Peer-reviewed version available at *Information* **2020**, *11*, 90; doi:10.3390/info11020090

Article

k-root-*n*: An Efficient Algorithm for Avoiding Short Term Double-Spending Alongside Distributed Ledger Technologies Such as Blockchain

Zvi Schreiber

Individual, London, UK; zvi@zvi.net

Abstract Blockchains such as the bitcoin blockchain depend on reaching a global consensus on the distributed ledger; therefore, they suffer from well know scalability problems. This paper proposes an algorithm that avoids double-spending in the short term with just $O(\sqrt{n})$ messages; each node receiving money off-chain performs the due diligence of consulting $k\sqrt{n}$ random nodes to check if any of them is aware of double-spending. Two nodes receiving double-spent money will in this way consult at least one common node with very high probability, due to the 'birthday paradox', and any common honest node consulted will detect the fraud. Since the velocity of money in the real world has coins circulating through at most a few wallets per day, the size of the due diligence communication is small in the short term. This `*k*-root-*n*' algorithm is suitable for an environment with synchronous or asynchronous (but with fairly low latency) communication and with Byzantine faults. The presented *k*-root-*n* algorithm should be practical to avoid double-spending with arbitrarily high probability, while feasibly coping with the throughput of all world commerce. It is resistant to Sybil attacks even beyond 50% of nodes. In the long term, the *k*-root-*n* algorithm is less efficient. Therefore, it should preferably be used as a complement and not a replacement to a global distributed ledger technology.

Keywords: blockchain; bitcoin; cryptocurrency; distributed ledger technology

---

1. Introduction

In blockchains such as bitcoin, all n nodes reach Nakamoto consensus [1] on each block of transactions, thereby creating a scalability problem [2] [3] [4] that notoriously limits the entire bitcoin network to a few transactions per second while consuming massive power [5]. Bitcoin is considered the first digital currency algorithm to solve the double-spending problem without the need for a trusted authority or central server. Additionally, it can cope with Byzantine faults (e.g. [6] [7]) including a Sybil attack [8] of up to 50% dishonest nodes. However, it requires $O(n)$ communication messages per transaction that limit its scale.

Practically, bitcoin transactions suffer from a lag time of 15 minutes to several hours before being included in a block on the bitcoin blockchain [9] (this lag time has a complex dependency on how high a fee is offered by the transaction participants to the miner [10]), and then an hour longer to reach the generally desired threshold of 6-block confirmation [11]. Thus, before received bitcoin transfers are confirmed and are safe to re-spend, there is typically a several hours latency.

At the time of writing, the typical fee paid to the miner for a single bitcoin transaction is tens of thousands of Satoshi or about US$0.50–$5 [9]; this is not cheaper than most domestic bank transactions with fiat currency.

Therefore, efforts are being made to redesign the blockchain algorithm itself to ensure greater scalability, such as SCP [12], Algorand [13], bitcoin next generation (bitcoin-NG) [14], which all

involve selecting a subset of users (committees or a rotating leader) in various configurations to reduce the number of messages required to reach consensus. In an alternative approach, a subset of nodes transacts with each other off-chain for a time, [15] as in the lightning network [16].

In this paper we consider an approach for protecting against double-spending [17] possibly combined with a Sybil attack without the need for a global ledger consensus. In this paper, we did not consider other forms of attacks, such as eclipse attacks [18], routing attacks [19], attacks based on time advantage [20], incentive attacks [21] and quantum computing attacks [22]. Relevant surveys of other attack types are [23] [24]. Some previously research on avoiding double-spending are [25] [26]. This study focuses on the protection against the most central vulnerability of cryptocurrencies, namely double-spending covered up by a Sybil attack of malicious nodes.

We propose a scalable low-latency algorithm that can run off-chain in parallel to a global ledger consensus mechanism, such as blockchain, protecting against double-spending in the short term. Thus, commerce may continue at a high pace even while the n nodes are working to reach consensus on transactions possibly with a lag of some hours from the transaction time. By applying this algorithm, we accept a situation where consensus is infrequently achieved. Therefore, we can accept longer blockchain blocks that are created every hour, or every few hours, instead of bitcoin's current average of 10 minutes, thereby increasing blockchain's throughput of transactions per second [27], while compensating for longer latency with our complementary off-chain algorithm for preventing short-term double-spending.

For example, each morning the nodes may reach a consensus on the valid transaction histories and wallet balances as of the preceding midnight Greenwich Mean Time, and they may asynchronously do so, reaching the consensus by, say, 6 a.m. the next morning. For example, in the particular case of bitcoin, all the transactions by 6 a.m. from the previous day would typically have achieved six-block verification and may be considered final. In this case, the role of our algorithm is to allow fast and safe transactions in the 30 hours say from Sunday midnight to Tuesday 6 am when consensus is finalized for the ledger as of Monday midnight. Thus, in this example, to reach a consensus only every 24 hours with a 6-hour lag, we allow for a situation where the distributed ledger is relaxed relative to the current configuration of bitcoin. Therefore, this enables the transaction rate of the ledger to increase. Furthermore, our proposed solution can allow people to trade, particularly to safely pass on the received coins with next to zero latency.

The proposed algorithm, which is called '$k\sqrt{n}$' or 'k-root-n', avoids double-spending in the short to medium term, while there is no global ledger consensus with an arbitrarily high probability of detecting double-spending requiring just $O(\sqrt{n})$ messages per transaction. This is based on the assumption that specific money balances only circulate through O(constant) wallets in 24 hours. This assumption is realistic since money circulates in the real economy with a velocity measured in one or two transactions per month [28], and bitcoin is already practically constrained by the transaction confirmation lag times to circulate a few times per day, and in practice rarely more than once or twice a day.

In this algorithm, every transaction should eventually be on-chain. The initial transaction verification is off-chain, thereby allowing transactions to continue off-chain at high speed and waiting for the blockchain to catch up. The algorithm only involves $O(\sqrt{n})$ nodes and messages per transaction; we typically choose $10\sqrt{n}$.

2. Overview of $k\sqrt{n}$ random double-spending detection

Suppose there are n nodes, in which each node is also a wallet, connected to a network, and they achieved consensus on the global distributed ledger (or at least on the balance of each node) using blockchain (or another algorithm) sometime recently, a time we