed transactions.

Low transaction fees might tempt some bad actor in the network to flood the network with new transactions in order to disturb the network. It is therefore needed to limit the number of unconfirmed transactions which are handled by the nodes.

Simply limiting the number of unconfirmed transactions that a node accepts is bad because normal actors still should be able to send a transaction even when someone is spamming the network. Limiting the number of unconfirmed transactions per account is also not an option since the attacker can create as many accounts as (s)he wants.

NEM does filtering in a smarter way. A custom spam filter decides whether a new unconfirmed transaction should be processed or rejected. It works the following way:

- Consider the unconfirmed transaction cache as having 1000 slots
- As long as less than 120 slots are filled, no transaction is rejected
- Otherwise if there are already fs filled slots and a new unconfirmed transaction with signer A arrives, the fair share of slots for account A in the cache is calculated as

$$\begin{aligned} \text{eff. importance} &= (\text{importance of } A) + \max\left(0.01, \frac{\text{transaction fees}}{100000}\right) \\ \text{fair share} &= (\text{eff. importance}) \cdot \exp\left(-\frac{fs}{300}\right) \frac{1000 (1000 - fs)}{10} \end{aligned}$$

If A occupies less slots than its fair share, the new unconfirmed transaction is accepted.

Note that you can increase the chance that your new transaction is accepted by voluntarily increasing the transaction fees.

[Figure 3](#) shows the fair share of slots versus the fill level of the cache for an effective importance of 1%. An attacker that tries to occupy many slots cannot gain much by using many accounts as the importance of each account will be very low. He could attack by increasing his transaction fees but that will make him expend his funds at a higher rate.

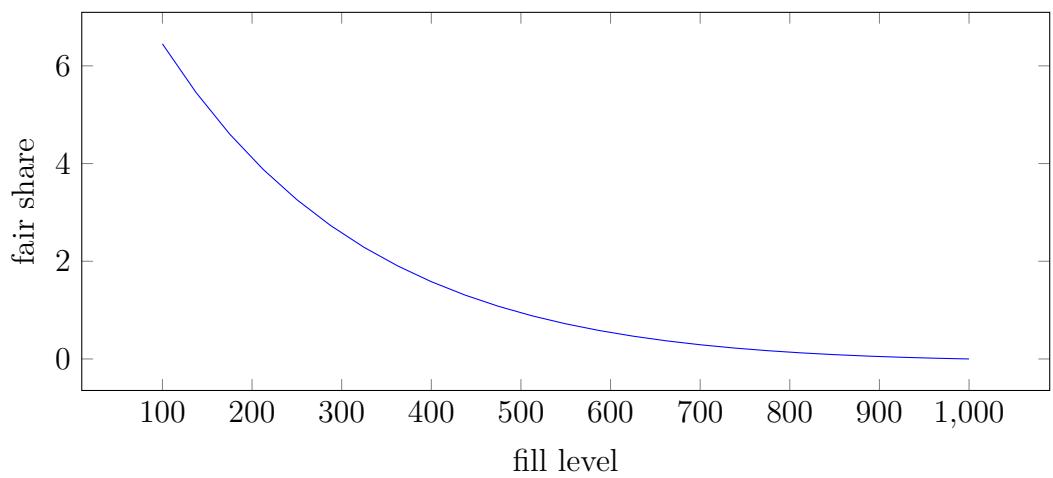


Figure 3: fair share for effective importance of 1‰

5 Blocks and the block chain

If ever I to the moment shall say:
Beautiful moment, do not pass away!
Then you may forge your chains to bind me,

”
- Johann Wolfgang von Goethe



HE central element of every crypto currency is a public ledger called the block chain that links blocks together. Each NEM block can contain up to 120 transactions. Since the blocks in the chain and the transactions in the blocks are ordered, the complete transaction history is held in the block chain. Blocks are stored in a database as permanent medium. NEM calls the first block in the chain the *nemesis block*.

Each block consists of the following parts:

1. the block version
2. the block time stamp
3. the public key of the harvester (block creator)
4. the signature for the block data
5. the previous block hash
6. the generation hash (needed for block creation)
7. the block height
8. the list of transactions

In the following sections, the hash function H always means the 256-bit Sha3 hash function.

5.1 Block difficulty

The difficulty for a new block is calculated from the difficulties and time stamps of the last 60 blocks. If less than 60 blocks are available, only those are taken into account.

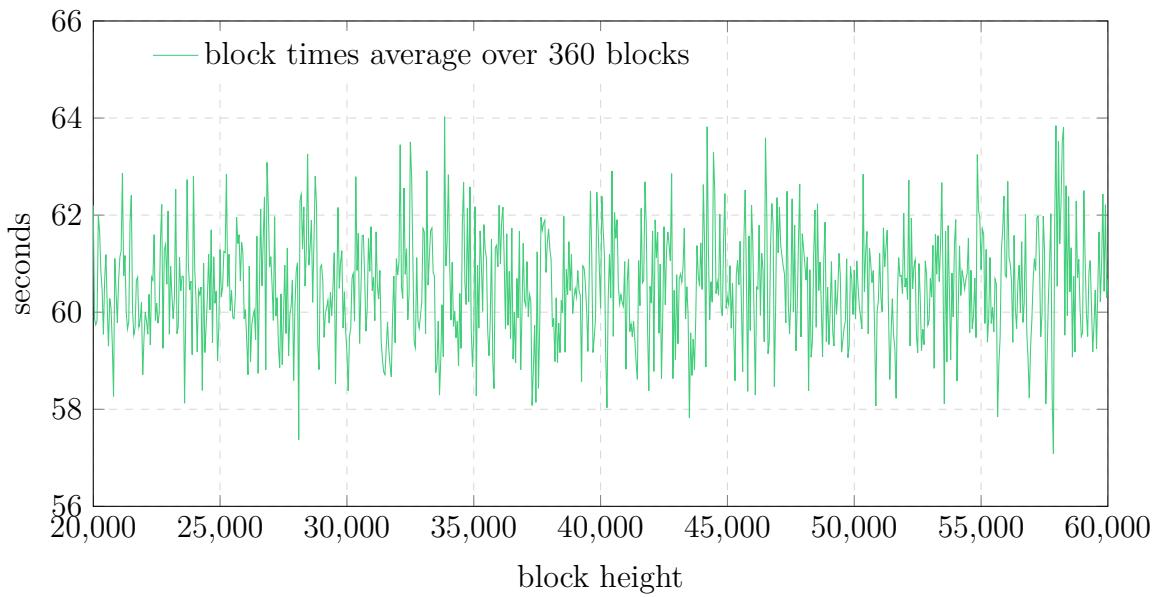


Figure 4: Main net average block times over 360 blocks

If only one block is available, then the block has a predefined *initial difficulty* of 10^{14} . Otherwise, the difficulty is calculated from the last n blocks the following way:

$$d = \frac{1}{n} \sum_{i=1}^n (\text{difficulty of block } i) \quad (\text{average difficulty})$$

$$t = \frac{1}{n} \sum_{i=1}^n (\text{time to create block } i) \quad (\text{average creation time})$$

$$\text{difficulty} = d \frac{60}{t} \quad (\text{new difficulty})$$

If the new difficulty is more than 5% greater or smaller than the difficulty of the last block, then the change is capped to 5%.

Additionally, difficulties are kept within certain bounds. The new difficulty is clamped to the boundaries if it is greater than 10^{15} or smaller than 10^{13} .

Simulations and the NEM beta phase have shown that the algorithm produces blocks with an average time of 60 ± 0.5 seconds.

The slow change rate of 5% makes it hard for an attacker with considerably less than 50% importance to create a better chain in secret since block times will be considerably higher than 60 seconds for the beginning of his secret chain.

5.2 Block score

The score for a block is derived from its difficulty and the time (in seconds) that has elapsed since the last block:

$$score = difficulty - time \text{ elasped since last block} \quad (\text{block score})$$

5.3 Block creation

The process of creating new blocks is called *harvesting*. The harvesting account gets the fees for the transactions in the block. This gives the harvester an incentive to add as many transactions to the block as possible. Any account that has a *vested balance* of at least 10,000 XEM is eligible to harvest.

To check if an account is allowed to create a new block at a specific network time, the following variables are calculated:

$$h = H(\text{generation hash of previous block}, \text{public key of account})$$

interpreted as 256-bit integer

$$t = \text{time in seconds since last block}$$

$$b = 8999999999 \cdot (\text{importance of the account})$$

$$d = \text{difficulty for new block}$$

and from that the *hit* and *target* integer values:

$$hit = 2^{54} \left| \ln \left(\frac{h}{2^{256}} \right) \right|$$

$$target = 2^{64} \frac{b}{d} t$$

The account is allowed to create the new block whenever $hit < target$. In the case of delegated harvesting, the importance of the original account is used instead of the importance of the delegated account.

Since *target* is proportional to the elapsed time, a new block will be created after a certain amount of time even if all accounts are unlucky and generate a very high hit.

Also note that *hit* has an exponential distribution. Therefore, the probability to create a new block does not change if the importance is split among many accounts.

5.4 Block chain synchronization

Since blocks are assigned a score, a score can be assigned to a chain of blocks too:

$$score = \sum_{block \in blocks} block\ score \quad (\text{block chain score})$$

Block chain synchronization is a central task for every block chain based crypto currency. From time to time a local node will ask a remote node about its chain. The remote node is selected using the calculated trust values (see [section 6: A reputation system for nodes](#)).

If the remote node promises a chain with a higher score, the two nodes try to agree on the last common block. If successful, the remote node will supply up to 400 blocks of its chain to the local node.

If the supplied chain is valid, the local node will replace its own chain with the remote chain. If the supplied chain is invalid, the local node will reject the chain and consider the synchronization attempt with the remote node to have failed.

This algorithm will also resolve forks. The last common block may have a height difference of at most 360 compared to the local node's last block. Thus, the maximal depth of forks that can be resolved via the synchronization algorithm is 360.

The flow chart on the next page illustrates the process in more detail.

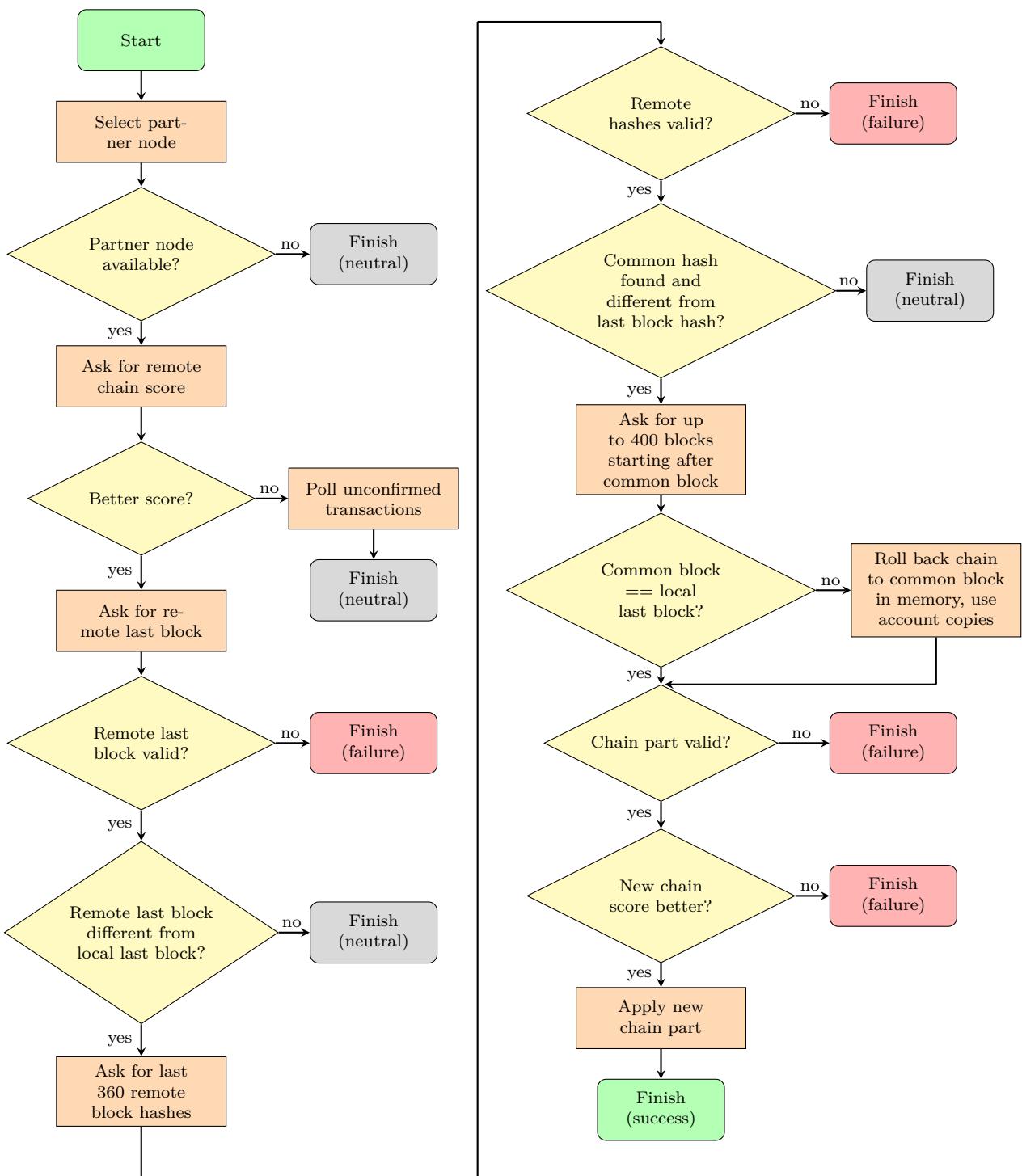


Figure 5: Block chain synchronization flow chart

6 A reputation system for nodes

“

Beauty is something that burns the hand when you touch it.

”

- Yukio Mishima



LIKE other crypto currency networks, the NEM network is a peer-to-peer (P2P) network. P2P networks have the great advantage of being robust because they cannot be shut down by eliminating a single entity. Nevertheless, experience in the last few years has shown that P2P networks also have their share of disadvantages. The participants of the network are anonymous and anyone can join. This makes it very easy to inject hostile nodes into the network that spread invalid information or try to disturb the network in some way.

There is a need to identify hostile nodes and reduce communication with them. There have been many approaches to achieve this. One of the most successful is building a reputation system for nodes. NEM follows this approach by implementing an algorithm similar to the EigenTrust++ reputation system. This section will outline the algorithm used.

For a more detailed discussion, see the original papers about EigenTrust[8] and EigenTrust++[5].

6.1 Node interactions

Nodes in the NEM network interact with each other by sharing information about entities like transactions and blocks of transactions. A node can either broadcast new entities to other nodes or ask other nodes for those entities.

Having received information from a remote node, a node can verify the validity of the information and check if the information is usable. Each node can decide if an interaction (or 'call') should be seen as a success (new valid information was received), neutral (valid, but already known, information was received) or failure (invalid information was received).

Each node i keeps track of the outcomes of all of its interactions with node j in its experience map by counting its successful and failed interactions; $success(i, j)$ and $failed(i, j)$. Neutral interactions are ignored.

6.2 Local trust value

Each node starts with a set P of so called pre-trusted nodes that are supplied in a configuration file and can be edited by the user to his/her own needs. The node then asks

the pre-trusted nodes for other existing nodes in the network. Knowing about n nodes in the network each node can build an initial trust vector \vec{p} by setting:

$$p_j = \begin{cases} \frac{1}{|P|} & \text{if } \text{node}_j \in P \\ 0 & \text{otherwise} \end{cases}$$

p_j indicates how much trust a node initially has in node j .

After some time node i had some interactions with other nodes and can update its local trust values by first calculating:

$$s_{ij} = \begin{cases} \frac{\text{success}(i,j)}{\text{success}(i,j) + \text{failed}(i,j)} & \text{if } \text{success}(i,j) + \text{failed}(i,j) > 0 \\ p_j & \text{otherwise} \end{cases}$$

and then normalizing the local trust values:

$$c_{ij} = \frac{s_{ij}}{\sum_m s_{im}}$$

to define the local trust vector \vec{c}_i with components c_{ij} .

6.3 Aggregating local trust values

From time to time nodes broadcast their local trust values to other nodes. Having received the local trust values from other nodes, node i can calculate an aggregate trust value for node k by weighing the local trust node j has in node k with its own trust in node j :

$$t_{ik} = \sum_j c_{ij} c_{jk}$$

This can be written in matrix notation by defining $C = (c_{kj})$ and \vec{t}_i having components t_{ik} :

$$\vec{t}_i = C^T \vec{c}_i$$

If we define the iteration:

$$\vec{t}_{i+1} = C^T \vec{t}_i$$

then this will converge to the left principal eigenvector \vec{t} of the matrix C under the assumptions that C is irreducible and aperiodic. To guarantee the assumptions being valid, we slightly change the iteration by mixing a portion of the initial trust vector \vec{p} into the iteration:

$$\vec{t}_i = \begin{cases} \vec{p} & \text{if } i = 0 \\ (1 - a)C^T \vec{t}_{i-1} + a\vec{p} & \text{otherwise} \end{cases} \quad (9)$$

where a suitable $0 < a < 1$ is chosen. This iteration will always converge to a vector \vec{t} , which represents the trust a node has in other nodes.

6.4 Enhancing the algorithm

The above algorithm to compute trust suffers from the fact that the veracity of the feedback reported from other nodes is unknown. Malicious nodes could collude and report low trust values for honest nodes and high trust values for dishonest nodes.

An improvement is to estimate the credibility of the feedback of other nodes and weight the reported trust values by the credibility score. To do that, $\text{common}(u, v)$ is defined for two nodes u and v as the set of nodes with which both nodes have interacted. Given that, a measure for the similarity of the feedback of the two nodes ($\text{sim}(u, v)$) can be calculated:

$$\text{sim}(u, v) = \begin{cases} \left(1 - \sqrt{\frac{\sum_{w \in \text{common}(u, v)} (s_{uw} - s_{vw})^2}{|\text{common}(u, v)|}}\right)^b & \text{if } \text{common}(u, v) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

where b is a positive integer (the original paper suggests $b = 1$).

Then, feedback credibility can be defined as:

$$f_{ij} = \begin{cases} \frac{\text{sim}(i, j)}{\sum_m \text{sim}(i, m)} & \text{if } \sum_m \text{sim}(i, m) > 0 \\ 0 & \text{otherwise} \end{cases}$$

and, finally, the matrix $L = (l_{ij})$ can be defined as:

$$l_{ij} = \begin{cases} \frac{f_{ij} c_{ij}}{\sum_m f_{im} c_{im}} & \text{if } \sum_m f_{im} c_{im} > 0 \\ 0 & \text{otherwise} \end{cases}$$

which incorporates both the reported trust values and the feedback credibility.

It is now straightforward to define an iteration analog to [Equation 9](#):

$$\vec{t}_i = \begin{cases} \vec{p} & \text{if } i = 0 \\ (1 - a)L^T \vec{t}_{i-1} + a\vec{p} & \text{otherwise} \end{cases}$$

which converges to the left principal eigenvector of the underlying matrix of the power iteration.

The original Eigentrust++ paper suggests using additional measures to limit trust propagation between honest and dishonest nodes. The paper was written with file sharing networks in mind. In such networks, incomplete data is shared (parts of files) and cannot be checked for validity. Therefore, even honest nodes could distribute malicious data. Nodes in the NEM network always download entities in their entirety and verify their integrity before distributing them to other nodes. NEM network simulations have shown that the results without the additional trust propagation measures are good enough to mitigate the presence of malicious nodes.

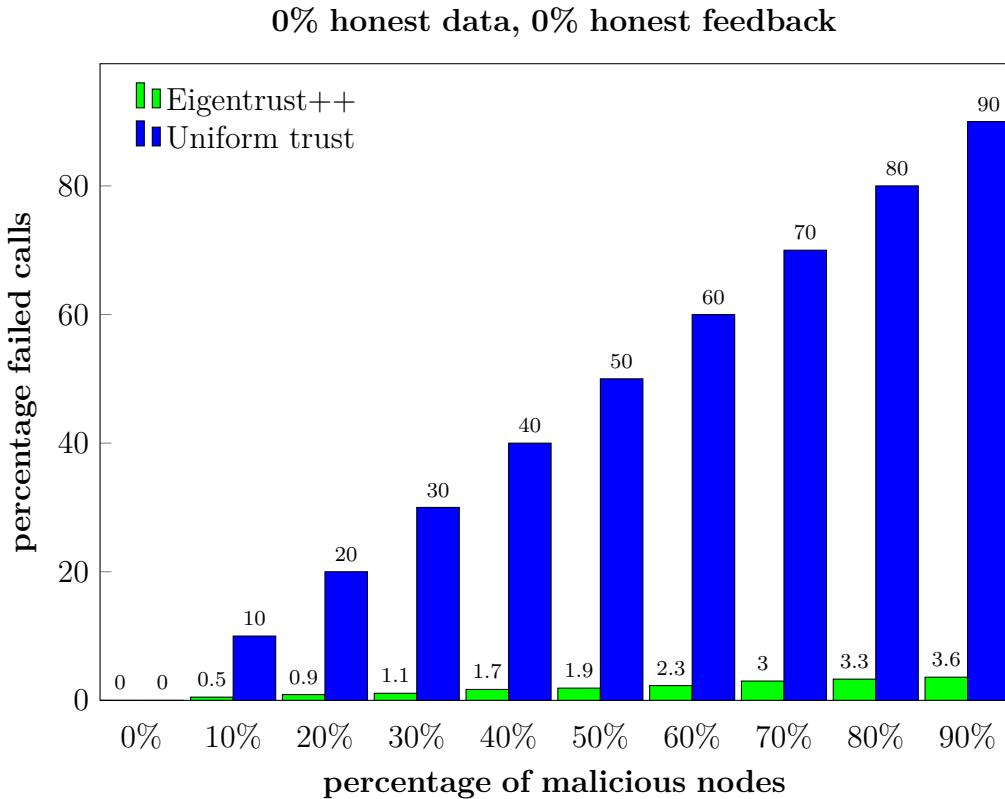


Figure 6: Simulation with attacking nodes that are always dishonest

6.5 Benefits of the reputation system

Having a reputation system for nodes allows nodes to select their communication partner according to the trust values for other nodes. This should also help balance the load of the network because the selection of the communication partner only depends on a node's honesty but not its importance.

Simulations show that the algorithm reduces the number of failed interactions considerably. If malicious nodes only provide dishonest data and dishonest feedback they are easily identified (Figure 6).

But even if the malicious nodes collude to give other malicious nodes a high trust value and provide false data and feedback only to a certain percentage, the trust algorithm still cuts down the percentage of failed interactions (Figure 7).

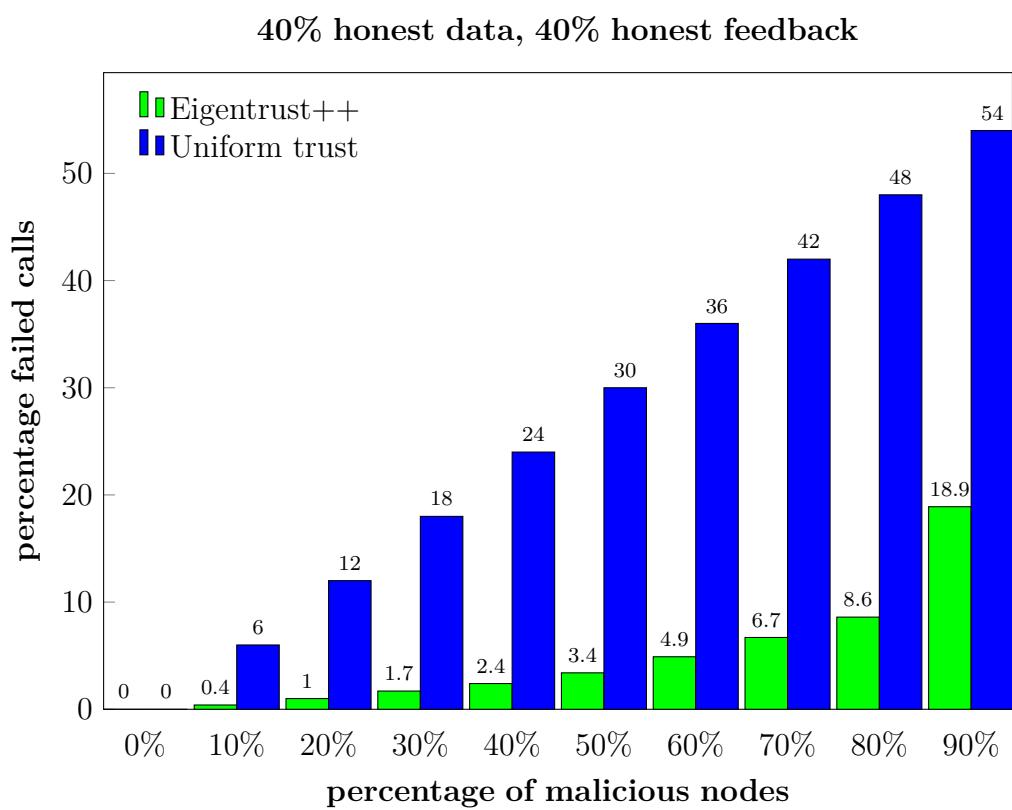


Figure 7: Simulation with attacking nodes that are sometimes dishonest

7 Proof-of-Importance

“

”

It was hot, the night we burned Chrome.

- William Gibson

 ROOF-OF-IMPORTANCE (PoI) is the name of the block chain consensus algorithm used by NEM. Each account is assigned an importance score that proxies its aggregate importance to the NEM economy. Accounts with higher importance scores have higher probabilities of harvesting a block (see [section 5: Blocks and the block chain](#)). Because all transactions are publicly available in NEM, the transaction graph of the NEM economy can be calculated exactly. The topology of the transaction graph can be used as an input into the importance of an account. The insight that the transaction graph can be used for elucidating the importance of an account is the key innovation of Proof-of-Importance.

The NEM block chain platform allows all transactions to be transparently viewed. This information about value transfers between accounts can be used to determine a rating for the importance of accounts. The intuition that not all nodes in a graph have the same salience, or importance, is not new. The literature in the graph theory community is well established for computing the importance of nodes in graphs in the areas of search [11], social networks [1], street networks [7], and neuroscience [6], among others. Building off of this intuition, one of NEM’s core innovations is to use graph theoretic measures as a fundamental input into block chain consensus. The outlink matrix that defines the transaction graph is centrally important and used in the PoI calculation.

7.1 Eligibility for Entering the Importance Calculation

To be eligible for entering the importance calculation, an account must have at least 10,000 vested XEM. All accounts owning more than 10,000 vested XEM have a non-zero importance score.

With a supply of 8,999,999,999 XEM, the theoretical maximum number of accounts with non-zero importance is 899,999. In practice, the number of actual accounts with non-zero importance is not expected to approach the theoretical max due to inequalities in held XEM and also the temporal costs associated with vesting.

If NEM becomes very popular, a threshold of 10,000 vested XEM could be undesirable. If necessary, this number could be updated in the future via a hard fork, which is the same procedure for adjusting transaction fees and other parameters related to harvesting.

7.2 The outlink matrix

Suppose the PoI calculation is done at height h . Accounts that have a vested balance of at least 10,000 XEM at height h are eligible to be part of the PoI calculation. For those accounts, NEM collects all transfer transactions that satisfy the following conditions:

- Transferred an amount of at least 1,000 XEM
- Happened within the last 43,200 blocks (approximately 30 days)
- Recipient is eligible to be part of the PoI calculation too (see [subsection 7.1: Eligibility for Entering the Importance Calculation](#))

For each such transaction T_k that transferred *amount* μ XEM from account A_i to account A_j and happened at height h_{ijk} , a weight is calculated according to the following formula:

$$w_{ijk} = \text{amount} \cdot \exp \left(\ln(0.9) \left[\frac{h - h_{ijk}}{1440} \right] \right)$$

where $[x]$ denotes the floor function. The graph in [Figure 8: amount decay of 10000 XEM](#) illustrates how a transaction amount of 10,000 XEM is weighted over time.

These values are aggregated to

$$\tilde{w}_{ij} = \sum_k w_{ijk}$$

Setting

$$\tilde{o}_{ij} = \begin{cases} \tilde{w}_{ji} - \tilde{w}_{ij} & \text{if } \tilde{w}_{ji} - \tilde{w}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

finally, the outlink matrix \mathbf{O} is comprised of components o_{ij} as

$$o_{ij} = \begin{cases} \frac{\tilde{o}_{ij}}{\sum_i \tilde{o}_{ij}} & \text{if } \sum_i \tilde{o}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The outlink matrix element o_{ij} thus describes the **weighted net flow** (which is set to 0 if negative) of XEM from A_i to A_j during the (approximately) last 30 days. This means that only **net** transfers contribute to an account's importance.

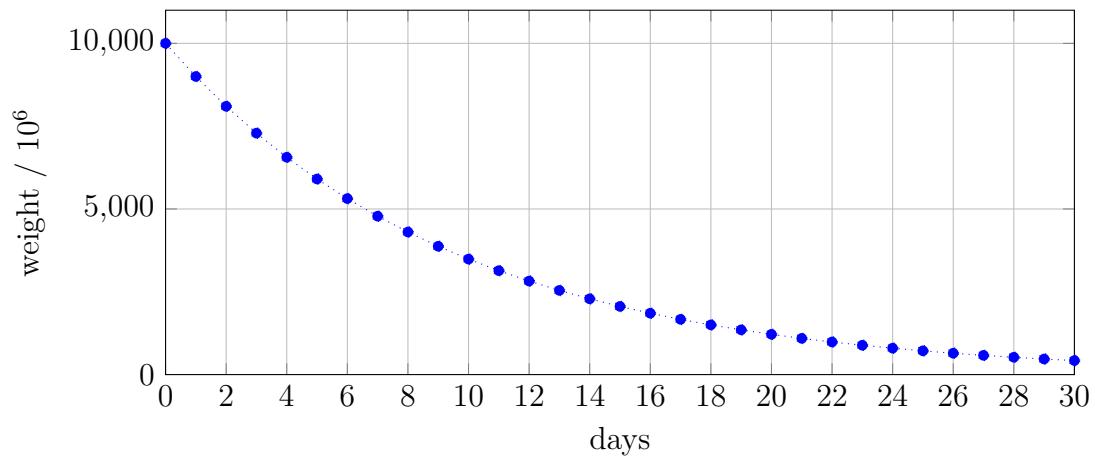


Figure 8: amount decay of 10000 XEM

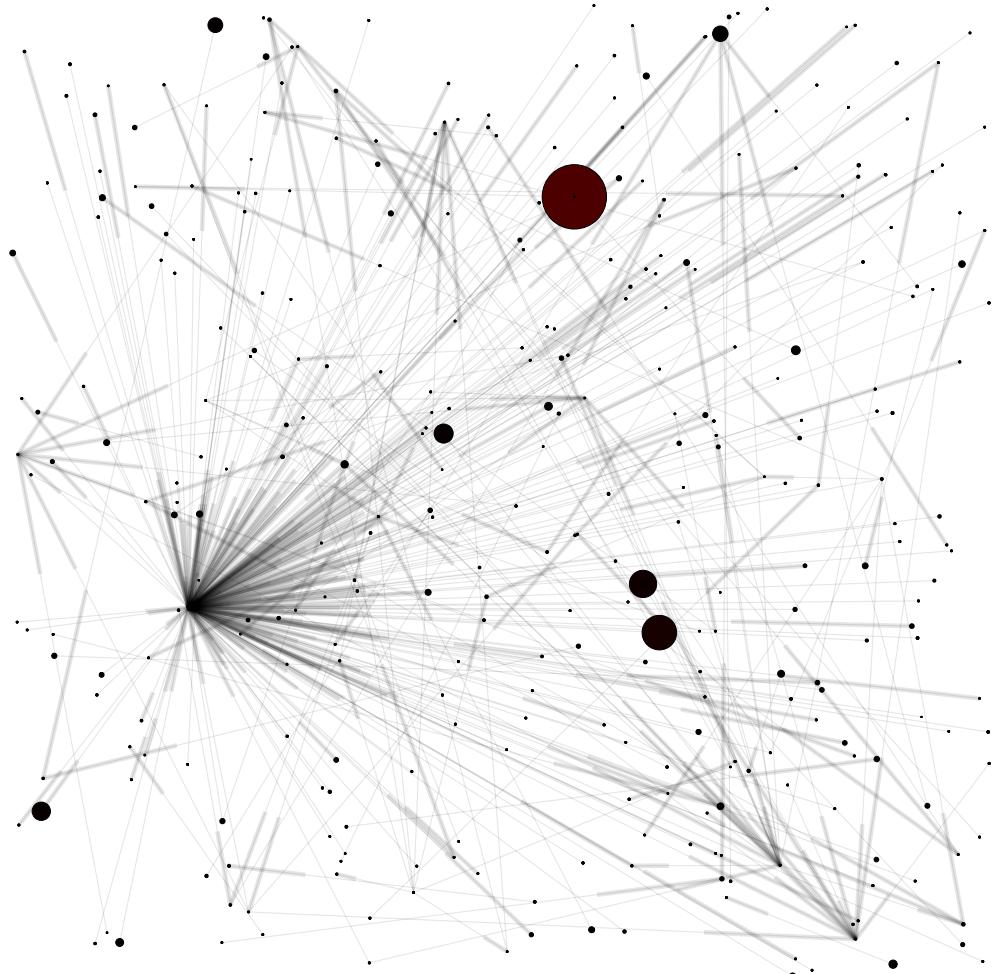


Figure 9: Plot of the NEM transaction graph (outlink matrix) as of 29 April, 2015, containing 1,456 harvesting-eligible accounts.

7.3 NCDawareRank

There are many ways to determine the salience of nodes in a network, and PageRank is one method. NCDawareRank is similar to PageRank, where the stationary probability distribution of an ergodic Markov chain is calculated [9, 11]. NCDawareRank additionally exploits the nearly completely decomposable structure of large-scale graphs of information flows by adding an inter-level proximity matrix as an extra term, \mathbf{M} . The inter-level proximity matrix models the fact that groups of nodes are closely linked together to form clusters that interact with each other. This allows NCDawareRank to converge faster than PageRank while also being more resistant to manipulation of scores, because the rank for nodes within the same level will be limited.

Shown in matrix notation, NCDawareRank is calculated as:

$$\hat{\pi} = \mathbf{O}\eta\pi + \mathbf{M}\mu\pi + \mathbf{E}(1 - \eta - \mu)\pi, \quad (10)$$

where:

\mathbf{O} is the outlink matrix

\mathbf{M} is the inter-level proximity matrix

\mathbf{E} is the teleportation matrix

π is the NCDawareRank

η is the fraction of importance that is given via outlinks

μ is the fraction of importance given to proximal accounts

This definition is the same as for PageRank, only with the addition of \mathbf{M} and μ . For NEM, η is 0.7 and μ is 0.1. The details of how each of these variables is calculated are as follows.

Let W be the set of all harvesting-eligible accounts. For $u \in W$, G_u is the set of accounts that have received more in value transfers from account u than have sent u . Nearly completely decomposable (NCD) partitions of W are defined as $\{A_1, A_2, \dots, A_N\}$, such that for every $u \in W$, there is a unique K such that $u \in A_K$. The proximal accounts of each u , χ_u , are thus defined as:

$$\chi_u \triangleq \bigcup_{w \in (u \cup G_u)} A_{(w)}, \quad (11)$$

and N_u denotes the number of NCD blocks in χ_u .

The definition of the outlink matrix \mathbf{O} is described in [subsection 7.2: The outlink matrix](#). To guarantee convergence, $\mathbf{O}\eta + \mathbf{M}\mu + \mathbf{E}(1 - \eta - \mu)$ must be an irreducible, column-stochastic matrix. This is accomplished by dispersing the rank of dangling accounts (accounts without outlinking value transfers) so that every account has a non-zero teleportation probability.

The inter-level proximity matrix \mathbf{M} is calculated by clustering the transaction graph (described in [subsection 7.4: Clustering the transaction graph](#)) to define each NCD block A_N and then determining the proximal accounts for each cell vu in \mathbf{M} :

$$M_{v,u} \triangleq \begin{cases} \frac{1}{N_u |A_{(v)}|} & \text{if } v \in \chi_u \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

The teleportation matrix \mathbf{E} is calculated as:

$$\mathbf{E} \triangleq \mathbf{e} \mathbf{v}^\top \quad (13)$$

where \mathbf{v}^\top is a teleportation probability vector and \mathbf{e} is the vector with all components set to 1.

Practically, NCDawareRank is calculated via the power iteration method as follows:

$$\begin{aligned} NCDawareRank^r(i) = & (1 - \eta - \mu) \frac{1}{|G|} + \\ & \eta \sum_{k=1}^s o_{ik} NCDawareRank^{r-1}(k) + \\ & \mu \sum_{k=1}^s m_{ik} NCDawareRank^{r-1}(k) \end{aligned} \quad (14)$$

where o_{ki} denotes the k^{th} row for column i in \mathbf{O} and m_{ki} is the k^{th} row for column i in \mathbf{M} . The algorithm continues until the change in NCDawareRank between iterations is less than a specified ε :

$$\left(\sum_{i \in G} |NCDawareRank^r(i) - NCDawareRank^{r-1}(i)| \right) < \varepsilon \quad (15)$$

Teleportation for both the outlink and the inter-level proximity matrices ensures that the transition probability matrix between accounts is stochastic, irreducible, and primitive, so that the algorithm is guaranteed to converge. For more details on the mathematical theory of PageRank, see [9].

As discussed in [10], it is possible to reduce the number of calculations and speed up the calculation of NCDawareRank by decomposing the sparse inter-level proximity matrix \mathbf{M} into two matrices \mathbf{R} and \mathbf{A} :

$$\mathbf{A} \triangleq \begin{bmatrix} e_{|A_1|} & 0 & \cdots & 0 \\ 0 & e_{|A_2|} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e_{|A_N|} \end{bmatrix} \quad (16)$$

$e_{|A_k|}$ is the vector with $|A_k|$ components all set to 1.

$$c = \left(\frac{1}{|A_1|} \quad \frac{1}{|A_2|} \quad \cdots \quad \frac{1}{|A_N|} \right) \quad (17)$$

$$\mathbf{R} \triangleq \left[c_1 \mathbf{R}'_{1*} \quad c_2 \mathbf{R}'_{2*} \quad \cdots \quad c_N \mathbf{R}'_{N*} \right] \quad (18)$$

where \mathbf{R}'_{i*} is the i^{th} row of the matrix \mathbf{R}' which is defined as

$$R_{i,j} \triangleq \begin{cases} \frac{1}{N_u} & \text{if } A_i \in \chi_u \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

The NEM implementation uses the decomposition of \mathbf{M} into \mathbf{A} and \mathbf{R} . See [10] for a thorough discussion of the storage and computational savings of this decomposition.

7.4 Clustering the transaction graph

Clustering is done on the transaction graph using a high-performance implementation [13]⁵ of the SCAN clustering algorithm [14]. In this high-performance implementation, clusters are created by finding nodes that are *core* and then expanding the clusters, calculating the structural similarity between groups of nodes that are two-hops away from each other. Details of the algorithm are as follows.

Assume a graph G of accounts V , where edges E between accounts are defined such that an edge is created if the sum of the decayed value transfers (see subsection 7.2: The outlink matrix) between the accounts is over a predefined threshold of 1,000 XEM. Clustering of accounts in the transaction graph G is done by performing structural similarity-based clustering, which elucidates clusters, hubs, and outliers in the graph. The structural

⁵<http://db-event.jpn.org/deim2014/final/proceedings/D6-2.pdf>

similarity between two accounts u and v in the transaction graph, $\sigma(u, v)$, is calculated as follows:

$$\sigma(u, v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{\sqrt{|\Gamma(u)| |\Gamma(v)|}}, \quad (20)$$

where $|.|$ denotes set cardinality and Γ is the set of structurally connected accounts (inclusive of self), defined as:

$$\Gamma(u) = \{v \in V | \{u, v\} \in E\} \cup \{u\}. \quad (21)$$

N_ϵ is the set of structurally connected accounts that have structural similarity with an account over a pre-determined threshold, ϵ :

$$N_\epsilon(u) = \{v \in \Gamma(u) | \sigma(u, v) \geq \epsilon\}. \quad (22)$$

Core nodes are used for pivoting and expanding clusters and are defined as follows:

$$K_{\epsilon, \mu}(u) \Leftrightarrow |N_\epsilon(u)| \geq \mu \quad (23)$$

where μ is the minimum number of epsilon neighbor accounts that an account must have to be considered *core*. During clustering, clusters are centered (pivoted) around *core* accounts. The initial members of the cluster are the members of N_ϵ . This means that μ controls the smallest possible size of a cluster. In NEM, μ is 4 and ϵ is 0.3. An account v has *direct structure reachability*, $u \mapsto_{\epsilon, \mu} v$, with account u for a given ϵ and μ , if u is *core* and v is a member of $N_\epsilon(u)$:

$$u \mapsto_{\epsilon, \mu} v \Leftrightarrow K_{\epsilon, \mu}(u) \wedge v \in N_\epsilon(u). \quad (24)$$

In the SCAN algorithm, accounts that are *core* are set as pivots and then clusters are expanded by including accounts with direct structure reachability ([Equation 24: Clustering the transaction graph](#)). This requires computing the structural similarity with each neighbor and the neighbors' neighbors.

The improved version of SCAN only looks at the pivot accounts and accounts that are two-hops away from the pivot accounts. Accounts two-hops away from account u , $H(u)$, are defined as follows:

$$H(u) = \{v \in V | (u, v) \notin E \wedge (v, w) \in E\}, \quad (25)$$

where account w is an account, such that $w \in N_\epsilon(u) \setminus \{u\}$. For each *core* account that is two-hops away from the pivot, a new cluster is generated and pivoted around it. All of the *core* account's epsilon neighbors (N_ϵ) are added to the new cluster. When computing the accounts that are two-hops away, accounts with *direct structure reachability* from the pivoted node are removed from the calculation. When expanding the two-hops away accounts, the accounts are processed, such that:

$$H(u_n) = \left\{ v \in V | (u, v) \notin E \wedge (v, w) \in E \wedge v \notin \bigcup_{i=0}^{n-1} N_\epsilon(u_i) \cup H(u_i) \right\}. \quad (26)$$

After all accounts in the graph have been processed, all nodes are analyzed. If an account belongs to multiple clusters, then those clusters are merged. Afterwards, any account that is not in a cluster is marked as a hub if it connects two or more clusters or as an outlier if it does not.

The use of the two-hop away nodes to expand the clusters reduces the computation cost of clustering because the calculation of structural similarity σ is the slowest part of the algorithm.

The computed clusters are also used to determine the levels in the NCDawareRank inter-level proximity matrix, as these clusters are representative of the nearly completely decomposable nature of the transaction graph.

7.5 Calculating Importance Scores

The importance score, ψ , is calculated as follows:

$$\psi = (\text{normalize}_1(\max(0, \nu + \sigma w_o)) + \hat{\pi} w_i) \chi, \quad (27)$$

where:

normalize₁(v) is: $\frac{v}{\|v\|}$

ν is the vested amount of XEM

σ is the weighted, net outlinking XEM

$\hat{\pi}$ is the NCDawareRank [10] score

χ is a weighting vector that considers the structural topology of the graph

w_o, w_i are suitable constants

χ considers the topology of the graph and assigns a higher weight to nodes that are members of clusters, rather than outliers or hubs. Outliers and hubs are weighted at 0.9 of their score, whereas nodes that are in clusters are weighted at 1.0. In NEM, w_o is 1.25 and w_i is 0.1337.

Taken together, the information about vested balance, sent XEM, and transaction graph-topology form the basis for heuristic evaluation of the importance of accounts in the NEM economy. Furthermore, since importance cannot be arbitrarily manipulated or gamed (see subsection 7.6: Resistance to Manipulation), importance scores are useful for purposes other than just block chain consensus. For example, they can be interpreted as a form of reputation. Because all importance scores sum to unity, they represent a finite quantity that can be used for purposes such as voting or preventing spam. This allows even anonymous actors to interact with each other because it is not possible for people to gain control by creating many pseudonymous identities.

7.6 Resistance to Manipulation

The use of NCDawareRank, vested balance, decayed and weighted outlinks, and summation to unity in the calculation of importance scores makes the scores resistant to arbitrary manipulation.

7.6.1 Sybil Attack

In peer-to-peer systems, faulty or malicious entities can often present themselves as multiple identities in order to gain control of a system [3]. This is called a Sybil attack.

In NEM, there is a financial reward for harvesting blocks because harvesters receive fees (see section subsection 5.3: Block creation), and accounts with higher importance scores have higher probabilities of harvesting a block. As a result, attackers are expected to be highly motivated to attempt Sybil attacks. Sybil-style attacks were considered when designing the PoI algorithm. The usage of NCDawareRank, vested balances, and net decaying outlink weights makes the importance score calculation resistant to Sybil attacks.

With respect to the importance score calculation, some of the potential vectors for Sybil attackers include:

- account splitting and transacting among split accounts to boost the NCDawareRank score
- account splitting and transacting with random accounts

-
- sending XEM between accounts in a loop to boost the outlink score (see [subsubsection 7.6.2: Loop Attack](#))

The following mitigations are in place to counter Sybil attacks.

- **Use NCDawareRank instead of PageRank as a graph-theoretic importance metric**

The inter-level proximity matrix \mathbf{M} in the NCDawareRank algorithm makes the algorithm more resistant to link spamming than PageRank [10]. For web pages, PageRank is commonly spammed by creating many sites that all link to a main site, magnifying the PageRank of the main site [4]. The analog to this in NEM would be for an account to send portions of its balance to other accounts and then send all the XEM back to the main account.

- **Vest the balance of accounts over a temporal schedule that is qualitatively “slow” for humans**

Requiring several weeks to fully vest inflows into an account makes it impossible to acquire large amounts of XEM and then immediately attack the network.

- **Use net-outlinking XEM for outlink score calculation**

The net-outlinking XEM is used in the importance score calculation. There is no advantage in sending XEM around to many accounts because the inlinking XEM sent to the account will cancel out the outlinking XEM.

- **Decay the weight of outlinking value transfers**

Outlinking value transfers are decayed over time. Sending XEM to another account will only give a temporary boost in the importance score.

- **Normalize importance scores to sum to unity**

The actions of others affect your importance score because all scores sum to unity.

- **Require a minimum balance of 10,000 vested XEM required for harvesting**

Requiring at least 10,000 vested XEM to harvest creates bounds on the number of accounts that could theoretically be involved in Sybil attacks.

- **Choose relatively small values for w_o and w_i**

This ensures that the largest component of importance is the amount of vested XEM, which cannot be exploited.

Taken together, these countermeasures make Proof-of-Importance resistant to Sybil-style attacks. To test this, we performed simulations with between 1 and 10,000 colluding Sybil accounts. Figure 10 shows the results of the simulation for 3 conditions:

-
- colluding accounts sending XEM in a loop to each other
 - colluding accounts all sending XEM to a common, colluding account
 - colluding accounts sending XEM to accounts randomly

As figure 10 shows, after adding a small number of accounts, a Sybil attacker does not gain much importance by adding even thousands of additional accounts. Although the Sybil attacker gains more importance than the honest actor, the gain is limited and bounded by the PoI calculation. Some actors can boost their importance by splitting their accounts or sending transactions to other accounts to mimic economic activity. Because PoI scores sum to unity, other rational actors should take the same actions. This would cause the expected benefit to disappear.

Compared to PoW mining, the advantage that can be gained by gaming PoI is minimal. In PoW, miners who buy specialized mining hardware have a large advantage over those who use only commonly available video cards. As of this writing, an equal amount of money can buy either (1) two video cards with combined 800 Mhash/s of mining power or (2) a specialized ASIC miner with 800,000 Mhash/s. The difference between the mining power of (1) and (2) is approximately 1000x. In PoI, due to the constraints of the algorithm, an actor perfectly gaming the system will get an advantage well less than an order of magnitude. In other words, PoI gaming will not result in a significant qualitative advantage over actors with the same balance who do nothing.

For the case where controlled accounts all send to a common master account, figure 10b shows that the importance can actually decrease when more nodes are involved in an attack. For figure 10b, the importances decreased after a threshold of accounts were combined in the attack because the graph topology changed such that the attacking nodes were no longer in the same cluster. This caused the structural topology weight, χ , to be reduced from 1.0 to 0.9.

7.6.2 Loop Attack

In the loop attack, accounts controlled by the same entity send XEM around via transfer transactions in a loop to boost their importance score. While the outlinking XEM from an account is a large portion of the importance score calculation, the outlinking XEM is the *net* outlinking XEM, such that inlinking XEM to an account is subtracting from the outlinking XEM. Thus sending 1,000 XEM around in a loop millions of times will not give you a higher net outlinking XEM score than sending out just once.

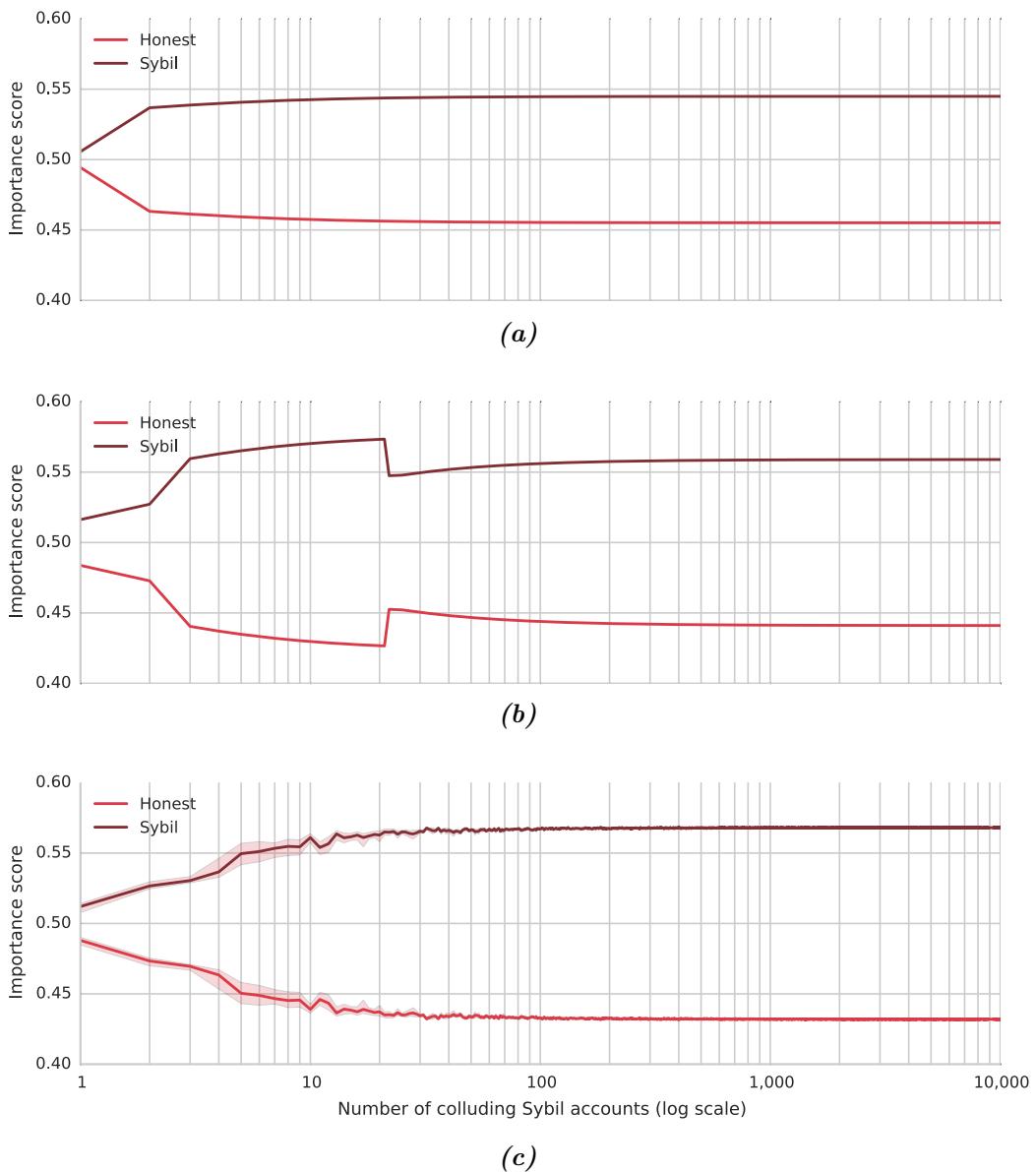


Figure 10: Importances graphed for two actors with 800 million XEM with combined 40 million XEM sent via outlinking value transfers. The honest actor keeps his XEM in one account, whereas the Sybil actor controls multiple accounts and: (a) sends XEM around in a loop between controlled accounts, (b) vests the XEM in controlled accounts and then sends them all to a single master account, and (c) sends XEM from each controlled account to a random account. Importance scores are plotted on the y-axis and the number of colluding accounts for the Sybil actor is plotted in log scale on the x-axis. For (c), shading denotes 95% CI from 10 trials and solid lines denote the mean.

7.7 Nothing-at-Stake Problem

Theoretically, algorithms for probabilistic Byzantine consensus that do not require the expenditure of external resources are subject to the *nothing-at-stake* problem⁶. The nothing-at-stake problem theoretically exists when the opportunity cost of creating a block is negligible. In other words, the cost of creating a block is so low that it is easy to create a block for all known forks of a block chain (including those forks that were self-created in secret). In contrast, the nothing-at-stake problem does not exist when Proof of Work is used for consensus because creating a block using such a consensus mechanism is resource-intensive.

In NEM, the cost of creating a block is negligible. To mitigate the nothing-at-stake problem, NEM caps the change in block difficulty and also limits the length of the chain that can be unwound during fork resolution.

As described in [subsection 5.1: Block difficulty](#), the block difficulty change rate is capped at a maximum of 5%. If an attacker has considerably less than 50% of the harvesting power of the network, this will cause the block creation time of the attacker's secret chain to be much higher than that of the main chain in the beginning. This large time difference will make it unlikely for the attacker's chain to be better and accepted. In addition, the block chain unwind limit is 360 blocks (see [subsection 5.4: Block chain synchronization](#)), which prevents long-range forks from harvesters working on multiple chains.

7.8 Comparing importance to stake

As [subsection 7.5: Calculating Importance Scores](#) shows, the vested balance of an account is a large component of the importance score. Taking the vested balance as the "stake" used in currencies implementing the Proof-of-Stake (PoS) algorithm, it could be argued that PoS and PoI are similar. In order to explore the differences and similarities between PoS and PoI, the NEM and Bitcoin transaction graphs were analyzed. Bitcoin was chosen due to the relatively large size of its user base and transaction graph.

As of April 29, 2015, the NEM transaction graph had 1,456 harvesting-eligible accounts (see [subsection 7.1: Eligibility for Entering the Importance Calculation](#)). In October 2014, approximately one month of data from the Bitcoin network was downloaded and analyzed. 54,683 accounts were harvesting-eligible when BTC amounts were normalized to NEM. Amount normalization was done by multiplying the BTC amounts by the market cap ratio. Thus, 0.0177 BTC was the minimum amount of vested BTC for an account to be considered harvesting-eligible. Block heights were normalized to NEM by multiplying by

⁶See Vitalik Buterin's blog post on the subject for a good overview: <https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/>.

10.

Figure 11(a) shows importance scores and vested balances plotted on a log scale for each of the 1,456 harvesting-eligible accounts (sorted by vested balance, ascending) in the NEM transaction graph, and (b) for the 54,683 harvesting-eligible accounts in the Bitcoin transaction graph. As can be seen, while the vested balances are monotonically increasing, the importance scores are non-monotonic, demonstrating that accounts with lower vested balances are able to gain higher salience in PoI than in PoS in both transaction graphs.

Figure 12 shows the plotted NEM transaction network graphs for both normalized importance scores and vested balances in order to give an overview of the qualitative differences between PoI and PoS. Normalization was done to make the differences between scores with low and high values more visible. It was accomplished by scaling the importance scores and vested balances to sum to unity, taking the absolute value of the logarithm of that, mapping it to an exponential function, and then rescaling to sum to unity. As can be seen in the graph, vested balances give larger weight to fewer nodes, whereas importance scores have less noticeably larger nodes.

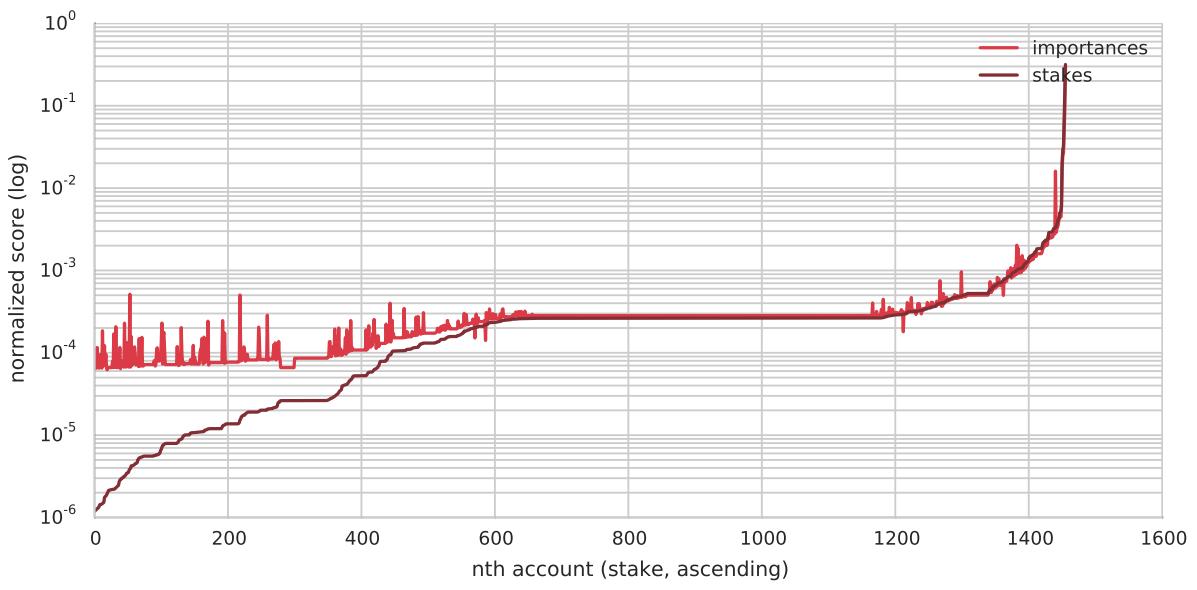
To quantify the differences in account saliences, all the accounts for NEM and Bitcoin were ranked (densely; accounts with the same score/balance were given the same rank) based on vested balances and importance scores and looked at the differences in ranks between the two metrics. [Table 1: Differences between NEM account ranks for importance scores vs. vested balances](#). shows the results for accounts in the NEM transaction graph and [Table 2: Differences between Bitcoin account ranks for importance scores vs. vested balances](#). shows the results of ranking Bitcoin accounts. Overall, NEM accounts ranked by importance scores were ranked approximately 338 ranks lower than when ranked by vested balances. The richer half of accounts decreased by an average of 443 ranks while the poorer half decreased by an average of 232 ranks. Bitcoin accounts were ranked an average of 67.5 ranks lower when ranked with importance scores than when ranked by vested balances. The poorer half of accounts gained an average of 2,814 ranks and the richer half lost an average of 2,950 ranks. This suggests that PoI gives less salience overall to richer accounts than PoS.

Table 1: Differences between NEM account ranks for importance scores vs. vested balances.

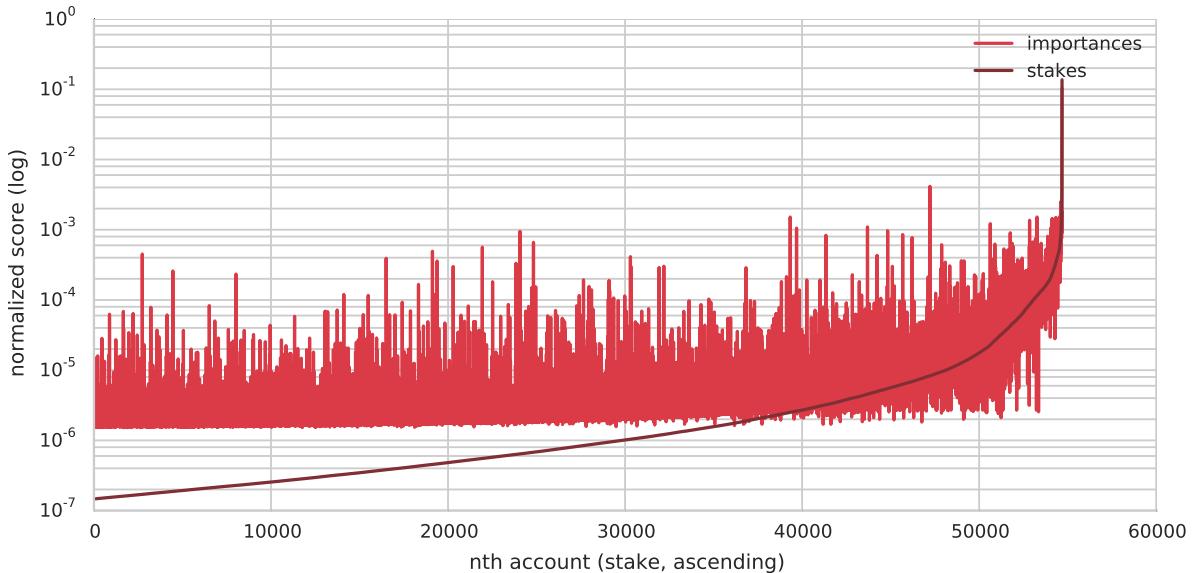
average rank increase for importance vs stakes (poor half):	-232.4
average rank increase for importance vs stakes (rich half):	-443.8
average rank increase for importance vs stakes:	-338.1

Table 2: Differences between Bitcoin account ranks for importance scores vs. vested balances.

average rank increase for importance vs stakes (poor half):	2814.6
average rank increase for importance vs stakes (rich half):	-2949.6
average rank increase for importance vs stakes:	-67.5

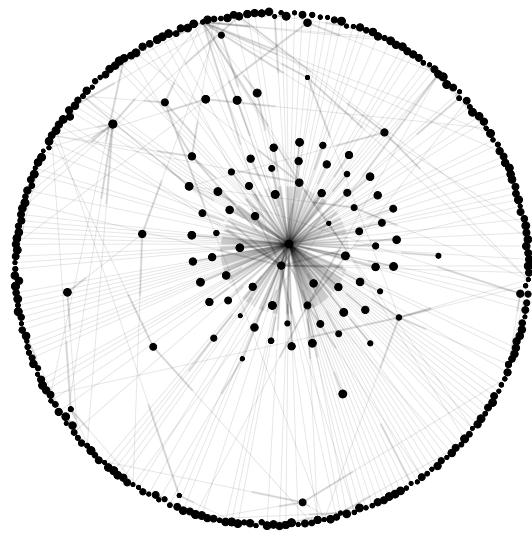


(a)

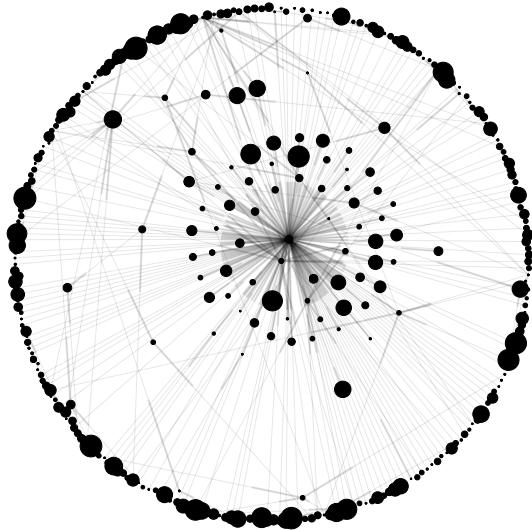


(b)

Figure 11: Importance scores and vested balances for accounts in the harvesting-eligible subset of the (a) NEM and (b) Bitcoin transaction graphs are plotted. Importance scores and vested balances were normalized to sum to unity (1.0), with accounts sorted in ascending order. Graphs are plotted on the y-axis with a log scale, allowing a clear comparison between stake and importance scores, and accounts are on the x-axis.



(a)



(b)

Figure 12: NEM transaction graphs where node sizes denote normalized (a) importance scores and (b) vested balances. Normalization is described in subsection 7.8: Comparing importance to stake.

8 Time synchronization

You spend too much time on ephemeras. The majority of modern books are merely wavering reflections of the present. They disappear very quickly. You should read more old books. The classics. Goethe.

- Franz Kafka



IKE most other crypto currencies, NEM relies on time stamps for transactions and blocks. Ideally, all nodes in the network should be synchronized with respect to time. Even though most modern operating systems have time synchronization integrated, nodes can still have local clocks that deviate from real time by more than a minute. This causes those nodes to reject valid transactions or blocks, which makes it impossible for them to synchronize with the network.

It is therefore needed to have a synchronization mechanism to ensure all nodes agree on time. There are basically two ways to do this:

1. Use an existing protocol, such as NTP
2. Use a custom protocol

The advantage of using an existing protocol like NTP is that it is easy to implement and the network time will always be near real time. This has the disadvantage that the network relies on servers outside the network.

Using a custom protocol that only relies on the P2P network itself solves this problem, but there is a trade off. It is impossible to guarantee that the network time is always near real time. For an overview over different custom protocols see for example [12].

NEM uses a custom protocol in order to be completely independent from any outside entity.

8.1 Gathering samples

Each node in the network manages an integer number called *offset* which is set to 0 at start. The local system time in milliseconds incremented by the offset (which can be negative) is the *network time* (again in milliseconds) of the node.

After the start up of a node is completed, the node (hereafter called *local node*) selects up to 20 partner nodes for performing a time synchronization round. Only nodes that expose a minimum importance are considered as partners.

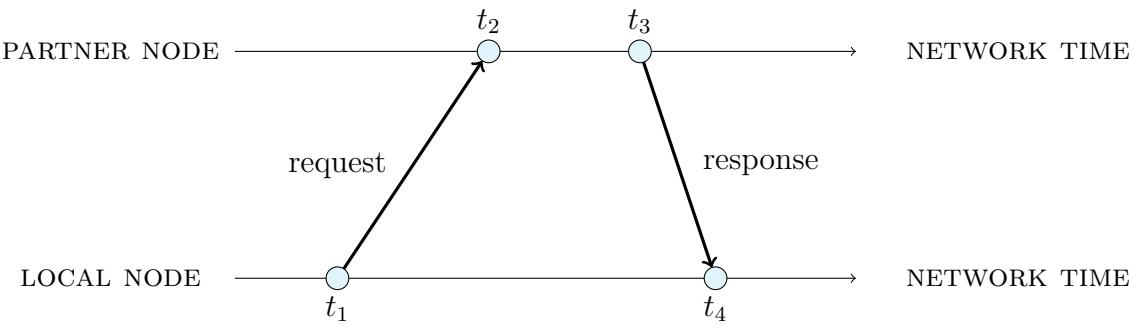


Figure 13: Communication between local and partner node.

For all selected partners the local node sends out a request asking the partner for its current network time. The local node remembers the network time stamps when the request was sent and when the response was received. Each partner node responds with a sample that contains the time stamp of the arrival of the request and the time stamp of the response. The partner node uses its own network time to create the time stamps. Figure 13 illustrates the communication between the nodes.

Using the time stamps, the local node can calculate the round trip time

$$rtt = (t_4 - t_1) - (t_3 - t_2)$$

and then estimate the offset o between the network time used by the two nodes as

$$o = t_2 - t_1 - \frac{rtt}{2}$$

This is repeated for every time synchronization partner until the local node has a list of offset estimations.

8.2 Applying filters to remove bad data

There could be bad samples due to various reasons:

- A malicious node can supply incorrect time stamps.
- An honest node can have a clock far from real time without knowing it and without having synchronized yet.
- The round trip time can be highly asymmetric due to internet problems or one of the nodes being very busy. This is known as channel asymmetry and cannot be avoided.

Filters are applied that try to remove the bad samples. The filtering is done in 3 steps:

1. If the response from a partner is not received within an expected time frame (i.e. if $t_4 - t_1 > 1000ms$) the sample is discarded.
2. If the calculated offset is not within certain bounds, the sample is discarded. The allowable bounds decrease as a node's uptime increases. When a node first joins the network, it tolerates a high offset in order to adjust to the already existing consensus of network time within the network. As time passes, the node gets less tolerant with respect to reported offsets. This ensures that malicious nodes reporting huge offsets are ignored after some time.
3. The remaining samples are ordered by their offset and then alpha trimmed on both ends. In other words, on both sides a certain portion of the samples is discarded.

8.3 Calculation of the effective offset

The reported offset is weighted with the importance of the boot account of the node reporting the offset. This is done to prevent Sybil attacks.

An attacker that tries to influence the calculated offset by running many nodes with low importances reporting offsets close to the tolerated bound will therefore not have a bigger influence than a single node having the same cumulative importance reporting the same offset. The influence of the attacker will be equal to the influence of the single node on a macro level.

Also, the numbers of samples that are available and the cumulative importance of all partner nodes should be incorporated. Each offset is therefore multiplied with a scaling factor.

Let I_j be the importance of the node reporting the j -th offset o_j and $viewSize$ be the number of samples divided by the number of nodes that were eligible for the last PoI calculation.

Then the scaling factor used is

$$scale = \min \left(\frac{1}{\sum_j I_j}, \frac{1}{viewSize} \right)$$

This gives the formula for the effective offset o

$$o = scale \sum_j I_j o_j$$

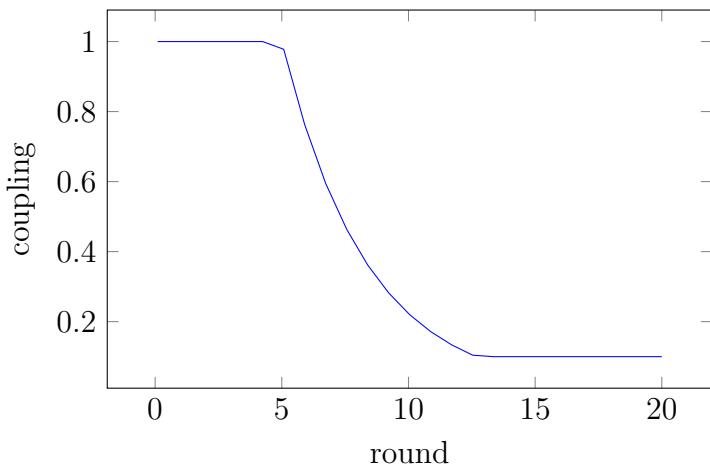


Figure 14: Coupling factor

Note that the influence of an account with large importance is artificially limited because the *viewSize* caps the scale. Such an account can raise its influence on a macro level by splitting its NEM into accounts that are not capped. But, doing so will likely decrease its influence on individual partners because the probability that all of its split accounts are chosen as time-sync partners for any single node is low.

8.4 Coupling and threshold

New nodes that just joined the network need to quickly adjust their offset to the already established network time. In contrast, old nodes should behave a lot more rigid in order to not get influenced by malicious nodes or newcomers too much.

In order to enable this, nodes only adjust a portion of the reported effective offset. Nodes multiply the effective offset with a coupling factor to build the final offset.

Each node keeps track of the number of time synchronization rounds it has performed. This is called the node age.

The formula for this coupling factor c is:

$$c = \max(e^{-0.3a}, 0.1) \text{ where } a = \max(\text{nodeAge} - 5, 0)$$

This ensures that the coupling factor will be 1 for 5 rounds and then decay exponentially to 0.1.

Finally, a node only adds any calculated final offset to its internal offset if the absolute

value is above a given threshold (currently set to 75ms). This is effective in preventing slow drifts of the network time due to the communication between nodes having channel asymmetry.

9 Network

“

For what is it? Nothing but a little blood and bones; a piece of network, ” wrought out of nerves, veins, and arteries twisted together.

- *Marcus Aurelius*



HE NEM network is comprised of NIS nodes. Each node is associated with a single primary account, which is used to authenticate responses by that node. This prevents an attacker from impersonating a node without having its private key even if he can spoof the node's IP address.

Each NIS node has the following configuration settings:

- `nis.nodeLimit` - the number of other nodes to which the local node should broadcast information
- `nis.timeSyncNodeLimit` - the number of other nodes the local node uses to synchronize its clock [section 8: Time synchronization](#)

Typically, `nis.timeSyncNodeLimit` should be larger than `nis.nodeLimit` in order to allow better time smoothing. This isn't too expensive because the data transferred as part of time synchronization is relatively small.

9.1 Node Protocol

NIS nodes communicate amongst themselves using a proprietary binary format by default. This format minimizes network bandwidth by compacting requests and processing by reducing the cost of serialization and deserialization. In fact, all NIS APIs support requests encoded in either the NEM proprietary binary format or JSON.

In order to prevent impersonation-based attacks, NIS nodes engage in a two part handshake when communicating:

1. The local node sends the request data and a random 64-byte payload to the remote node.
2. The remote node signs the random payload and sends the signature along with the requested response data back to the local node.
3. The local node checks the signature and only processes the response data if it can verify that the remote node signed the random payload.

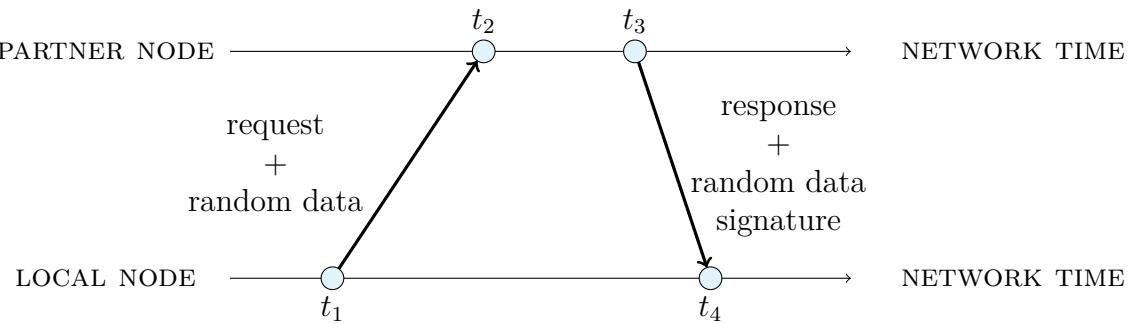


Figure 15: Communication between local and partner node.

9.2 Node Startup

When a NIS node is launched, the node processes the block chain and caches some data in memory to improve online performance. Once the processing is finished, the node is still not connected to the network because it is not yet booted.

A non-booted node is not associated with an account. This prevents it from being able to sign responses and prevents other nodes from being able to verify its identity.

In order to boot a node, the private key of a NEM account must be supplied to the node. This account is the primary account associated with the node. A delegated account can be used to boot a node in order to better protect the private key of the real account (see [subsection 4.2: Importance transfer transactions](#)).

9.3 Node Discovery

Once a node is booted, it connects to the NEM network and starts sharing information with other nodes. Initially, the node is only aware of the well-known nodes. These nodes are the same as the pre-trusted nodes described in [subsection 6.2: Local trust value](#).

Over time, the node becomes aware of more nodes in the network. There are typically two ways this happens: via an announcement or a refresh.

9.3.1 Announcement

Periodically, a node announces itself to its current partner nodes and includes its local experience information in the request. If the node is unknown to the partner, the partner marks it as active and updates the node's experiences. If the node is known to the partner, the partner only updates the node's experiences but does not change its status.

9.3.2 Refresh

Periodically, a node asks its current partners to provide updated information about themselves and the state of the network.

First, the node requests update information about the remote node. If successful, the local node will update the remote's endpoint (necessary to support dynamic IPs) and metadata. This step fails if any of the following occur:

- The remote node returns a valid request from a different node
- The remote node is not compatible with the local node (e.g. the remote node is testnet but the local node is mainnet)

On success, the remote node is asked to provide a list of all of its current partner nodes. In order to prevent an evil node from providing incorrect information about good nodes and causing them to be blacklisted, the local node attempts to contact each reported partner node directly. It is important to note that a node's metadata will only be updated by metadata provided by that node itself.

As an optimization, blacklisted nodes are not refreshed. A node can be temporarily blacklisted if it returns a fatal error or the local node cannot connect to it. Periodically, blacklisted nodes will be removed from the blacklist and will be allowed to participate in the refresh operation again. If the original blacklisting was due to a non-transient error, the node will likely be blacklisted again.

9.4 Node Selection

Over time, as a result of the node discovery process, the local node will become aware of more nodes in the network. Eventually, the number of known nodes (including both well-known and other nodes) will be much greater than the number of partner nodes.

Periodically, a node recalculates its partner nodes. When this recalculation occurs, all known nodes are eligible to become a partner node, including nodes that have been detected as busy. With default settings, this recalculation will occur more frequently than the recalculation of trust values. By default, trust values are recalculated every 15 minutes.

The known nodes are weighted by their trust values and the partner nodes are randomly selected from among them. Nodes with larger trust values (which are more likely to be good nodes) are more likely to be chosen as partners. Nodes that have been minimally interacted with typically have trust values at or close to 0. In order to give these nodes

a chance to prove themselves and build trust, they are effectively given a small boost in trust so that they have the opportunity to be selected and participate in the network. 30% of trust is evenly distributed among nodes with less than 10 interactions.

In order to ensure that the network is connected, one adjustment to the random process is made. If the local node is a well-known node, it is additionally connected with all online well-known nodes. If the local node is not a well-known node, it is additionally connected with one random, online, well-known node. This ensures that all nodes are actively partnering with at least one well-known node.

References

- [1] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644. ACM, 2011.
- [2] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, pages 124–142, 2011.
- [3] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.
- [4] Nadav Eiron, Kevin S McCurley, and John A Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318. ACM, 2004.
- [5] Xinxin Fany, Ling Liu, Mingchu Li, and Zhiyuan Su. Eigentrust++: Attack resilient trust management. 2012.
- [6] Tayfun Gürel, Luc De Raedt, and Stefan Rotter. Ranking neurons for mining structure-activity relations in biological neural networks: Neuronrank. *Neurocomputing*, 70(10):1897–1901, 2007.
- [7] Bin Jiang. Ranking spaces for predicting human movement in an urban environment. *International Journal of Geographical Information Science*, 23(7):823–837, 2009.
- [8] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proceedings of the 12th international conference on World Wide Web*, pages 640 – 651, 2003.
- [9] A.N. Langville and C.D. Meyer. *Google’s PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, Princeton, USA, 2006.
- [10] Athanasios N Nikolopoulos and John D Garofalakis. Ncdawarerank: a novel ranking method that exploits the decomposable structure of the web. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 143–152. ACM, 2013.
- [11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab, Stanford, USA, November 1999.

-
- [12] Sirio Scipioni. *Algorithms and Services for Peer-to-Peer Internal Clock Synchronization*. PhD thesis, Universit'a degli Studi di Roma „La Sapienza”, 2009.
 - [13] H Shiokawa, Y Fujiwara, and M Onizuka. Fast structural similarity graph clustering. In *The 6th Forum on Data Engineering and Information Management (DEIM2014)*, 2014.
 - [14] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.



VECHAIN DEVELOPMENT PLAN

THIS IS NOT A WHITE PAPER !

No, it is NOT!

Preface

The VeChain team and the VeChain Blockchain platform has been running for more than two years.

Fortunately, while running down the path of Blockchain which everyone has big hopes on right now, we met many people sharing the same goal and lots of enterprise customers that dare to explore this new area. We also met many passionate business partners and co-workers with strong believes. Moreover, we accumulated lots of experience of business use cases for different industries and we kept adjusting and making corrections during the process so that we can continue on searching the right way to use this “secret technology” that may change the world.

Our original vision has never been changed. The dream is still same as before,

Building a trust-free and distributed business ecosystem to enable transparent information flow, efficient collaboration, and high speed value transferring.

Table of Content

1.	Concept Background	5
1.1	Call for ICO	5
1.2	The Understanding of Blockchain Technology	6
1.2.1	Synergy and Value Transfer	6
1.2.2	Data and Information Symmetry	7
1.3	VeChain's Vision.....	8
1.3.1	Distributed Business Ecosystem.....	9
1.3.2	The "Blood" in the Distributed Ecosystem – VeChain Token(VET).....	10
1.4	VeChain's Attitude on Blockchain Technology	12
2.	Methodology and Technical Support.....	13
2.1	Methodology	13
2.2	Technique Support.....	14
2.3	Technical Structure	15
2.4	Achieve the Technical Details.....	18
2.4.1	VeChain ID Creation and Hashing.....	18
2.4.2	Storage of VID on Blockchain.....	19
2.4.3	Digital Ownership on Blockchain	20
2.4.4	Data Hashed Storage (proof of data)	21
2.4.5	API Gateway.....	22
2.4.6	Service Discovery (SDP).....	23
2.4.7	Micro-Service	24
2.4.8	Hashed Storage Service (HSS)	24
2.5	Blockchain and IoT	26
2.5.1	The Issue of IoT	26
2.5.2	Blockchain and IoT	26
2.5.3	VeChain and IoT	28
2.6	Technical Testing	29
2.7	Technology Development's Path and Plan.....	31
3.	The Industrial Application and Expansion.....	34
3.1	Fashion and Luxury Industry	35
3.2	Food Safety	36
3.3	Car Industry	39
3.4	Supply Chain Industry	40
3.5	The Agricultural Industry.....	41
3.6	Blockchain Government Affairs	42
3.7	This is just the Beginning	44
4.	Governance Structure and Management Philosophy	45

4.1 The Establishment of VeChain Foundation.....	46
4.2. Governance Principle	46
4.3 VeChain Governance Model	48
4.3.1 Strategic Steering Committee.....	49
4.3.2 General Secretary	50
4.3.3 Technical Audit Committee.....	50
4.3.4 Remuneration and Nomination Committee	50
4.3.5 Public Relation Committee.....	51
4.3.6 Supervisory Committee	51
4.3.7 Other Functional Department	51
4.4 VeChain Human Resource Management	51
4.5 Risk Assessment and Decision Making Mechanism of VeChain Foundation	52
4.6 VeChain Foundation Economy.....	52
4.6.1 Funding Sources	53
4.6.2 Fund Budgeting	54
4.6.3 Fund Use Restriction	56
4.6.4 Financial Planning and Implementation Reports.....	57
4.6.5 Digital Asset Management	57
4.7 Legal Compliance Matters and Other Matters	57
5. Introduction of the Team and Team Member.....	58

1. Concept Background

1.1 Call for ICO

The Blockchain technology is experiencing rapid development especially in recent one or two years. The change is so fast for us entrepreneur. Despite the direction is technology development and expansion, or application research, we face changes all the time. Every time we site together, discuss and conclude "the only thing that does not change is change itself", just like a poet. From the second half of 2015, with an article from The Economist "Blockchain: The Trust Machine", the Blockchain technology started to walk out of the geek community and rapidly gained worldwide attention in various industries.

The term "Blockchain" is no longer an obscure technical term for many people. Lots of the new ideas and projects are coming out, which includes many imaginative models and new directions for Blockchain. At the end of last year, the Blockchain technology was even written directly into the national "13th Five-Year plan", which encouraged us many peers. In addition, it attracts so many aspiring young people to join the Blockchain industry.

Needless to say, Blockchain has been recognized by the world as a new generation of powerful technology. Blockchain is considered as being able to change the world again just like what the Internet technology did. In addition, based on the pattern of human technology development, the development process of the Blockchain technology will suddenly get speeds up by a huge margin without doubt. We believe that it will have substantial breakthroughs and extensive expansion in the upcoming few years for the Blockchain industry.

However, the reality is tougher than expected. The application direction of Blockchain is either for financial industry or non-financial industry. For financial industry, it is so obvious with its high standard of compliance, so it is very hard to make a break through. For non-financial industry, it has a variety of collaborative cooperation modes but all these participants lack of drive to move further. That is why even though there is some new concepts derived from Blockchain, only few practical Blockchain business application has been established. Even when some project get partially established can make the teams feel excited.

Although everything is hard at the beginning, there is always someone who is very careful and willing to be the first one to get into the field. In order to reduce the possibility of failure as much as possible, we want to share VeChain with the investors, enterprise customers, cooperative partners and colleagues – a product that we started making strategic plan two years ago. It has been through several platform software updates, many practical cases and debates with arguments at

so many sleepless nights. Finally we created some matured ideas and we want to CALL for the ICO for VeChain project.

1.2 The Understanding of Blockchain Technology

1.2.1 Synergy and Value Transfer

In the world of traditional business, different varieties of collaborative and business operations as well as the whole financial industry, which is at the top of the "food chain", shows trust is the biggest cost in the field. Though Blockchain carries a "trust aura". The Blockchain technology is widely accepted around the world since the famous article "Blockchain: the trust machine" published by the Economist.

The essence of a Blockchain is an Internet protocol and a collection of technologies about Trust. We can define the meaning of Blockchain from three dimensions - data, system and application:

- *From the data point of view: Blockchain is a distributed database system that is continuously updated in chronological order. The data can only be added but not tampered with.*
- *From the system point of view: Blockchain is a distributed deployment and real-time synchronization system, allows participants from different parties to create and maintain the data through mechanism for consensus. It makes each active node on the Blockchain has exactly the same data.*
- *From the application point of view: Blockchain is a standard global platform allows multiple participants to connect at the same time and records all digital objects, users, and their relative operations on this platform.*

With the development of information technology and Internet, the application of various systems makes the collaboration more convenient and efficient. Because of the existence of trust issue, the majority of such efficient collaboration exists mainly within an enterprise or a certain organization. However, people are using the methods and tools from 40 years ago when it comes to the collaboration between different enterprises. The majority of collaboration is still completed by e-mail. System interfacing is actually not as simple as imagined. Since it involves data security, trade secrets, cooperation, trust and other problems. The connection is not just a technical issue. In addition, due to the same problem, the financial service that match and support all kinds of business needs improvement in both efficiency and cost.

For example, a classic business collaborative mode includes supply chain (as the graphic shows below), brand, manufacturer, distributed retailer, consumer and regulator. All the parties share the same goal: to achieve the same value of

improving the life quality of the consumer. However, even if the different enterprises worked together for the same goal, due to the lack of sufficient trust guarantee, the cooperation is still on a peer-to-peer manner and with traditional communication tools, and the data exchange will be very inefficient and expensive. In such a traditional product life cycle, even if the logistics could be relatively smooth and efficient, the flow of information is often fragmented and the transfer of funds is also relatively slow. For the participants on the whole supply chain, the utilization rate of funds has always been quite a headache.

The Blockchain technology can help us to establish a new trust-free sharing business collaboration model (as the graphic shows below). Various parties can ensure the security of data in a more convenient and smoother manner. With the support of a more timely and accurate information flow, the value transmission in the ecological environment can be developed and executed during the business activities. This way each enterprise can increase the utilization rate of funds, and greatly improve the speed of value transmission in order to support more business development.

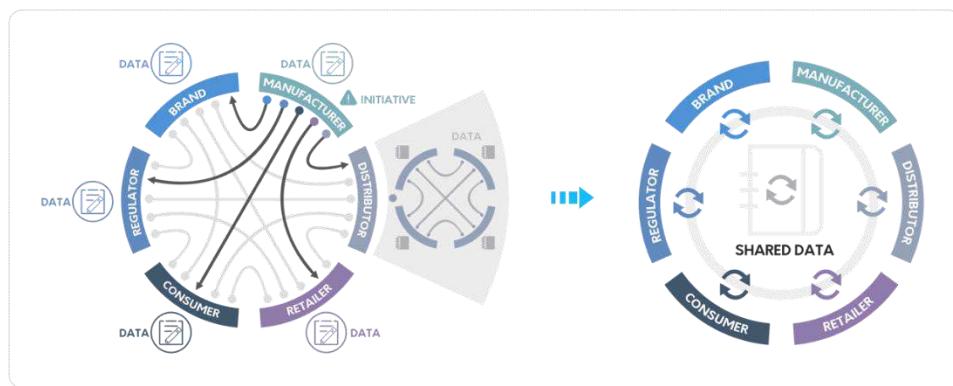


Figure 1.2.1 Distributed business collaboration from traditional business collaboration to Blockchain

1.2.2 Data and Information Symmetry

Most enterprises have three types of data:

- 1) Public data, such as enterprise data that is publicly available on the official website.
- 2) Private data, such as enterprise product developing documents, financial reports for non-listed companies.
- 3) Permitted shared data, usually exists between different cooperation partners. Such as the record data from the identification, logistics, payment information and after-sale service for different level enterprises.

The first and second kind of data are generally well understood. The interesting part is the third type of data, which is usually converted into private data by the participants.

For example, after a car is sold, maintenance data exists in the 4S store or the maintenance business. When the owners need to buy insurance, the insurance company spend huge cost to buy those maintenance data as a service provider. Or, based on the market demand, this could become as a new data service. This service can collect data, centralize maintenance and management, and get paid by providing data to participants who are interested in those data. For users, the risk comes from the centralized/ isolated data, where stay in a variety of online automotive service platform. This kind of service breaks the asymmetric information (due to the difference of region and time) by using information technology, and constructs a new centralized asymmetric information, and then produce profit base on this.

We BELIEVE that the Blockchain technology can continuously break those asymmetric information and allow data to return to the real owner eventually. For instance, in the car scenario described above, the data generated by the owner when he used the car should naturally be owned by the owner. That means the data generated from car maintenance should belong to car owner without doubt. While enjoying other service like insurance later, insurance company can reduce data audit costs through user authorized credible data. Thus car owners could enjoy premiums service with lower cost.

The Blockchain technology allows data owners really own the data, and let the data owner have the authority to choose whether to share their data, which completely breaks the traditional asymmetric of the centralized information. In this way, the value goes back to where they were. The data owned by one side needs to be shared and maintained by all parties. Thus new values are generated and is properly allocated in the activities of multi-party participation.

1.3 VeChain's Vision

What does the VeChain want to do? **The vision of VeChain is to build a trust-free and distributed business ecosystem based on the Blockchain technology self-circulated and expanding.**

- In this ecosystem, the information is relatively **transparent** and symmetrical. A large portion of the source of the profit comes from the realization of true value, and only a small portion of it comes from asymmetric information (absolute symmetry does not exist).
- In this ecosystem, each business party can reduce the **potential trust issue** between different parties. This makes business cooperation simpler, more efficient, low cost, and the business can concentrate resources on more advanced technology, better product and service to create more value.
- In this ecosystem, each person and each enterprise can find their own

- place. Based on their contribution and value, they can obtain relatively fair reward.
- In this ecosystem, the technology of Blockchain should have room for all aspects of business, including commercial activities and economic activities that should be supported.
 - In this ecosystem, the value is in a **closed loop that keep expanding** accompanied by the development of commercial activities with high-speed transmission. The form of value may be commodities, services, or direct fund.

1.3.1 Distributed Business Ecosystem

In the ecological environment made by VeChain, there are several main types of participants:

- 1) Enterprise organization
All kinds of enterprise organization that provide products and services to end-users to meet all the needs such as various manufacturing enterprises, brands, service providers for end-users, and so on.
- 2) Application service provider
It means an enterprise that provides various application development and services for enterprise organizations and users on VeChain Blockchain. The product or service can be a variety of decentralized applications and services to users, technical products and related services for all enterprises and institutions, functions of the government agencies, regulators and third party credit service providers.
For example, end-user oriented Internet platforms like BAT, sharing products and service providers like Uber, Didi, Airbnb;
For instance, enterprise oriented technology, product, service provider such as Oracle, IBM. Supply chain services providers of commodity enterprises, third party credit service providers such as PwC, DNV, GL and financial service providers such as banks and insurance companies.
- 3) Smart contract service provider
Organizations provide VeChain smart contract technology service to enterprises, and allow the end- enterprise or service providers to develop Blockchain applications in a faster and more convenient manner.
- 4) VeChain network node provider
Enterprises and organizations that directly participate in the Blockchain network and maintain a certain number of nodes to protect the overall network security.
Maintaining a specific function node to provide related services, such as customs, quality inspection node, audit node, wallet service, and user private key management service provider.
- 5) VeChain Foundation

VeChain is responsible for the construction of Blockchain network, technology research and development, upgrade and maintenance and other basic technical services. Meanwhile, in the initial stage it is responsible for business development, creating reference cases, encouraging and supporting more of the new smart contract service providers and existing technology enterprises to transform. Based on the demands of ecological development, it provides support to more technology companies on offering Blockchain services, such as wallet development, payment services, private key management, internal exchange, smart contract templates etc.

6) End-user

The service target of end-enterprise, end-users and service provider (investors) enjoy the bonus from the future commercial ecology development together.

These participants set up the whole VeChain distributed business ecosystem. On one hand, it can form an effective closed loop. On the other hand, it can connect and assimilate with the environment outside of the ecosystem and constantly grow itself, as figure shows below:

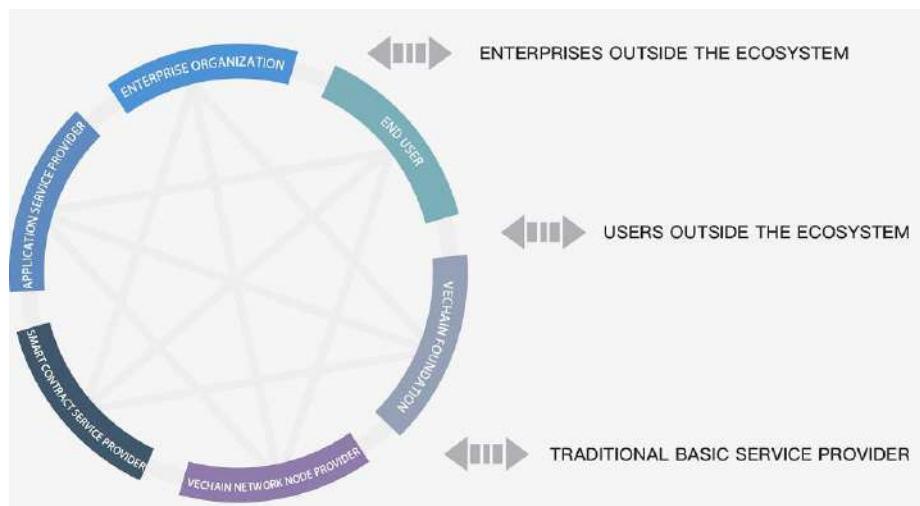


Figure 1.3.1 Distributed business ecosystem environment

1.3.2 The "Blood" in the Distributed Ecosystem – VeChain Token(VET)

If the entire distributed business ecosystem is a body, then the Blockchain infrastructure is the skeleton and various application services are the muscles and organs. Such body needs the circulation of the blood, and the blood is the VeChain tokens - VeChain Token(VET), carrying the value transfer function of the entire Blockchain network and various commercial activities running on it. VeChain Token(VET) will be openly available for sale in a variety of ways in this ICO.

As the carrier of value transmission in the whole ecosystem, VeChain Token(VET) flows through the smart contract which describes and executes the cooperation among the parties, as well as forming a special closed loop with an open interface. On one hand, the value transfer very fast in the ecological. On the other hand, it opens and communicates with outside the ecosystem as a medium, and further expanding the scope of ecology.

The main function of VeChain Token(VET) is to circulate as much as possible and to let each participant use it. So we will sell more than **70%** of the total amount of tokens to communities, businesses, and users.

Just as the graphic shows above:

- 1) This loop begins with the end-user and enterprises as investors to use ETH to obtain VeChain Token(VET) in the beginning. VeChain uses ETH to perform technical development, commercial application cooperation promotion, and Blockchain services support.
- 2) VeChain Foundation receives VeChain Token(VET) from each smart contract development and service provider to pay for the GAS needed to run the smart contract and maintain the operation of each business smart contract. The 75% to 99% of the VeChain Token(VET) income will be awarded as a node reward to the node provider, while the remaining 25% will be used for the daily operation, business promotion and technical development of the VeChain Foundation.
- 3) Smart contract service providers use VeChain Token(VET) to pay for GAS and provide smart contract service of BaaS (Blockchain as a service). According to different business rules and contributions, each participant receives VeChain Token(VET) from its client - Application Development provider provides smart contract services through collecting VeChain Token(VET).
- 4) Application provider develops and processes based on end-user's needs with the foundation of smart contract service, as well as providing products for the application of the traditional enterprise customers or end-users and receive VeChain Token(VET) as corporate income.
- 5) End-users can pay VeChain Token(VET) to obtain enterprise products and services.

Of course, such ecological development will experience different stages, and maintain an open status. A better fusion with the traditional business world will help the transformation of traditional commercial enterprises, and then expand such a distributed business ecosystem.

In this process, there must be a variety of new technology service enterprises, to provide a bridge for communication and value transfer between traditional

commercials and VeChain's distributed business environment.

VeChain will be responsible for the actual development. On the other hand, we will encourage and support outstanding teams to join us so we can have a better understand to the various sectors of all enterprises. We can apply better and more focus on developing and make the right people to do what they do best.

According to the actual experience of the past two years, we have summed up a few reference methods to carry out this ecological promotion:

- 1) The breakthrough point should select the enterprise with most "Blockchain" strategy. These enterprises value greatly on the development of Blockchain technology in the future.
- 2) Initial cases should be combined with the real enterprise issue, which can solve the actual problems, or may bring new value.
- 3) The business scenario has the multi participants, and space for deeper expansion.
- 4) Target enterprises, target cases in the industry or in different industry has considerable influence.

In this ecological development, the tactics needs to expand both horizontally and vertically:

- 1) Horizontally, make more duplicated expansion of the same types of enterprises within the same industry.
- 2) Vertically, the expansion of different enterprises and participants.

More participants will bring more extensive collaboration, more efficient value flow, give birth to a new and strong coupling business model, and then build a future distributed business ecology.

1.4 VeChain's Attitude on Blockchain Technology

The development of any new technology is bound to go through several important stages:

- ✓ The first stage, **technical barriers stage**; at this stage, being capable or not capable plays a very big difference; doing well and not well is not very obvious.
- ✓ The second stage is the **business barrier stage**. At this stage, the development of technology has been advancing by leaps and bounds, and with the trend of social resources, more and more talents have been pouring in. More technical theories and skills are being shared, and technical barriers are becoming increasingly blurred. To do or not to do already is not a problem; well done and badly becomes prominent. The key point of this stage is whether we can apply the technology skillfully and reasonably to the actual commercial products and services and

- produce greater value.
- ✓ The third stage is the **scale barrier stage**. At this stage, the snowball effect is very obvious, and the scale advantage is becoming more and more important. More business activities and social activities focus on one or more ecological environments, and the more participants, the faster they develop.
- ✓ The fourth stage is **the subdividing the vertical phase**. At this stage, the industry scale and pattern are basically formed, and new breakthroughs are made. The new breakthrough comes from the division of vertical areas with more concentrated resources advantages to produce better products, services and values.
- ✓ The fifth stage, **the birth of the new technological revolution**. More advanced technology was born in the human pursuit of higher value, and then enter the next cycle.

Blockchain has no exemption to this route. Although the Blockchain technology itself still has a long way to go, there is lots of space for improvement. Nevertheless, as things stand now, we have unwittingly entered the **early second stage** of Blockchain.

So, this *non-whitepaper* does not include mysterious algorithms and technical details. It focuses on the concept and design of the business ecosystem, and the support and further development needs of the related technologies.

We hope that our investors, partners and communities will be able to come together and build this ecosystem together.

We recognize that VeChain technology may not be the most advanced in the world. VeChain had a good technical starting point and a cohesive technical team and continuous iterations based on the needs of the application. In the process, we will be very grateful if VeChain could contribute our discovery and breakthroughs to the community, industry and Blockchain technology.

2. Methodology and Technical Support

2.1 Methodology

Based on the understanding and following the objective of business rule, VeChain wants to begin with the smallest elements in business (people, object and money). VeChain wants to digitalize all of these small elements and build a general type of connection. VeChain builds the reflection on the coordinating activities of modern business through different smart contracts. It provides related value flow tool and system in order to create a new business model based on this

coordination pattern. After that VeChain builds a new kind of distributed business ecosystem that will be operated on Blockchain.

- 1) Digitalized the objective in a common way. The result by this digitalization can technically be accepted and used by any of the participant. VeChain uses the unified VID to mark the object and make a connection between the hashed data and VID in order to build the corresponded target data to VID. It also helps to IoT technology to complete the connection between VID and real life target.
- 2) To build a relationship type of connection with different object data by using the smart contract.
- 3) To use the abstract smart contract to cooperate relevant authorizations to composite modelling and customization, reflecting the different business activities in the business world.
- 4) A brand new digital asset (VeChain Token(VET)) that can provides the support of high speed value transaction.
- 5) Create a new trustworthy interconnect business model.
- 6) Different business model communicate and merge together to build a distributed business ecosystem.

Through this method, we can “translate” the target product, participants and business activities from the real business world into the world of VeChain. We can combine all enterprise, customers, and government resource and data information from different industries. In this way we can digitalize the cooperation and systematize the operation. So it can reduce the cost of the industries and even the whole society. It can also improve efficiency since resource can be optimized distributed and all kinds of brand new business model will be born.

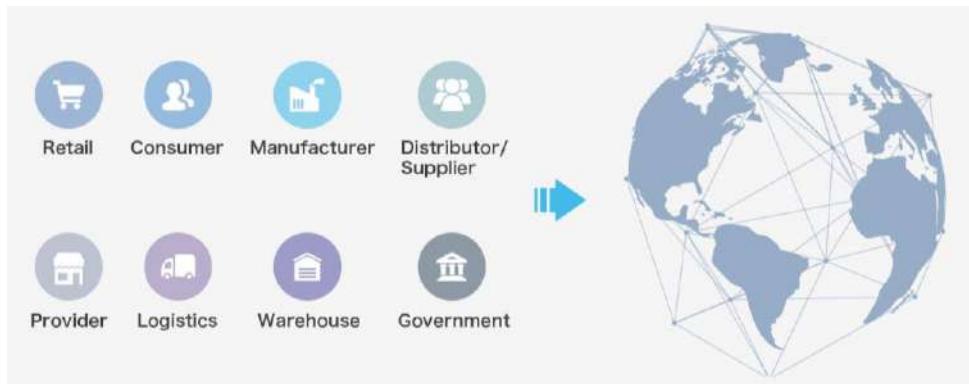


Figure 2.1 Standard digitalization in the traditional world

2.2 Technique Support

The path of VeChain's technology development is almost identical as the Blockchain technology. The initial idea was generated from the middle of 2015 and then VeChain started doing a series of technology verification (TPOC).

At the very beginning of the technology verification, we tried to use Bitcoin network (UTXO) to build the module and use Colorcoin to make VID come true. SideChain technology improves the speed of the trade performance so that the system can face the future business challenge. In the end of 2015, the smart contract of Ethereum has been improved step by step and we worked for it. We also made huge modifications and technical innovation to commercial requirement and Ethereum Fork based on the open source. Based on the customers and project survey, we kept improving the system in the September of 2016. For instance, we increased the data embedding and read performance to 300TX/s and we also improved the data security control in the bottom layer structure of the enterprise. In addition, we made joint development with CHAOS data management model, the IoT technology, the Blockchain technology and many different business smart contract.

Overall, the logic behind VeChain technology is always surrounded by business application. The idea of VeChain's management is practical requirement leads product design, product design leads product development and product development leads practical requirement.

In the whole process of evolving, our technical team got many supports from lots of great leaders, like the founder of the Ethereum- Vitalik, the founder of Jaxx- Anthony and many others. We feel grateful and appreciate of your open mind and the passion to the technology innovation

2.3 Technical Structure

Vechain's structure is based on application needs, and make standardized abstraction for every single technical structure layer. It enables every single layer to have the independent universality and let every model in each layer combine to each other efficiently. The standard unit model have tens of thousands combination of the application.

Below is the figure of Vechain's overall structure:

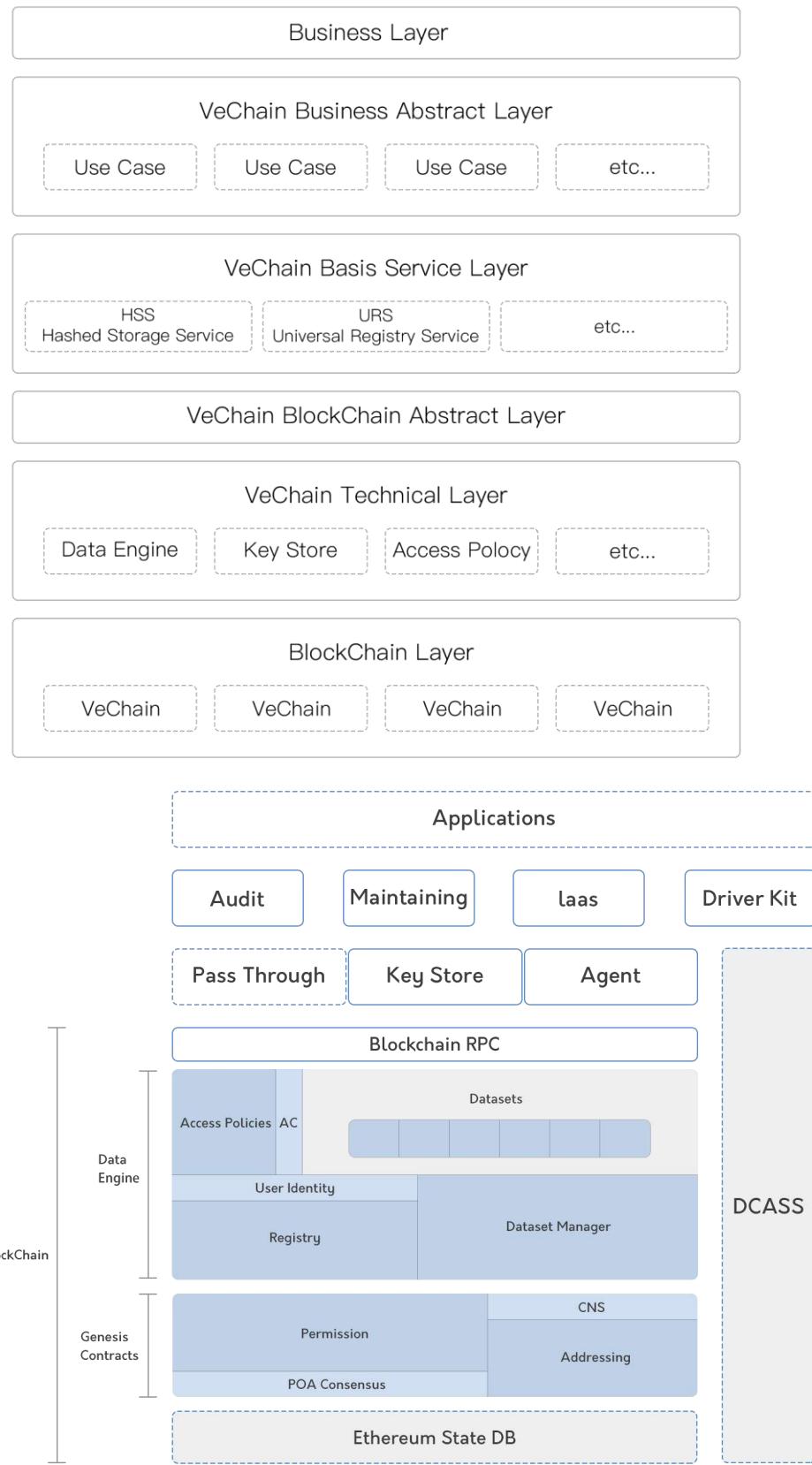


Figure 2.3 VeChain's technical structure

The structure is separated mainly by two parts of abstract layer: Blockchain abstract layer and business abstract layer.

Blockchain abstract layer:

- 1) It is the bottom structure of the basic layer. The technical focus is to fork Ethereum codebase and improve based on this, it includes:
 - a. DBGP- after maintained a certain level of security, then making a dynamic change for Block based on needs and requirements. This way it saves many storage space and system recourse cost. On the other hand, this protocol helps VeChain network to have 3 times better of efficiency than Ethereum after maintaining the security level.
 - b. DMBSP- combine with the traditional safety technique to cooperate with Blockchain's mining system in order to give enterprise level Blockchain application to provide data security protection.
 - c. DGIP- Implanting the data within the same category.
 - d. Under Development- -BLACP- storing Blockchain data with classification and use to differentiate saved data with different time and value.
 - e. Under Development - - PBCP- to distributed implanting and reading data with the same blockchain category.
 - f. Under Development - - DCCP- Syncing the data with different categories of Blockchain.
- 2) The upper level is the smart contract abstract layer for building a standard and modularity smart contract module (SSCU) in order to combine further, customized to face different industry, enterprise and smart contract for application scene (ASCM). Currently the smart contract inventory (VSCL) includes VID registration, data connection, status data implement, digital ownership, ownership transfer, authorization declaring, authorization transfer, multiple authorization and so on.
- 3) Building a Blockchain connector standard protocol (BGAP) to connect with upper business application layer based on the foundation.

Business application abstract layer:

- 1) In this bottom of this layer is the basic service abstract layer. The purpose is mainly to do a secondary operation for the smart contract of the bottom layer to build GBSM. It includes –hashing Storage service, a service model for CHAOS through URS. Meanwhile, this layer contains a special customized module for the data from the bottom layer. It includes index service for Blockchain browser, UDAS, HDMS, DCASS, CNS and DGS which includes the standard basic functions for smart contract on VeChain to save the time on deploying customized smart contract.

For this layer in the future, people will develop tools for visualized smart contract and through the service to build connections with smart contract. So even developers from different industries, or with no Blockchain experience

- can deploy and develop smart contract in order to push the application for the industry.
- 2) On top of this, the interface between the basic service layer and the business application layer is implemented for the two level application interface layer. The core of the development is standardization and to build the connection of business system that faces different types of data. In addition, accumulating more standard types by using application for many big enterprise that faces SAP, WMS and Salesforce, etc. As well as some common used website and mobile application connection.
 - 3) The top layer is business application abstract layer. It has standard application process module for different business scenes, different business practical developing module. So it can make the delivery and deployment for the development of the final application more convenient and fast. The developer of this layer does not even require to have any knowledge of Blockchain development so this can make more developers and technical service providers to use VeChain as the application of final customer development for Blockchain.

2.4 Achieve the Technical Details

VeChain's Blockchain is forked and improved based on Ethereum codebase. The basic technical index can see Ethereum's whitepaper as reference.

Below we are going to focus on discussing the technical application feature of VeChain.

2.4.1 VeChain ID Creation and Hashing

VeChain IDs are created by using a sha256 function which generates a random ID which is hashed before being written into a NFC, QR Code or RFID tag(s) to be used for each product.

All IDs are hashed by using a sha256 function which goes as follows: **SHA256(domain + '!' + ID)[12:]**. In which the domain is the qualified name of table that the ID settled. e.g. "**com.VeChain dbname tablename**"

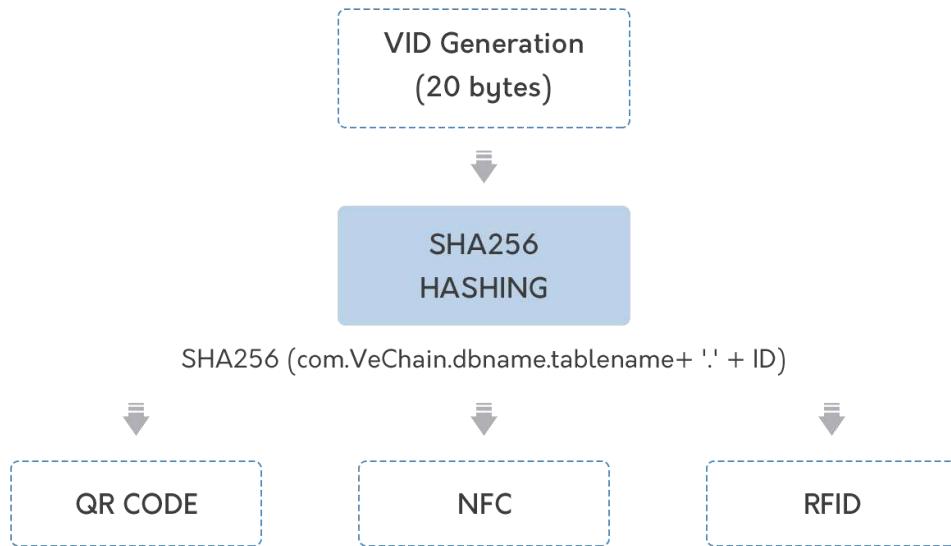


Figure 2.4.1 the creation of VID and hashing

2.4.2 Storage of VID on Blockchain

As previously mentioned, the hashed VeChain ID is written into tag(s) depending on the client's needs. After the tags are ready, they go to a testing process and are “activated”. Activation is done by using a custom-made software called “V-Operation” which can either run on Mobile or desktop operating systems. Upon activation, the ID is then written into Blockchain and replicated among all nodes.

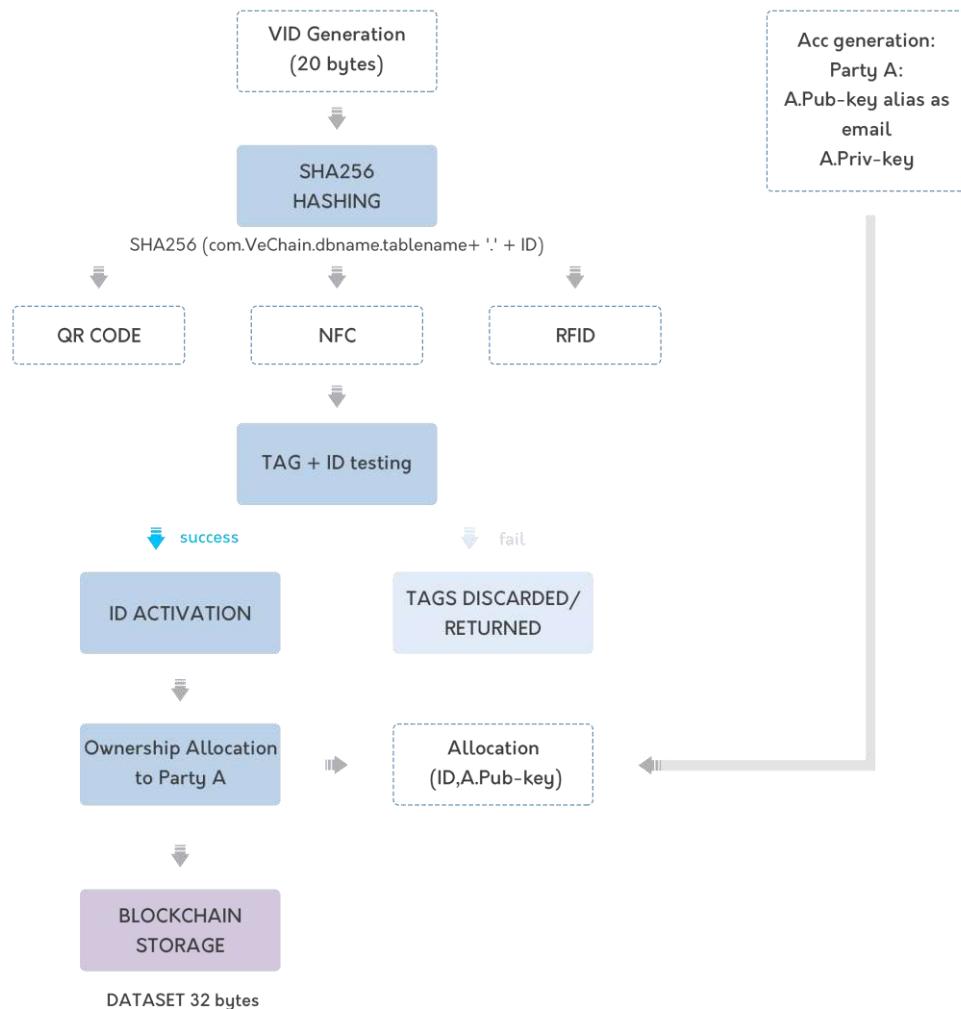


Figure 2.4.2 The storage of VID on Blockchain

2.4.3 Digital Ownership on Blockchain

VeChain uses a custom-tailored Smart Contract which enables **authorization-based digital ownership management**. The ownership of objects, represented by VeChain ID is linked to an account with the key pairs combined with public key and private key.

The public key is public and known as alias email address which can be recognized and accessed by anyone. The private key is to represent the authorization and access, just like a password, to the objects with the corresponding public key. The ownership management is to set a specific linkage between the objective ID and the public key of owner who controls the corresponding private key.

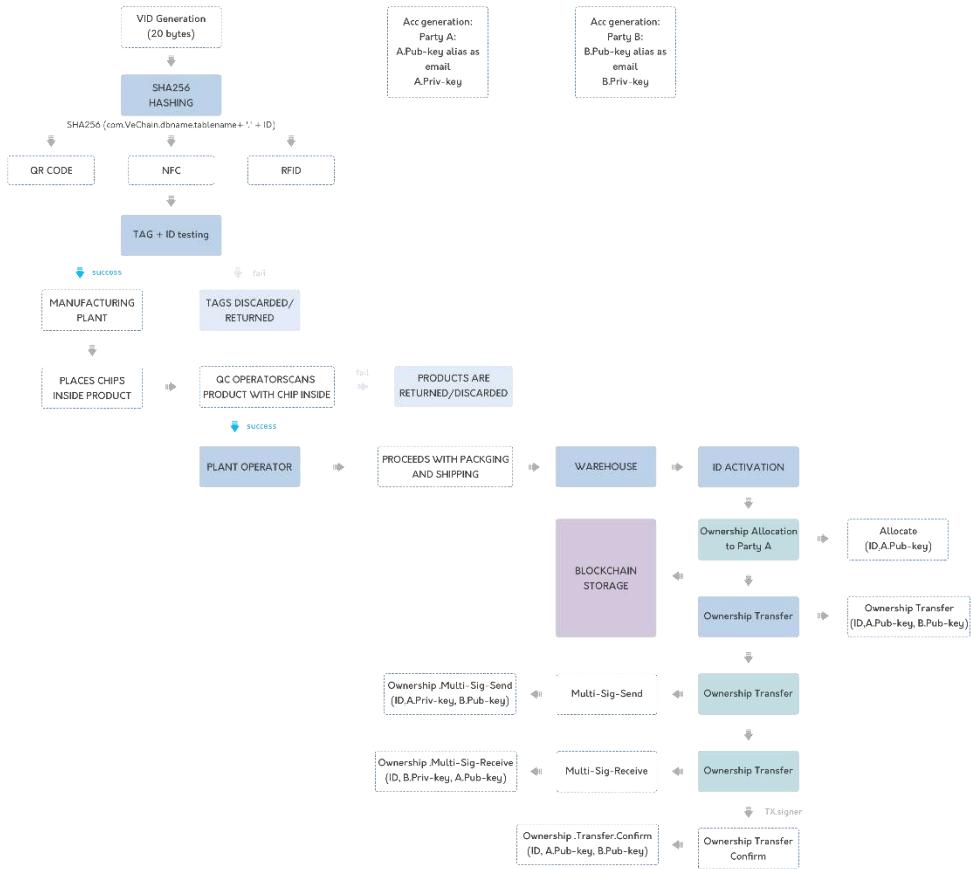


Figure 2.4.3 Digital ownership on Blockchain

2.4.4 Data Hashed Storage (proof of data)

VeChain accepts any type of data: (strings, numbers, booleans, etc). Data is identified by its hash (SHA256). Sample for accessing data via RESTFUL APIs:

- **Store data**
- POST <https://domain/hss/>

- **Retrieve data**
- GET <https://domain/hss/{hash}>

The data is self-verifiable. When the data is retrieved, it can be verified by comparing its hash to the hash provided.



Figure 2.4.4 Data hashed storage

2.4.5 API Gateway

Universal application architecture interface designed for complex processes. The API gateway is the main entry for all API requests, it encapsulates the internal structure of the application, and the client only needs to interact with the gateway without calling a particular service. When the internal structure of the upgrade or new features, the client is completely transparent, the client does not need to consider too many changes in access, only need to ensure that the exchange protocol is correct.

The following is about the network topology graph of the API gateway, deployment graph and functional graph.

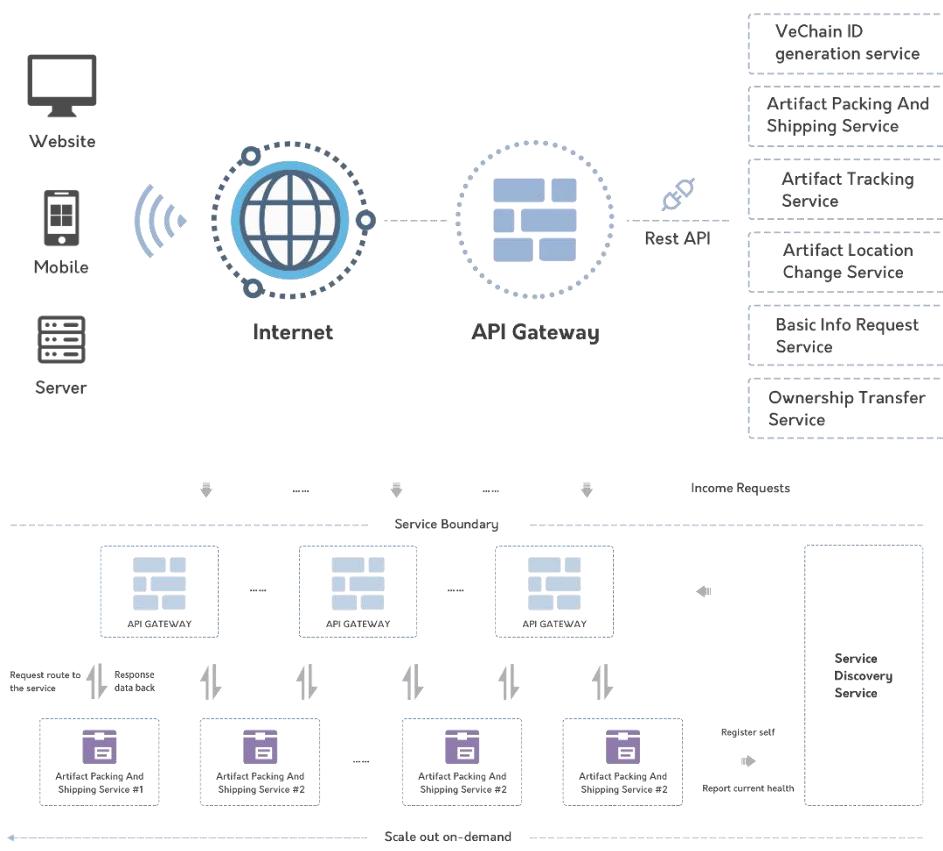


Figure 2.4.5-1 API Gateway-1

The resources of a server are limited, and the characteristics of the horizontal expansion make it possible for large-scale access. Different instances of the same service can guarantee a service request shunt by API Gateway. In API Gateway we can use different access policy like consistent-hash, ip-hash, random access or priority access. At the same time, API Gateway and Service Discovery Service also can be scale out on-demand.

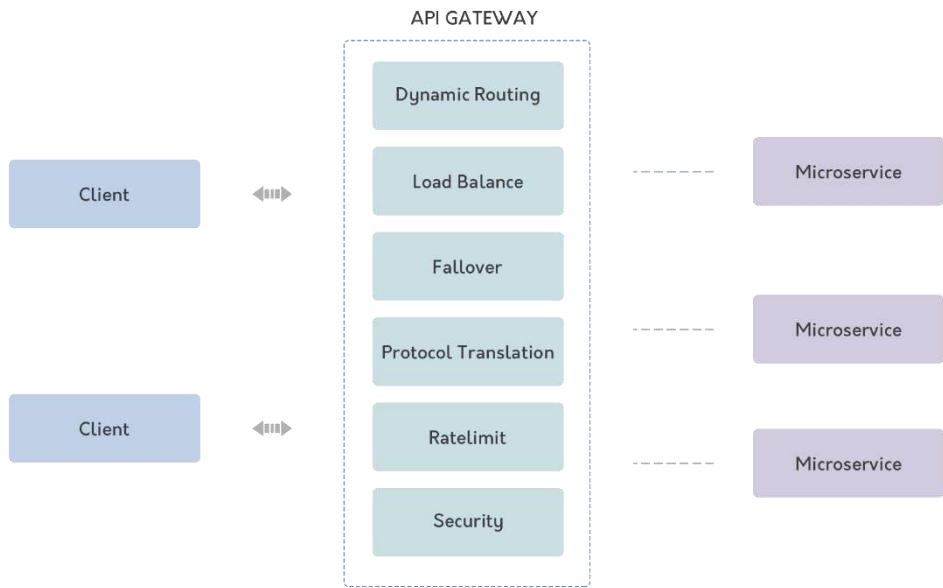


Figure 2.4.5-2 API Gateway

2.4.6 Service Discovery (SDP)

The API gateway needs to know the location (IP address and port) of each micro service that it communicates with. In traditional applications, it may be hard to connect this location, but now that it is based on cloud-based micro-service applications, which is not an easy problem to solve. Infrastructure services typically have a static location that can be specified by the OS environment variable. However, determining the location of an application service is not that simple. The application services are dynamically allocated, and a set of instances of a single service can also change dynamically with automatic scaling or upgrades.

Service discovery has two major modes: client discovery mode and server discovery mode. We are using the server-side discovery mode. The client makes a request to a service through the API gateway, the API Gateway queries the service registry, and forwards the request to an available service instance. The biggest advantage of the server-side discovery model is that the client does not need to focus on the details of the discovery, simply sending the request to the API gateway, which reduces the discovery logic that the programming language framework needs to complete.

The service registry is the core of the service discovery and is the database that contains the network address of the service instance. The service registry needs to be highly available and updated at any time. The service instance we use the self-registration mode. So the service instance needs to be responsible for registering and logging out in the service registry. In addition, a service instance

also sends a heartbeat to ensure that the registration information is not obsolete.

We choose etcd as a backend high availability, distributed, consistent key store for shared configuration and service discovery.

2.4.7 Micro-Service

Micro-service is the generic term for all backend VeChain services. This type of service can be customized according to the actual business interface to keep the separation between different businesses. Micro service can guarantee the service gray scale release, fast service upgrade or downgrade level. In our API Gateway ecosystem, the micro-service should provide below basic functions.

1) Register& UnRegister

Micro-service must be the initiative to register self to Service Discovery Service (SDS) when start up and must be unregister self when shutdown. SDS has 30sec to hold instance states, if unregistered when shutdown, after 30sec it will also be removed from service registry.

2) Report service health

SDS never know whether instances at backend are still available for serving. So microservice must report self-healthy in time and report interval must less than 30sec.

Micro-services are more complicated than traditional service, especially communicating between service and service at the backend. Currently service discovery service need instance to register self, so all instance need a logic that register to it. In the next time we should consider a 3rd party of register service for leverage. These service can deploy an instance of micro-service, and can config some information for it, can check instance health and report to SDS. So micro-service just can be consider a pure app for serving API.

2.4.8 Hashed Storage Service (HSS)

Hashed Storage Service (HSS) is a distributed storage service, which provides services such as digital files, pictures, text data and any other object-oriented reliable storage. Through the combination with VeChain, stored in various types of objects, HSS will ensure that the data cannot be tampered with. At the same time, the uploaded object, unless authorized by the uploader, otherwise it cannot be obtained and modified by improper means.

The HSS is mainly composed of two parts: object storage service and basic storage service. Object storage service is responsible for external interaction, save the object, access to objects and authorized object access; Basic storage

service is responsible for computing the object storage footprint, cut the object and the actual storage.

With the development of service scale, in order to system reliability, Data often need to be backed up. The number of backups per data is at least one copy, or even ensure the reliability of the backup, the number of backups may be two or more. This makes the storage utilization rate of only or less, each TB of data need to occupy at least 2TB of storage space. As data grows, the cost of replication becomes more and more obvious, with traditional replicas equivalent to at least 100 percent more storage overhead. Why distributed storage can help you setup a highly-available storage system with a single object storage deployment. With distributed service, it can simplify the data backup program, reduce disk space, but also make services available, improve data's availability and durability.

Erasure Code & Reed Solomon In the storage system. The erasure code technology is mainly through the use of mathematical algorithm to verify the original data to be verified to achieve the purpose of fault tolerance. It can be used to reconstruct missing or corrupted data. Reed-Solomon (RS) code is a storage system that more commonly used in an erasure code. Our Storage Service use RS algorithm to shard objects into and parity blocks. You can lose as many as drives (be it parity or data) and still reconstruct the data reliably from the remaining drives. Amazon S3 Compatible the APIs In the low level is compatible with the Amazon S3 API.

Compatibility will lead to data access bonuses. Most developers are familiar with Amazon's S3 service and are familiar with how to use its API. A compatible interface reduces the possibility of external access.

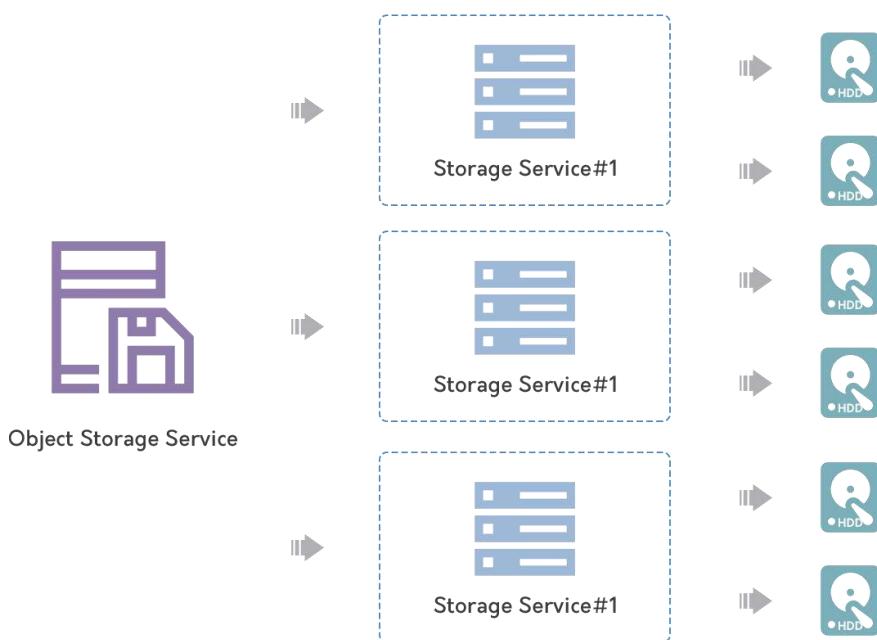


Figure 2.4.8 Hashed storage service

2.5 Blockchain and IoT

In 1999, the concept of Internet of Things (IoT) is proposed by a British Scholar Kevin Ashton in MIT. After discussed with many senior level managers from different enterprise, Ashton defines IoT to:

“A network containing all ‘smart’ devices with some sort of sensing mechanism that can communicate via the internet with other smart devices or the cloud, without human interaction.”

We can see IoT with mainly two parts – Perception and Connection based on this definition.

- ✓ Perception is the idea of digitalized the real world data, like temperature, humidity, similar environment type data and personal ID.
- ✓ Connection is sharing the digital information through Bluetooth, WIFI and mobile internet.

In the future, the trends of IoT will be widely used, more different types, high sales and fast development. It will have a huge impact on the global industries. By quantity, the sales amount of IoT equipment will increase as a rate of 15% to 20% in the near future. IDC predicted that sales amount of IoT equipment will hit 45 billion in year 2020.

2.5.1 The Issue of IoT

The issue of IoT technology has been discovered in the nineties. Until June of 2016, 3GPP announced “release 13” and defined the IoT connection standard. This solves the four issues of IoT: Limitation of the connection capacity, limitation of coverage area, low standby time and high cost. Since September of 2016, every mobile device manufacturers released IoT connection plan for business. This focus the concept of different application can choose eMTC, NB-IoT, EC-GSM and other different technology.

With the new IoT standard, it has the feature of great connection, huge coverage area, low energy waste and low budget cost. Since then, IoT industry started to grow rapidly. However, we believe there are still three major issues for IoT: fragmentation of the standard Communication protocol, high cost of development and maintenance and lack of privacy, but these issues can be solved by Blockchain perfectly.

2.5.2 Blockchain and IoT

Currently, there are many exploration on different IoT and Blockchain application on smart system. When those are applied on IoT, the concept of IoT opened up a road of innovation with unlimited possibilities. The Blockchain technology can be used as tracking the usage history of different equipment. It can also help to complete the trade with different equipment. This technique can provide the data transaction within different equipment that makes IoT equipment independent.

Blockchain will realize the self-management and maintenance of the equipment. It saves the huge maintenance cost of cloud system and reduce the maintenance budget for IoT equipment. The private key that made by the equipment will ensure personal data won't be stolen by strangers. It increases the safety level of IoT and the economic effectiveness with the combination of these two concepts.

We believe that the IoT technology and the Blockchain technology cannot be split in the application. From the view of self-development for IoT, IoT wants to build a world with everything connecting to each other and this process requires three steps:

- 1) It requires a unified communication standard between one thing and another. It means the equipment can communicate with the same language no matter where this equipment was built. IoT standard that published by 3GPP provides a physical channel for thing. In addition, the Blockchain technology provides logic language for thing and this makes different types of IoT equipment communicate with one unified language.
- 2) With the foundation of the unified language, personal Identification will be required during the communication between different things. Then it needs a standard identification code system to support the unified language. This means the Blockchain technology is the best solution to unify different manufacturers and the identification code will not be controlled by anyone.
- 3) With unified language and personal identification, the connection between those two will require more cooperation and business activities. This means smart contract needs to be built and digital currency will become the transmission carrier since the value needs to be transferred during the cooperation at the same time.

As a technology that provides the service of trust, Blockchain can ensure the true effectiveness of data on the Blockchain network, IOT is the key to ensure the true effectiveness of data when it's been uploaded from the first time from the original source.

On the one hand, IoT helps to establish the congruent relationship between the real physical world and Blockchain world. On the other hand, the IoT technology can reduce the disturbing factors from the source to ensure the true effectiveness of data.

2.5.3 VeChain and IoT

VeChain technology team contains a very important component – the IoT technology team. They focus on the responsibility to coordinate the IoT development with Blockchain application, which includes:

- 1) Encrypted chips tag technology development.
- 2) The identification of IoT sensor and data security.
- 3) Security and authorization module of NB-IoT.

IoT equipment is very complexed and we need to classify them with different point of views:

- ✓ From the point of view of power supply
 - Equipment with power source: equipment with battery equipped, like temperature sensor, GPS.
 - Equipment with no power source: equipment with no battery equipped, like NFC.
 - Mixed equipment: Equipment with battery equipped and can also get power from other places.
- ✓ From the point view of communication distance:
 - Close range distance equipment: resolve the communication within 10 meters distance, like NFC(1 meter), RFID(10 meters), bluetooth(10 meters), etc.
 - Middle range distance equipment: resolve the communication within 1 kilometre, like WiFi, sub 1g, lora, etc.
 - Long range distance equipment: resolve the communication with more than 1 kilometre, like NB-IoT, etc.

We upgrade traditional IoT equipment on the chip layer and put personal identification with asymmetric key algorithm.

- ✓ **Personal identification:** Every IoT equipment requires a unique identification on the network and this ID can be identified by other participants within the network. We enciphered each different equipment or object to ensure the code is unique and cannot be recognized.
- ✓ **Asymmetric key algorithm:** asymmetric key algorithm is the foundation of the internet and the important feature of Blockchain. It can identify and authorize the equipment based on the identification of equipment. Through public and private key algorithm, we can identify if the equipment could connect to the network, if the digital data source were reliable, or if smart contract can be operated. To ensure the safety level of asymmetric algorithm, we will put the private key in the safety area so it cannot be read. In addition, asymmetric algorithm runs with the processor's safety mode and it can ensure the safety during the process of calculation.

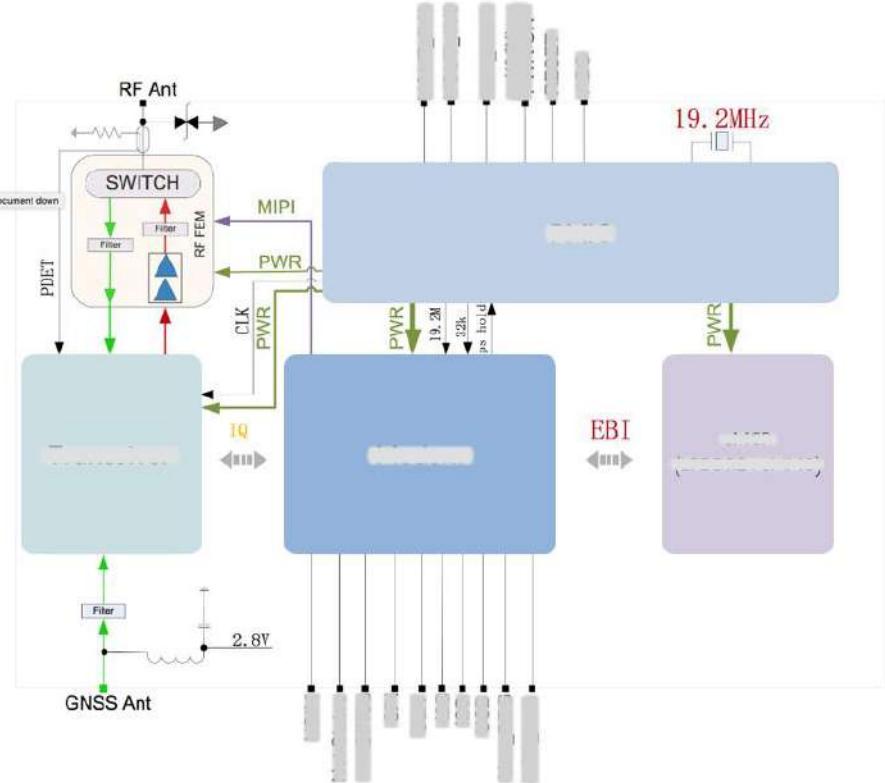


Figure 2.5.3 NB-IoT safety module

2.6 Technical Testing

VeChain team follows the procedure of professional software testing. Software should be predictable and stable and meet the standard of the product so there should be no unexpected results. With the digital information keep expanding, complicating and smart, the development of software testing is improving as well. With the maturation and systematizing of theory and practical, it shows more impact during the software quality check. General experience suggests that in a typical coding project, software and system testing cost about 50% of project time and over 50% of total cost.

The most concerned question that software testing care about is in which subset we can find the most problem during all the possible testing. We can put the testing into three category:

- **White box testing:** also named structure testing. This method regards testing software as white box. Based on the internal structure and logic, software is designed as testing sample to proceed a testing for the program's path and process. White box's main technique includes statement coverage, branch coverage, design coverage and elementary path coverage.
- **Black box testing:** also named functional testing. This method sees testing software as black box. Without considering of internal structure and

characteristic to test software's external's characteristic, black box technique mainly includes equivalence class partition method, boundary value analysis, cause and effect diagram method, status mapping method, measurement method of outline and many typical malfunction model.

- **Gary box testing:** This is the test between white and black box testing. Gray box testing mostly used on integrated test phase. It focus not only the validity of output and input, but also program's internal condition. In addition, Gray box testing is not detailed and completed like white box testing, but focus more on internal logic than black box testing. It usually estimates the operational status through certificate of phenomenon, event and symbol.

Team VeChain sets a special testing team and take on the role of "quality management". The purpose is to make corrections in time and ensure the smoothly operation. Thus software testing is mainly for verification and confirmation. The target for software testing is not just program testing, but also includes all the documents from different phases of development, such as testing guidance book, testing project plan and testing report.

- Testing guidance book: describing the testing requirements and theories
- Testing project plan: describing testing sample and testing methods
- Testing report: output testing results

Testing is a process of convergence step by step and it strictly follows PDCA quality circle. The testing process from the description above is only one part of the PDCA. PDCA is acronym for Plan, Do, Check and Adjust. The PDCA circulation perform the quality check and it will keep going like this by using this order.

- P (Plan) includes the confirmation of the testing plan that contains unit testing, integration testing, system testing (function, performance, safety and compatibility) and examine testing.
- D (Do), based on the testing plan to perform the test.
- C (Check), the final testing result and reflect the feedback to development team.
- A (Adjust), development team improve and fix the original code based on the testing results.

The goal of VeChain's software testing:

- Lower computer(PLC): Embedded software of IoT parts
- Client: PC end, mobile end (ios, Android)and terminal software
- Cloud
- Server: software on Website and server
 - Blockchain part
 - Smart contract part
 - Interface part

For individual and interface between different parts on the four goals, we have professional testing team defining the testing guidance book, testing plan, and output result report. In addition we will use quality management methods by PDCA to complete the software testing and ensure the quality of VeChain's product.

This is part of the pressure test data result:



Figure 2.6: Part of the pressure test data result. Testing environment is using the lowest cloud server configuration with different locations from global. The basic set up is single core 2G server.

2.7 Technology Development's Path and Plan

The development of VeChain technology has been through two years and the core of development focuses on three areas: application, standardization and

safety. VeChain team will keep following these three basic ideas to continue the development.

VeChain technology team has three units:

- 1) R&D – focus on the bottom level of the technology and development, as well as with the newest technology analysis and experiment. In addition, they will make plans for the next generation's possible path and feasibility analysis.
- 2) Development – Based on the result of R&D, to perform and complete the development and get the initial testing result.
- 3) Testing, deployment and maintenance – Based on the development result, R&D need to improve and correct the test result as well as taking care of related deployment and maintenance.

Below is the path of the VeChain technology development and future plans:

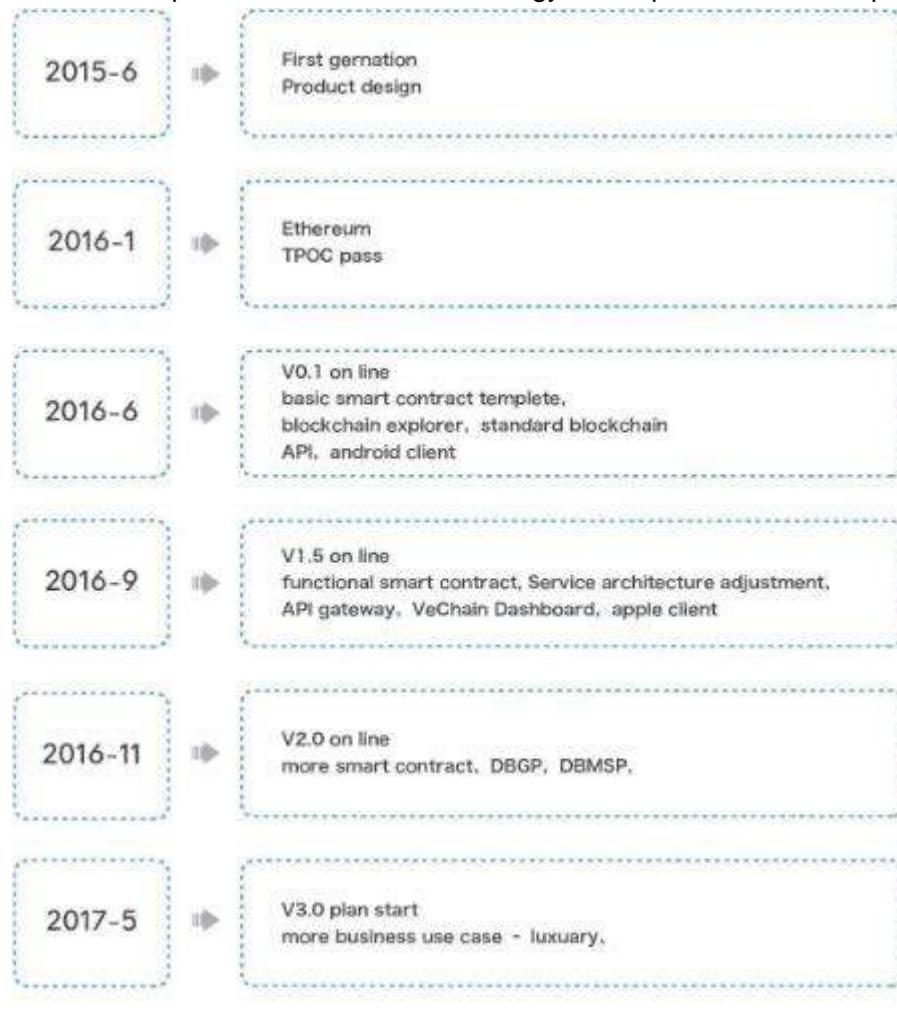


Figure 2.7-1 Technology development path

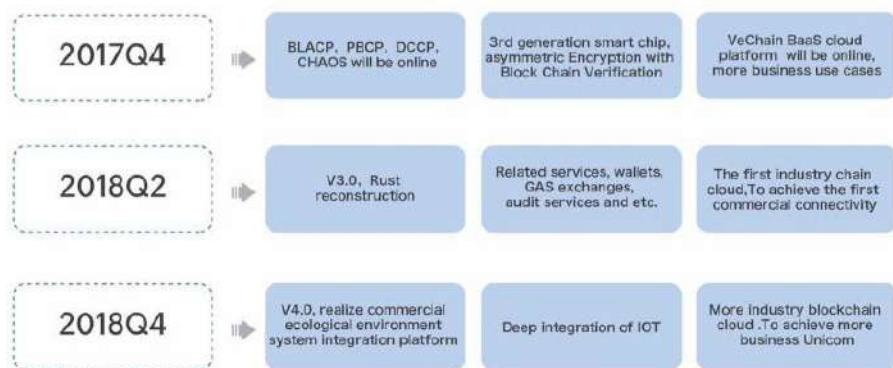


Figure 2.7-2 Future technology development path

3. The Industrial Application and Expansion

In the past two years, VeChain has gained great amount of experience from many different fields of the work and some clients are world famous firms. The figure below shows the application structure of the VeChain:

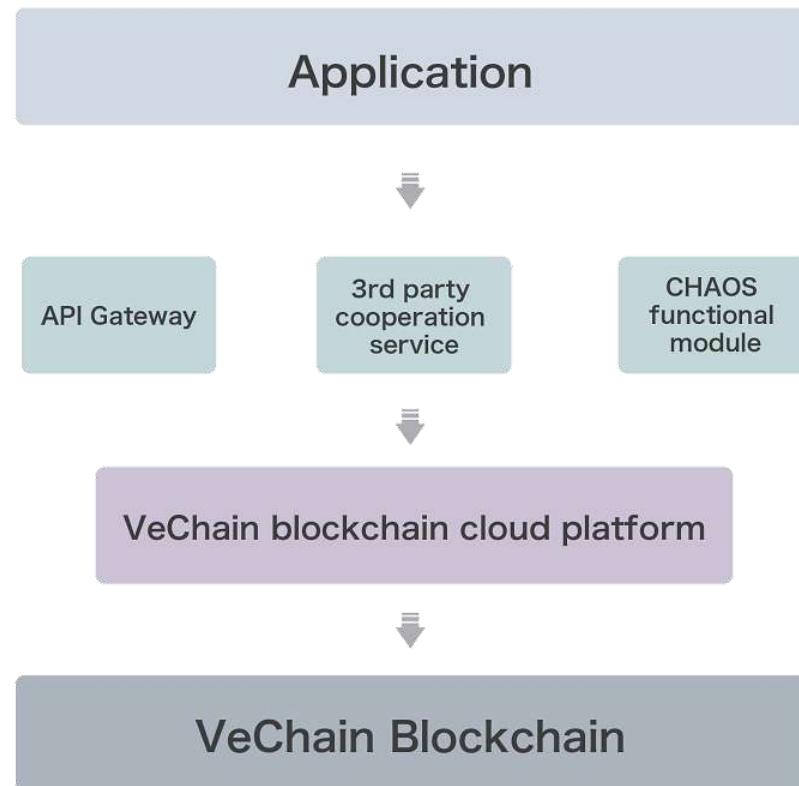


Figure3-1 Industrial application structure

Based on the bottom structure of Blockchain system, VeChain loaded a platform that is convenient to use for the business partner and a one-click deployment Blockchain Platform. Users only need to select the industry they are in and the solution plan they want, the system will allow the user to manage their own Blockchain nodes, smart contract and generate related API setting.

Meanwhile, together, VeChain and cooperative partner are developing a third party application which is focused on Blockchain to serve VeChain's customers. For instance, VeChain is developing a unique Blockchain audit service with a world famous accounting firm. It provides data auto-collection and audit suggestion on Blockchain overall operating states, smart contract status, business operation status.

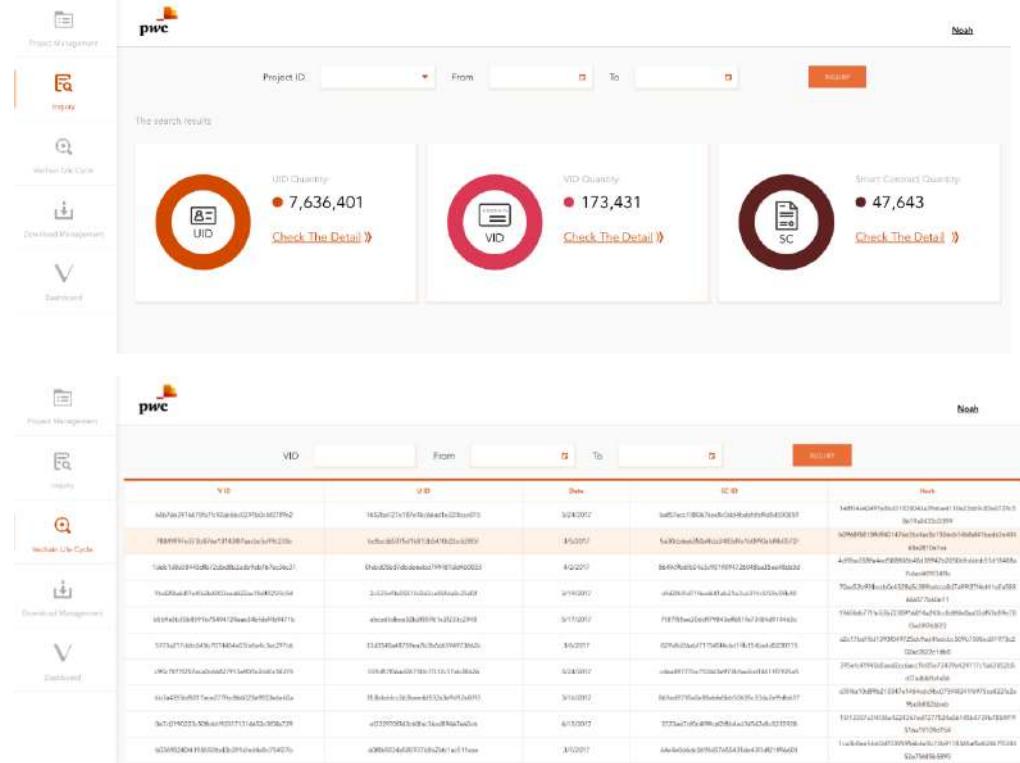


Figure 3-2 Third party service application

In addition, the application structure of VeChain also includes self-developed distributed encrypted database service- CHAOS, user private key management and smart contract authorization, etc. This modular service model makes customers and service provider's development more convenient and flexible.

VeChain plans to use these successful cases as template to expand and develop more quickly. VeChain wants to let more enterprise and business activities operate based on the VeChain platform. In addition, establishing the connection with these business activities step by step. At the same time, through developing related business smart contracts to promote the circulation of VeChain Token(VET) in order to complete and expand VeChain's distributed business ecosystem step by step.

3.1 Fashion and Luxury Industry

According to the survey from year 2015, it costs fashion and luxury brand of Europe 9.7% of the total sales every year, about 2.87 billion dollars, for anti-counterfeiting. Due to the overrun by the fake goods, it costs Europe lost 363,000 jobs in fashion, manufacturing and retail industry each year.

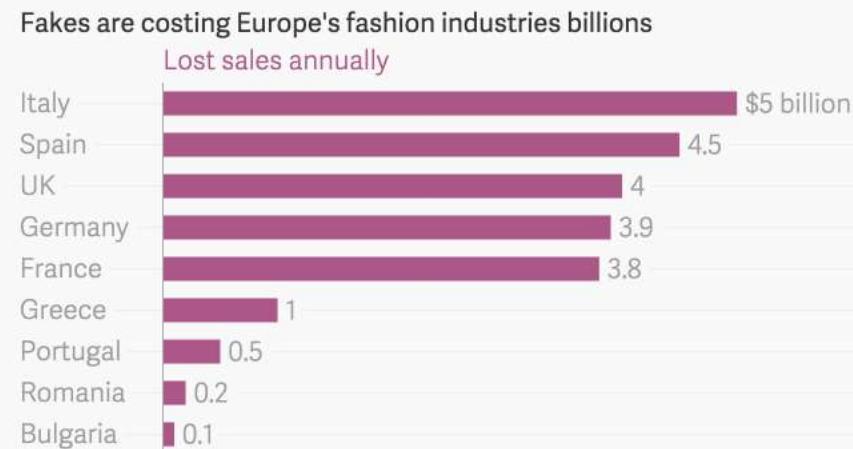


Figure 3.1-1 the impact of fake good to the Europe fashion industries

VeChain aims to focus on this industries by following production management, production channels, anti-counterfeit and the connection with customers. The third party makes product based on the order plan and at the same time, the brand party can “activate” this product when verifying it. This is the process of Chain to verify the enterprise’s SAP order form and authorize the real thing from the beginning of the process.

Meanwhile, VeChain establishes the data connection of WMS with the enterprise and complete the Blockchain process with dealers and retailers sales channel, in order to achieve the management to the sales channel. For the customers, VeChain provides a digital ownership to build a bridge between the brand and customers and keeping the transmitting to CRM and after-sales service. In order to establish a customized individual service, after-sales service and customer care for customers. It can even track the trends of the second-hand market. At the same time, the anonymous privacy protection feature of Blockchain perfectly fits the safety rule of EU’s GDPR (General Data Protection Regulation).

VeChain makes business with famous Europe luxury brand by putting IoT encrypted chips that is based on the Blockchain technology into product. As the medium of the data collecting for Blockchain, the chip records the every “logistics”, warehousing and transmission. After the customers get the product, they can scan the chip behind the tag through the APP and they can know the “history” of this product. So the customers can identify the authenticity of the products, and making a statement of the product’s digital ownership at the mobile end.

3.2 Food Safety

Food safety directly affects people’s health. Since the budget of being healthy has growing all the time, both producers and customers would care a lot about food safety. However, traditional food safety relies too much on the process control and

entrepreneur's sense of responsibility. So it is pretty hard to ensure the safety of food by using automated methods. It is quite difficult to track the original source if any problem occurred.

Yet the Blockchain technology could bring safe and reliable solutions to the food industry. The Chinese government has already confirmed that food certification and tracking through supply chain are the key steps to find and eliminate the source of pollution in a very fast way.

The Liquor tracking platform in Waigaoqiao free trade zone, which built by VeChain, can track the liquor product from the very beginning of the process, even when the liquor was still in the overseas winery. This is the first successful case in domestic. Every details about the bottle of red wine is marked and recorded at the beginning of the process. This way company can use smart contract to track the whole life period of the red wine, from the warehouse in the free trade zone to for the first time. From the distribution center and finally reach to each different sales channels and stores.

Customers can identify and track the information of the wine through the in-store touch-screen or even customer's own smart phone. High-end wine are also equipped with IoT Chips with the feature of safety and convenient. Customers can use their Cell phone to check these information as well which increases the security level.





Figure 3.2-1 VeChain application in the wine Free trade zone: background management system, smart front terminal, mobile showcase

For the next part of the plan, we are going to let more oversea wineries and welcome more imported product providers to join into our platform, so customers can feel safe about the product. We are also going to open the connection and start the cooperation, let the customer see more information about the product.

On the other hand, the product that VeChain is following right now is one of the most focal point product: dairy, dairy food safety is a hot topic that people pay huge attention to. The regulation requirement is stricter, especially when the news about domestic milk powder contains melamine and makes the milk powder toxic. Mengniu Cooper has to increase the breadth of the product sale and the multiple security check to ensure the source of the dairy product is safe. This is what they paid for the huge loss of customer trust. In 2007, Mengniu corporate spent 3.302 billion Yuan in sales and distribution cost, and the number increased to 4.428 billion Yuan in 2008. The advertise expense in sales percentage has increased by 2.1 percent to 9.3%. This is merely the cost for losing trust of one single enterprise. Although this event has passed for so long, the dairy industry is still paying for this. We can definitely make more example of this, it shows that the cost of trust is almost unbelievable.

VeChain provides the standard adding Chain function that let all participants upload the data by using the permission to complete a Chain. (Like Figure Below)



Figure 3.2-2 The process of Dairy industry and ID code

VeChain can help Dairy Company by providing farm information, including Fertilizers management, the audit for feed supplier, the healthy condition of the livestock and the drug use on the cow and environment report. This process ensures the food safety from the very beginning the milk source. VeChain can also help the production supplier to check the receiving time of the raw milk, the storage condition of the raw milk, production reference number, and the detail information of processing personnel and people who are responsible. In addition, Package storage can use the technology methods to tracking the temperature and humidity while transporting. Moreover, the production's loading information of distribution center and data information can also supported by VeChain. Finally, owning these information cannot just prevent counterfeit product but it also increases the product supplier and customer's faith for the industry.

3.3 Car Industry

Industry Chain for the Car industry is very complicated. There are many participants, like manufacturer, different agents, regulator, financial service provider (Insurance, Bank) and personal account. In the life circle of a car, there are a big portion of the “user data” are never owned by customer, instead these data are separated in the pocket of different participants. This causes many difficulties of car information collection and verification.

VeChain and Europe strategic partner Visco, Microsoft invent a verification idea for many international car enterprise together. In the project, VeChain team is responsible for completing the Blockchain deployment on Azure, developing and deploying smart contract and provide standard API to the upper level developers to complete the final product.

Digital maintenance principle: Every car can build their own digital record and build the authorization of the ownership. After car owner bought the car, they can use authorization and non-authorization feature to give permission by the server maintenance suppliers. So every single maintenance data is recorded to the Blockchain. This way the data provided by different maintenance service provider can build the real grouping record step by step. For instance, insurance, bank and other financial service provider can provide fast insurance and value assessment based on the data supplied by this trustworthy network. The operating expense that saved by this can be returned to the car owner.

“Green Driving”: Green driving is the shared electromobile project provided by this car enterprise. Every customers can record related driving records by internal computer in car and upload them to the Blockchain and connect the data together which owned. In the future, these data can proves the beneficial information of the certain project like “carbon emission” and even customer data can be used as

individual creditability.



Figure 3.3 Showcase of VeChain usage by a famous car enterprise

3.4 Supply Chain Industry

Traditional supply Chain includes: original material supplier, manufacturer, agent, logistics, customs inspection agency, storage, retail and finally customers.



Figure 3.4-1 Traditional supply chain

Traditional supply chain industry is facing many problems includes:

- 1) It is quite difficult to track supply chain since its cross-region
- 2) It lacks of transparency between different supply chain and information
- 3) Data security vulnerability in different enterprise of supply chain
- 4) Money flow transport has bad timeliness

VeChain provides Baas (Blockchain-as-a-service) service to one of the biggest freight forwarders K+N to track and manage all the products from many world's famous brand. To ensure the data protection and privacy as the precondition, VeChain completed the connection with different customers through a common service platform. The operation staff can complete related business work by directly using the handheld terminals.

SKU	Product Description	Quantity	Status
WYDYIW12979237GFCAS	Blue leather handbag. With a small shoulder strap. The bag has a rectangular shape and a chain shoulder strap. It features a front flap closure with a magnetic snap and a top zipper pocket. The interior has a single compartment with a small zippered pocket.	1434	PENDING: 131
WYDYIW12979237GFCAS	Blue leather handbag. With a small shoulder strap. The bag has a rectangular shape and a chain shoulder strap. It features a front flap closure with a magnetic snap and a top zipper pocket. The interior has a single compartment with a small zippered pocket.	1434	PENDING: 131
WYDYIW12979237GFCAS	Blue leather handbag. With a small shoulder strap. The bag has a rectangular shape and a chain shoulder strap. It features a front flap closure with a magnetic snap and a top zipper pocket. The interior has a single compartment with a small zippered pocket.	1434	PENDING: 131
WYDYIW12979237GFCAS	Blue leather handbag. With a small shoulder strap. The bag has a rectangular shape and a chain shoulder strap. It features a front flap closure with a magnetic snap and a top zipper pocket. The interior has a single compartment with a small zippered pocket.	1434	PENDING: 131

Figure3.4-2 The showcase of logistics issue solution by the famous freight forwarders

In the late period of the plan we are going to make connection with more related cooperative partners, service provider and regulators.

3.5 The Agricultural Industry

The Chinese market is facing many critical issues like the scale of the agriculture

which is too small and separated, the quality of the product is uneven, the lack of the safety level of the product, low productivity and environment pollution. It is quite hard to fix the issue completely by simply using a certain technology from the internet or a law regulation provided by the government. We can only change the thinking model by using the technology like the special Blockchain cloud project that is exclusive for the verification of the green organic agricultural in order to use industrial management to build modern agricultural.

China is promoting Agricultural Cultivation Management Plan by using IoT technology, agricultural planting process management, the Blockchain technology, big data and AI (artificial smart) to complete the management of the process before, in the middle and after the agricultural production. In this way, good currency drives out the bad currency to achieve standard agricultural market.

With this background, VeChain is cooperating with PwC, China Unicom and Liaoning academy of agricultural sciences to develop the special Blockchain cloud project that is exclusive for the verification of the green organic agricultural.

In this project, VeChain has registered the greenhouse for every farm by using the Blockchain tech to build a data model to record the functional data of every greenhouse. Data source has two main parts: the first part is the production operation data which recorded by the famers directly; and the second part of the data comes from the IoT sensor in the greenhouse. Based on the combination of the data and risk assurance service from PwC, it will establish the foundation of the trustworthy data for the green agricultural verification by the academy of agricultural sciences. In addition, with the support by the IoT equipment, it improves the efficiency of the farm work by about 9 times.

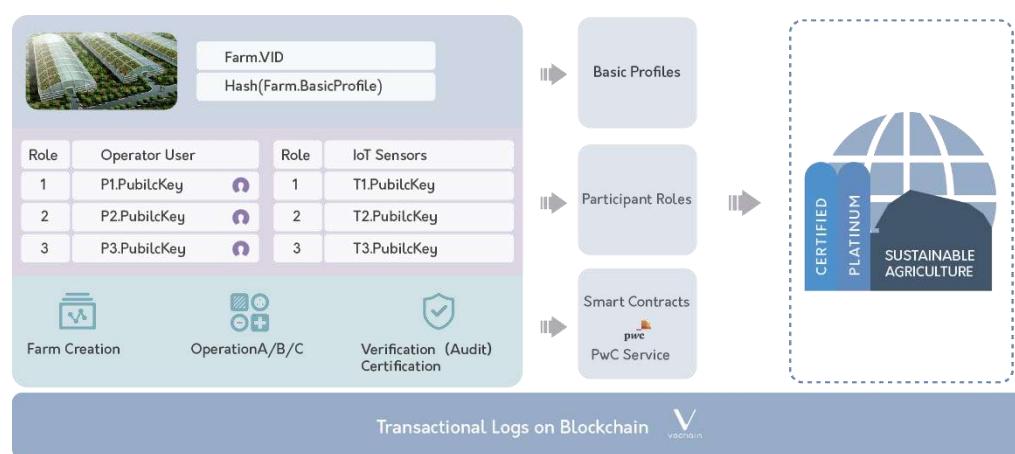


Figure 3.5 Application of agriculture combines with IoT technology

3.6 Blockchain Government Affairs

Government agency shows interest in the Blockchain technology. China ministry

of industry and information technology has released a white paper about the application and development of the Blockchain technology. The State Council underlined that Blockchain can bring a world with trust.

The science office of British government has reported the potential qualities and advantage of Blockchain technology in the recent report: "Distributed ledger technology has the potential to transform the delivery of public and private services. It has the potential to redefine the relationship between government and the citizen in terms of data sharing, transparency and trust and make a leading contribution to the government's digital transformation plan."

VeChain has signed strategic cooperation agreement with local government with big data collection to build case project for Blockchain Government affairs. VeChain has a targeted plan in some very typical application area of the Blockchain technology.

For instance, commodity inspection is always reported by manpower to the inspection agency, then agency will inform the client for the material of random sample. Because of the system that agency use is so different than the others, also agency and client are only using manpower to report and delivery every information transaction. This may cause the unmatched information, very long process, low efficiency, and even the risk of data manipulation.



Figure 3.6-1 Blockchain Government application

VeChain will share related data on the Blockchain platform, agent and the client both can check the random sample date through VeChain's APP. This makes the whole process paperless and complete the connection between Blockchain ID and waybill number, data ID and waybill number, then finally smart contract ID

and the related operation function. The goal is to reduce the paper use from 20-30 pages to 2-3 pages per file. This will increase the efficiency by 80%.

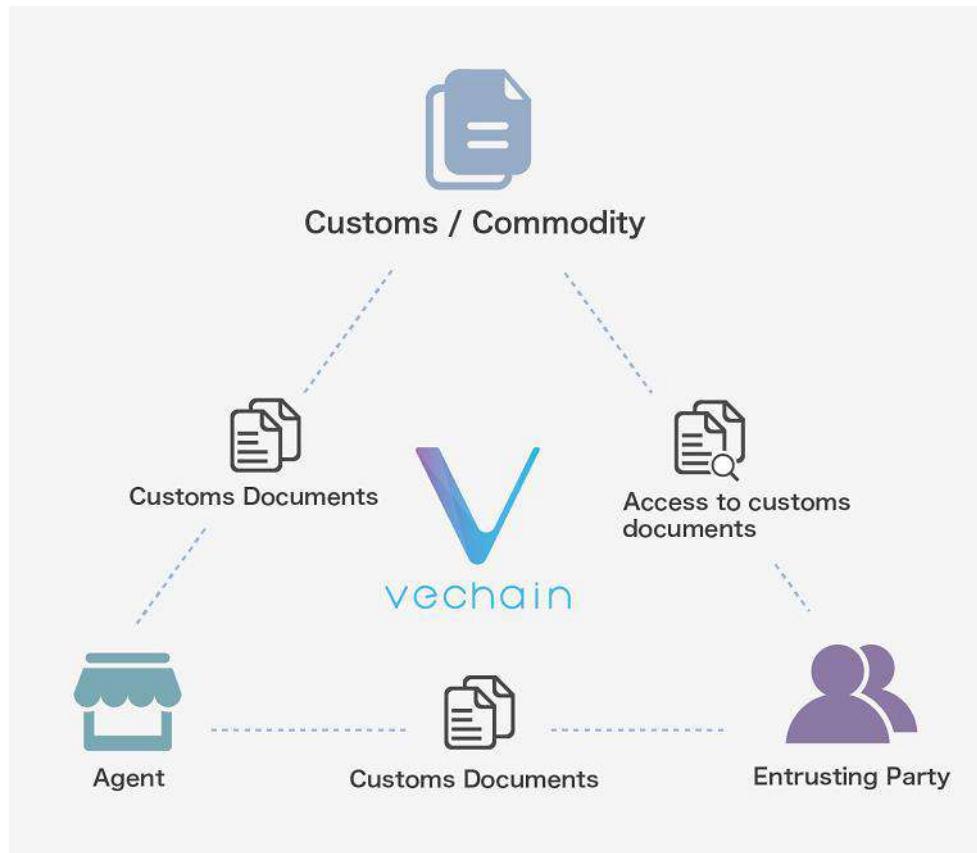


Figure-3.6-2 Blockchain Government Application

The Blockchain technology has shown significant meaning to the government. It represents the quality of government of open, public and transparent about the information. At the same time, as the “coordinator” of the whole commercial environment, government focus more on how Blockchain can improve the efficiency of the resource distribution in different industries.

3.7 This is just the Beginning

In the past two years, team VeChain has faced many big challenges. It's not the technical consensus, but the consensus of how to achieve the business model and change from the traditional business. However, we have been through the worst part already, and we thank so much for the cooperation partners and customers who dared to step into this field with us, so we can develop and verify the practical use of the Blockchain.

VeChain ICO is not just a beginning, because we have the experience from the

past two years. VeChain ICO is also a beginning because in front us, we still have a long way to go in this business environment. The challenge is we still need to invent new business model and promote them in order to complete building the thoroughly connection within the team VeChain, enterprise participant and community.

4. Governance Structure and Management Philosophy

In the beginning of the discussion about ICO and the preparation for the later stage, VeChain team has many heated debates and a long night talk, but we always have a consistent point of view:

"ICO is the beginning of everything, not the end of everything. All the so-called 'successful' ICO, in fact, is only a good starting power. And when the ICO stops, then it's really the time for the beginning of everything. The main theme is always about how to speed up, how to run on the right track, and how to avoid 'dying' during the process. Meanwhile, the team cannot just celebrate because of the ICO initial funding, but this should be regarded as the team bearing the hope of the community/business world and proceeding with great cautions. "

Therefore, maintaining the sustainable development of a team is also a proposition that the VeChain team has been discussing and thinking.

The corporate governance structure from the company system, is used to constrain the enterprise strategy, risk management, operation principle, human resources and legal compliance program.

Although the Blockchain technology utilizes decentralized concept as the starting point and establishes an efficient collaborative community platform, in order to improve the efficiency of collaborative Blockchain community and its operation, the team can still learn from the experience of corporate governance structure. The VeChain is also a framework of "non-traditional" community. In addition to the individual participants, there are more business users from different enterprise would agree to a reasonable corporate governance structure.

Of course, the structure concept cannot be applied mechanically. It is necessary to seek a dynamic balance between community culture and traditional enterprise management culture. This treatment method is combined based on our experience in the Blockchain industry during the past few years with the constant adjusting and optimizing in the future development.

4.1 The Establishment of VeChain Foundation

The VeChain Foundation (will refer as the Foundation hereinafter) is a non - profit entity established in Singapore in July 2017. The foundation will act as VeChain sponsor entity, who committed to support VeChain's development, construction and governance, transparency, advocacy and promotion work, as well as promoting the safety and harmonious development of the community.

The standard Blockchain community aims at a high degree of autonomy or decentralized as goal, allowing community participants to diversify their decision-making advice and usually use "vote" on important matters. However, such behavior is inefficient or unresolved because of the diversity of participants' opinions, which is not conducive to the continuous iteration and evolution of the Blockchain technology.

Moreover, because of the bifurcation behavior of the Blockchain, it causes serious divergence of opinion. The solution of "hard forking" has made people question the idea of "de-centering" of Ethereum" and even "Blockchain". This way of governance is not so much a "democracy" but an "anarchy."

VeChain development team highly recognized Blockchain's "decentralized" as the construction of the essence, while absorbing the essence of the traditional corporate governance structure, and improving the efficient formulation and implementation of the VeChain development strategy. At the same time it also prevent the serious Blockchain design philosophy differences and irreconcilable issue from showing up again.

The VeChain team commissioned a trusted third party organization to assist the team in setting up foundation entities in Singapore and to maintain the day-to-day operations as well as reporting the entity architecture. After establishing the Foundation, it selects the appropriate members of the community to join the functional Committee of the Foundation to participate in the actual management and decision-making.

4.2. Governance Principle

The design objective of the VeChain Foundation governance structure mainly focused on the **sustainability** of VeChain platform, the effectiveness of the strategy formulation, the management effectiveness, the risk control and the efficient operation of the platform economy.

1) The combination of centralized governance and distributed architecture

Although there are arguments advocating that Blockchain is a de-centralized or distributed self-governance community system. We believe that the absolute de-centralized may bring the absolute "fairness" but more likely to be further "inefficient". Therefore, the core idea of the Foundation is to absorb the concept of in the management structure of central governance, including the highest decision-making authority strategic Committee and major issues of the centralized procedure to improve the efficiency of the whole operation of the community.

2) The function of Committee and functional units coexist

The function of Committee and functional units coexist in the Foundation for daily affairs, which will set up permanent functional units, such as R & D department, marketing department, operation department, financial and human resources departments to handle daily affairs.

At the same time, a functional Committee is set up to make decisions on the important functions of the Foundation. Unlike the functional units, functional Committees exist in a virtual architecture where members of the Committee can be from any place of the world and do not have to work full-time. However, it must meet the requirements of the Committee's expert qualifications and be able to undertake to present and make comments when the Committee is required to present. The functional Committee will also set up a regular meeting system to ensure the effective promotion of major decision-making matters.

3) Risk oriented governance principles

The risk management will be the most important element in the process of studying the strategic development of VeChain Foundation. As a computer technology with great on-going revolutionary, the development of Blockchain is still in its infancy, so it is very important to grasp its development trend. The principle of risk management, when making sure the Foundation makes important decisions, takes full account of the risk factors, the possibility and influence of its occurrence, and makes corresponding countermeasures through decision-making. Thus, the development and iteration of the Blockchain is on the right path.

4) Technology and Commerce Coexisting

Any technology that is divorced from commercial applications is often difficult to develop. If the technology lack the practical function, it will stop and even come to the end.

From the creation of the VeChain, it always adhere to the close integration with the business as its purpose. The VeChain Foundation also follows this objective. Even if the Foundation will present as a non-profit organization, but the Foundation wants to get the maximum possible recognition of the business world. To get the business application of revenue, feedback to the Foundation

and the community, while further promoting the Foundation as well as the VeChain's development and upgrade. The VeChain Foundation also takes full account of this principle in the selection of talent and its architecture. It focuses on attracting experts with technical expertise, including industry experts who have a deep understanding of business.

5). Transparency and supervision

Referring to the governance experience of the traditional commercial world, VeChain Foundation also intends to set up special monitoring and reporting channels (Whistle-Blower). Designated by the strategic decision Committee as a window and we welcome the community participants to join in the management, supervision and the operation of Whistle-Blower channel. Those include, but are not limited to, new breakthroughs or recommendations that have significant implications for the Foundation or the Blockchain technologies, community operations issues, crisis information and fraud reporting or fraud, etc.

The Foundation will publish a unified information collection window, while ensuring the privacy of the information collected. At the same time, the Foundation publish periodic reports and irregular news releases in the form of community participation in all parties to disclose the Foundation operation and development of VeChain. Meanwhile, the main contacts of the Foundation management will be fully open, and accept the supervision and liaison of the participants.

4.3 VeChain Governance Model

The organizational structure of VeChain Foundation raised a combination of Specialized Committee and functional departments, which will deal with daily work and special matters. This section will discuss the functions of the functional Committees of the Foundation as well as the functions of the major functional departments.

With the reference to the operation of traditional entities, the Foundation will set up functional Committees, including the strategic decision-making Committee, the technical audit Committee, the remuneration and Nomination Committee and the public relations Committee.

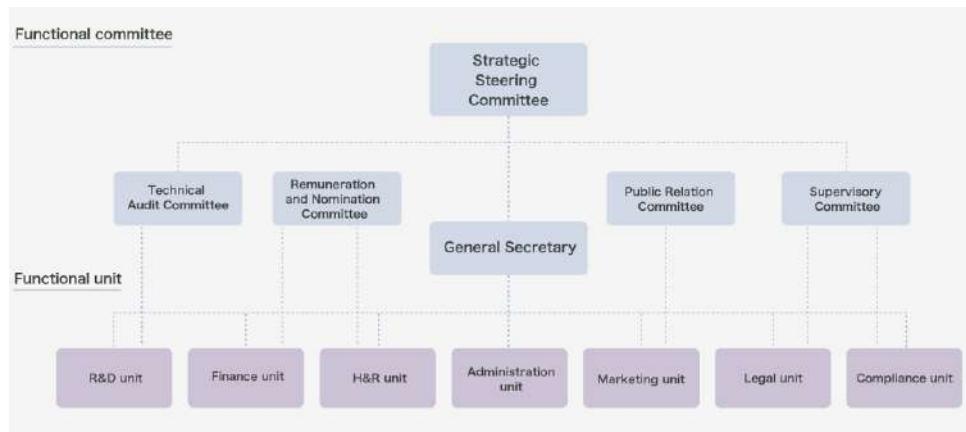


Figure 4.3 VeChain functional Committee structure graphic

4.3.1 Strategic Steering Committee

The strategic decision-making Committee is the highest decision-making Committee. The main objective is to negotiate and solve VeChain's decision making matters faced in community development include, but are not limited to:

- 1) Modifying governance structure of the Foundation.
- 2) The formation and rotation resolution of the policy-making Committee.
- 3) The appointment and rotation resolution of the Secretary General of the Foundation.
- 4) The appointment and dismiss of the chief executive and the head of each functional Committee.
- 5) Foundation review and amendment of the constitution.
- 6) The strategic decision of VeChain development.
- 7) VeChain's core technology changes and upgrades.
- 8) Emergency decisions and crisis management agendas.

The members of the strategic decision Committee and the president of the Foundation will serve for two years and the chairman of the fund shall not be reappointed for more than two sessions.

After the expiration of the decision making Committee term, VeChain will vote 50 community representatives by the consensus mechanism of next generation and then vote for the 7 core personnel of the decision-making Committee. Those elected representatives of the core staff will do emergency decision making, and accept the wage and salary investigation during his tenure, and the public their salary status as well.

These important matters need to be decided by the decision Committee with an open vote. Each member of the policy-making Committee has one vote, and the chairman of the Foundation has two votes. Decisions made by the decision Committee must be approved by more than half of all members of the Committee.

In addition, the person in charge shall convene the decision Committee to hold an interim meeting within 5 working days at the time of the following circumstances:

- ✓ The General Secretary of the Foundation considers when it is necessary.
- ✓ More than 1/3 of the decision Committee members jointly proposed.

The decision Committee meeting shall be attended by the members of the Committee. If members are unable to attend, they may entrust the other members of the Committee in writing. Failing to delegate is deemed to have given up the right to vote at the meeting.

4.3.2 General Secretary

The general secretary is the highest responsible person of VeChain administration. The responsibility is to make guidance and coordinate the daily operation of Foundation, technology development, community maintenance and public relations, as well as connecting various business unit with the governance structure of the functional Committee. The Secretary General will regularly report to the policy-making Committee.

4.3.3 Technical Audit Committee

The audit Committee comprises the core VeChain technology developer, who is responsible for the technology research, development direction of Blockchain, the underlying technology development, open port development and review, technology development and patent examination.

In addition, members of the technical review Committee regularly learn the dynamics and hotspots of the community and industry, communicate with participants in the community, and hold technical seminars on a regular basis.

4.3.4 Remuneration and Nomination Committee

The remuneration and nomination Committee is responsible for determining the selection and appointment of key managers of the Foundation. The Committee shall establish rules of procedure, assess the competence of the management, and authorize the appointment. At the same time, the Committee sets up a compensation system to encourage people who have important contributions to the Foundation.

The remuneration and Nomination Committee regularly reviews the performance of all the Foundation staff, advice on the human resource structure and raise

different incentive measures to attract talented experts.

4.3.5 Public Relation Committee

The public relations Committee is responsible for technically promoting the VeChain within the Committee, business alliance, the establishment and maintenance of the VeChain involved in each alliance party, and publicity regarding community crisis and other social responsibility.

4.3.6 Supervisory Committee

As a highly independent and autonomous form, the supervisory Committee is set up inside the Foundation as an independent risk control for the overall operation of the Foundation. The supervisory Committee conducts day-to-day guidance of the Foundation's legal and compliance departments. At the same time, the Foundation will set up a mechanism for reporting transparency and supervision to receive internal and external reporting issues, take corresponding improvement investigation and treatment, ensure that the Foundation operation is the perfect legal compliance, and continue to move forward in the acceptable level of risk.

The commission reports directly to the Committee on strategic decisions and does not have any conflicts or overlaps with other functions of the Foundation.

4.3.7 Other Functional Department

The Foundation refers to enterprise system framework and sets up day-to-day operations such as human resources, administration, finance, marketing, research and development (or laboratory) units, etc.

The functional departments maintain the normal operation of the VeChain Foundation, and directly deal with the relevant parties in the commercial society, such as enterprise customers, suppliers, regulators and the three party service organizations.

4.4 VeChain Human Resource Management

VeChain is committed to creating the world's most influential open source community ecology. To ensure the smooth development of the technology and the continuity of the Foundation operation, the Foundation will focus on recruiting excellent technology developers and managers with deep understandings of the business.

Talent Recruitment

Based on the characteristics of "Blockchain without borders", the Foundation

welcome talented people from all over the world to join the Foundation. In addition to the individual posts that must be recruited locally (e.g., logistics managers), recruitment is not limited to the place of work or the form of work. VeChain Foundation will, at the same time, follow the best practices in human resource management, develop appropriate human resources plans, recruitment procedures and review procedures to ensure that Foundations attract the right people.

As an open source community, VeChain will not only recruit full-time developers, but also employ well-known industry technical adviser. Relevant hiring and salary payment is required for discussion and decision, and signed the terms of cooperation by remuneration and Nomination Committee.

Performance Appraisal

VeChain will do the performance appraisal based on commercial company's best practice that comprehensively consider technology development, business expansion effect, economic operation, fund risk control management etc. The performance appraisal award will be submitted to the remuneration and Nomination Committee and the Strategic Decision Committee for review, and an optimization plan shall be worked out.

4.5 Risk Assessment and Decision Making Mechanism of VeChain Foundation

As an innovative technology, Blockchain is not only a disruptive breakthrough in computer core technology, but also a challenge to the traditional commercial society. Therefore, the importance of risk management system is self-evident. The VeChain Foundation is committed to build a risk oriented sustainable chain of block communities. It will continue to operate risk management of the foundation which includes the establishment of risk system, risk assessment, risk response and a series of activities.

For major risks, the strategic decision Committee will discuss and make decisions. It will classify risks based on event characteristics, such as event impact, extent of impact, probability of tokens and probability of occurrence, and decisions based on priority. For priority events, the relevant Committees of the foundation shall be organized as soon as possible.

4.6 VeChain Foundation Economy

In its economic operations, the Foundation promotes the following major principles:

- 1) Take the nonprofit as the main principle, and give back to the community;
- 2) Sustainable development
- 3) Collaboration and sharing of resources

Financially, the Foundation will seek the financial balance between expanding and community development. In addition to the initial funding received during the ICO, the Foundation will be able to obtain digital asset income through community eco operations. Under the arrangement of the third party trust institution, it will be transparent to distribute all the benefits to all operations and community development.

The Foundation will set up a full-time financial management team to maintain its financial and digital assets. The financial management team reports directly to the strategic decision Committee, and regularly prepare the financial reports and disclosures of the Foundation.

4.6.1 Funding Sources

The main income of the Foundation can be divided into two areas:

- 1) Non-operating income comprising the initial ICO's funds and the return on digital assets.
- 2) Regular operating income, including R & D, product sales, patent transfer or licensing, academic exchange and contribution, etc.

The following is a detailed description of the main sources of income:

- a. ICO initial startup funds.

VeChain tokens totaling 1 billion VeChain Token(VET). The allocation plan is as follows:

Ratio	Distribution Plan	Details
41%	VeChain Token(VET) crowdsale	The income of VeChain Token(VET) crowdsale will be used to VeChain Foundation operation, including development, marketing, finance and legal advisory.
9%	Private investor	Private investors are very influential in the community, and they will help a lot in technology and business development.
23%	Enterprise investors	Enterprise investor refers to an enterprise in VeChain distributed business ecosystem or a service provider for these corporate customers or end users; these enterprise investors will use the future VeChain Token(VET) as a key development target in their business activities.
5%	Cofounder, development team	To be distributed to the founders and development team of the VeChain Token(VET) as their rewards.

Ratio	Distribution Plan	Details
12%	Continuous operation and technological development	To be reserved for various operating costs and development of the VeChain.
10%	Business development case	To Choose the suitable industry, using VeChain technology to the strategic deployment of the industry, project support and tokens replacement.

- b. Digital asset investment. During continuous operation, the Foundation will allocate about 5% to 10% of the funds or digital assets to invest in the Blockchain industry, such as start-ups and incubators, angel investment in emerging scientific and technological investment.
- c. In the process of building the ecosystem, VeChain will serve as an underlying architectures provider of VeChain and receive a certain amount of digital assets or funds. For example, community participants, enterprises and other VeChain Token(VET) purchasers for GAS, as well as the Foundation. It will provide technical sharing and licensing gains. For this part of the proceeds, the Foundation will continue to invest in the community, form a community constantly expanding, and increase the positive cycle of influence.

4.6.2 Fund Budgeting

As mentioned above, the Foundation's funds spending mainly includes day-to-day operations, technology development, business development and reinvestment. The main categories are shown in the following table:

Classification	Percentage	Content
Technology Development	50%	It mainly includes reward for initial team, recruitment of experts and developers, technical patent and protection of intellectual property rights
Business Development	35%	VeChain business development and training, technical exchange and sharing, periodical publication, alliance establishment or participation, etc.
Reinvestment	10%	Blockchain, new technology and new team investment or absorption
Daily Operation	5%	Foundation daily logistics management, transportation and office, financial and reporting needs, etc.

See the foundation's initial forecast for the next four years of its operations:

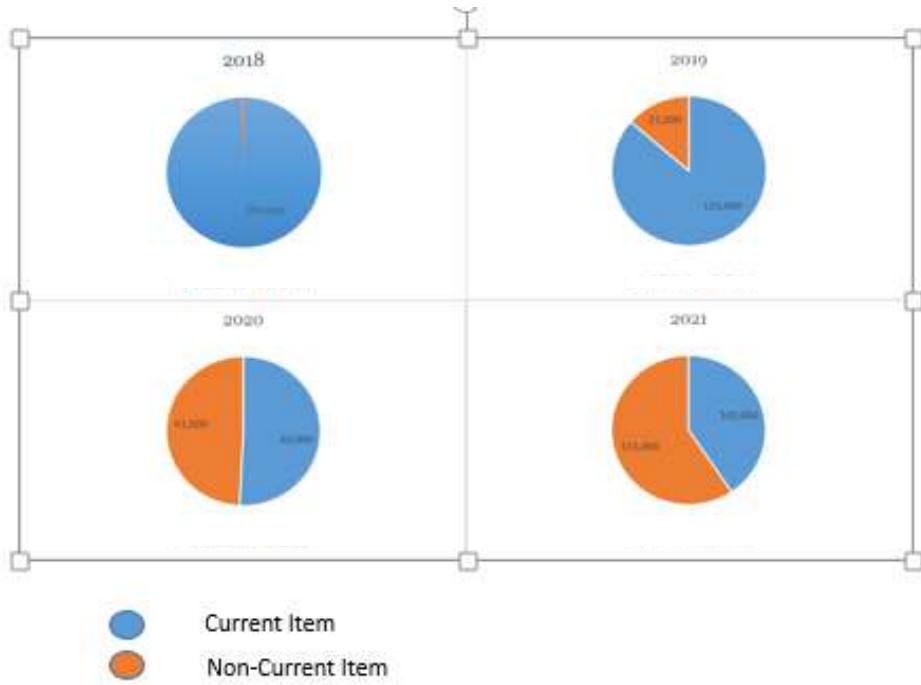


Figure 4.6.2-1 VeChain Foundation's 4 annual revenue forecast (000 RMB)

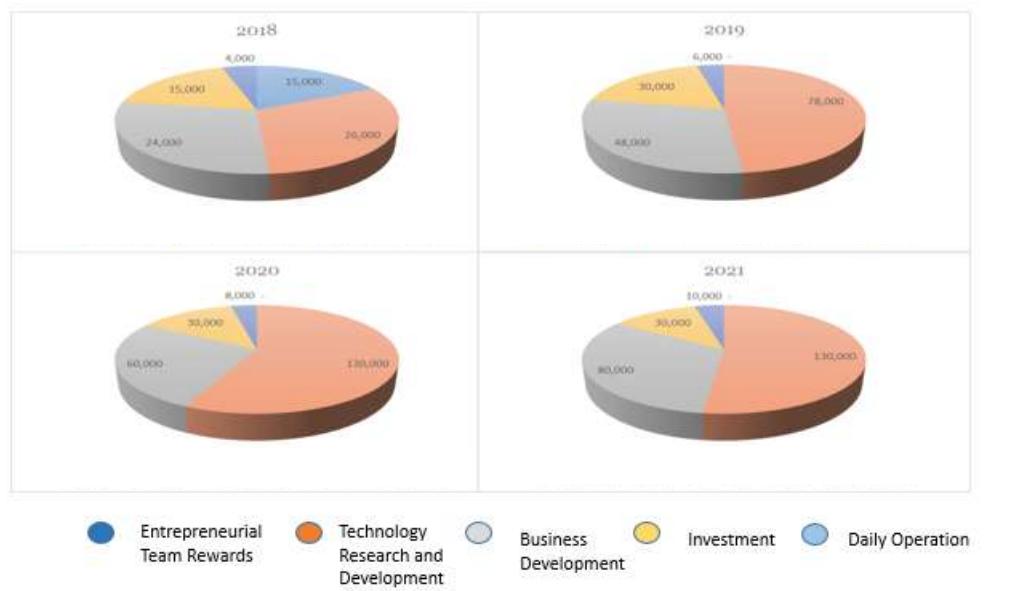


Figure 4.6.2-1 VeChain Foundation 4 year cost forecast (000 RMB)

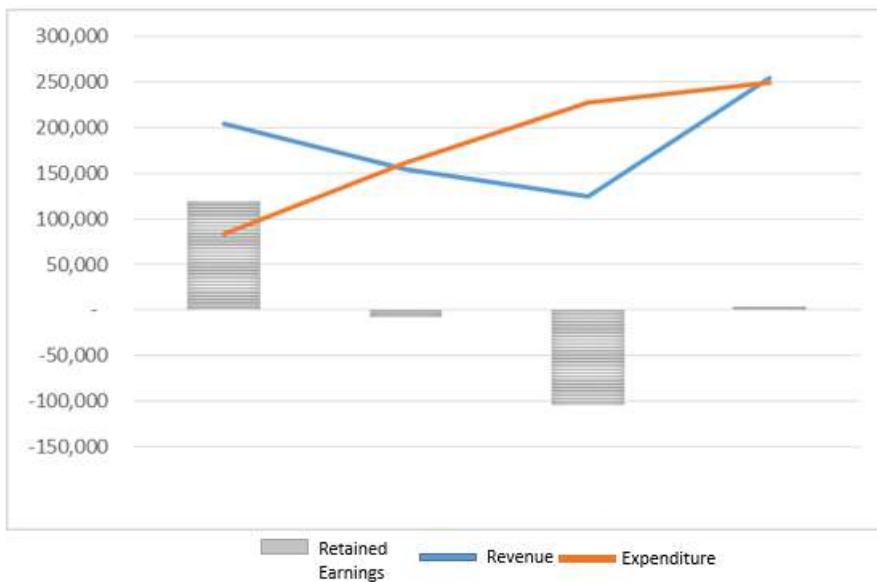


Figure 4.6.2-1 the VeChain Foundation's 4 year retained earnings forecast (000 RMB)

To sum up, the VeChain Foundation is expected to obtain start-up funds through ICO activities, which took about 3-4 years to achieve:

- 1) **Foundation scale and influence continues to grow.** This includes headcounts increasing to around 100. The Foundation attracted the business world continuously joining with than 150 billion yuan of goods in the VeChain flow.
- 2) **Foundation completes the self-circulation.** The Foundation relies on initial ICO start-up funds to get commercial value from the community and feed back to the community. The Foundation guarantees that the gains will be balanced with the expenditure.
- 3) **Focusing on R & D and commercial promotion.** According to the Foundation and VeChain concept, Foundation has always attached importance to the Blockchain based research and development, and business promotion and expand the influence. Most of the annual expenditure will focus on these two aspects.
- 4) **Adhere to the nonprofit principle.** The Foundation promises not to distribute profits, nor does it call dividends". Foundation operating income, in addition to the basic expenditure of the foundation, will all be put into the expansion of the community, to promote the community growing.

4.6.3 Fund Use Restriction

The use of VeChain assets is in line with the principles of openness and transparency. According to the principle of distribution and budget, VeChain will set up a separate account and digital asset wallet address used by depository

institutions to digital assets supervision and regularly share to the community. The principle of the use of revenue from public sale:

- ✓ Exceeding the value of 1 million yuan (or equivalent digital assets) requires approval by the head of the financial unit and the Secretary General.
- ✓ Over 5 million yuan (or equivalent digital assets) will need to be approved by the policy-making Committee.

4.6.4 Financial Planning and Implementation Reports

Each quarter, the financial and personnel management Committee prepare the financial planning, and summarize the last quarter financial performance. The formation of financial reports will be submitted to the decision-making Committee for approval.

4.6.5 Digital Asset Management

The digital assets belonging to the Foundation are appointed by the strategic decision Committee, and the full-time financial personnel are responsible for the arrangement. Digital assets and transaction currency are arranged independently and timely financial accounting. Following the best practices of financial control, the Foundation adopts multiple signatures to ensure the safety and accuracy of the assets. All the collected coins will be the timely transfer to digital assets and digital wallet. Foundation assets are not deposited in individual accounts.

- **Digital wallet management**

Based on the principle of independence, VeChain Foundation's wallet adopts 4/7 multi signature. Added Signature is subject to the approval of the strategic decision Committee. Large tokens are cold saved, and small tokens use multiple signatures.

- **Disclosure matters**

Each year, the Foundation will inform the community of chain development, operations, business promotion and the Foundation's operations. For the financial situation of the Foundation, the financial statements will be performed quarterly, and the work of the annual audit will be disclosed as well.

The Foundation establishes a public relations Committee, which serves as an external window for regular and irregular meetings and releases important information to the public.

4.7 Legal Compliance Matters and Other Matters

- **Legal affairs**

VeChain team commissioned a trusted third party organization to set up a Foundation entity in Singapore. All operations are subject to local laws, regulations and regulatory requirements. If there is a need to seek legal advice, it needs to be confirmed by local counsel.

- **Exemption clause**

VeChain Foundation insists the nonprofit nature of the unit's operations. Whether or not to obtain only chain tokens, Users who participate in the only chain community, can hold token or give up token rights. Holding tokens simultaneously means the holder's own rights to consume and use smart contracts on the Blockchain platform. Buyers should understand that within the scope of the law, VeChain foundation does not make any express or implied warranties and benefits. In addition, buyers should understand that there is no refund or refund after purchasing only chain tokens.

- **Settlement of dispute clause**

When a dispute arises, the parties concerned shall settle it by consultation in accordance with the agreement. If the settlement cannot be solved by negotiation, it can be settled by law

5. Introduction of the Team and Team Member

Team VeChain is a pure internationalized team. The team member is from different industries and countries, but they follows the same dream. The composition of the team is well balanced. Business, technology, operation and support are all important.



Sunny Lu , Project Leader

Sunny was graduated from Shanghai Jiao Tong University, majored in Electronics and Communication Engineering. He has been served as IT Executive in Fortune 500 companies over 13 years, former CIO of LV China.

He started VeChain project in 2015, and committed to Blockchain technology and business implementation.



Richard Fu, PR & Marketing Director

Richard has over 20 years' working experience in multi-national enterprises such as Shangri-la Group and LVHM specializing in sales and marketing.

He joined VeChain as director of PR and marketing



Chin Qian, Channel & Sales Director

Chin worked for HP from 2004 to 2016 and accumulated rich experience in marketing and project management.

He joined VeChain in 2017 as director of business partner recruitment and management.



Jay Zhang, Finance Director

Jay has worked for PwC and Deloitte as senior manager over 14 years.

He joined VeChain in 2015 as the leader of Blockchain governance framework design and digital assets management framework establishment.



Scott Brsbin, General Counsel

Scott is a well-known lawyer from the United States. His clients include Rolling Stones and lead singer Mick Jagger, Disney, MGM and so on.

He graduated from the University of California, Los Angeles in 1978, he joined the MSK law firm, and since 1989 began as a legal partner. In the company's legal affairs and patent maintenance, he has an absolute authority on the experience.

Scott joined VeChain in 2016 and worked for VeChain's legal security, organizational structure and property

escort.



Jerome Grilleres, Business Development

Jerome holds an MBA from London Business School and a MSc in Computer Science. He is from Barclays France and has 8 years' experience in Business Strategy and Development in Retail Banking and 6 years in Developing Real Time trading application in Investment Banks. Jerome joined VeChain in 2017 as Business Director of Europe.



Jianliang Gu, Technical Director

Jianliang was graduated from Shanghai University with master degree majored in Cybernetics. He was working in TCL communication technology as Technical Director. He has more than 16 years' experience in both hardware and software of embedded system development and management.

He joined VeChain in 2017 and commit to marry IoT and Blockchain



Peter Zhou, R&D director & Scientist

Dr. Peter Zhou obtained his Ph.D degrees in Computer Science from the University of Southampton. He was involved in projects funded by the European Commission and Academy of Finland when working as a postdoctoral researcher at the University of Kent, UK as well as a senior research scientist at the University of Oulu, Finland.

He has had more than ten years of scientific research experiences and published papers in top-tier international journals and conferences.



Bin Qian, Blockchain Director

Bin has over 10 years' experience in Mobile application development industry, specializing in developing Internet applications based real-time communication system. He is definitely P2P network technology expert. He joined VeChain in 2016 and is in charge of the Blockchain development.



Tony Li, Application development manager

Tony's majored in Information Security. He has 5 years' experience developing software and project management. Took part in numerous projects, including financial industry, insurance industry, luxury industry, the automotive industry.

He's interested in Bitcoin and Blockchain technology since 2014, and has two years' experience developing in the Blockchain product development.



Sherry Li, Product Manager

Sherry was graduated from Jiangnan University majored in Information Security. She has over 4 years of experience in application development, project management and product planning, including SAAS service platform, O2O platform and user oriented application.

She joined VeChain in 2016 as Product Manager



Jack Wu, Project Manager

Jack was graduated from St.John's university (New York). He has over 3 years iOS development and project management experiences. Took part in numerous project, including luxury goods industry, government agencies, the automotive industry
He joined VeChain in 2016 as Blockchain Project Manager.



Harvey Shang, DA Facilitator

Harvey was graduated from University of Florida majored in Computer Science. His study focus in distributed systems and advanced data structure.
He joined VeChain in 2016 and continued working in researching digital assets management area.



Lingbo Li, Risk Controller

Lingbo was graduated from Chinese Academy of Sciences with master degree in Finance Engineering. She has over 11 years' experience in credit risk management and assets management.
She joined VeChain in 2016 and is responsible for digital assets management and related risk control.



Cissy Chen, HR&Admin Manager

Cissy has over 6 years' experience in human resource management and worked for sub brands of Unilever before join VeChain in 2015.
She is in charge of human resource management, recruitment, staff training, compensation and other related strategies and policies establishment.

Collection References

- <https://bitcoin.org/bitcoin.pdf>
- <https://gavwood.com/paper.pdf>
- https://ripple.com/files/ripple_consensus_whitepaper.pdf
- https://eos.io/documents/EOS_An_Introduction.pdf
- <https://cryptoverze-0m5mfism.netdna-ssl.com/wp-content/uploads/2018/11/Downloadable-Tether-White-Paper-PDF.pdf>
- https://tron.network/static/doc/white_paper_v_2.0.pdf
- <https://www.stellar.org/papers/stellar-consensus-protocol.pdf>
- <https://iohk.io/research/papers/#ouroboros-a-provably-secure-proof-of-stake-blockchain-protocol>
- <https://whitepaperdatabase.com/monero-xmr-whitepaper/>
- <https://whitepaperdatabase.com/iota-miota-whitepaper/>
- <https://docs.dash.org/en/stable/introduction/about.html>
- <https://whitepaperdatabase.com/neo-whitepaper/>
- <https://makerdao.com/en/whitepaper/>
- https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf
- https://cdn.vechain.com/vechain_ico_ideas_of_development_en.pdf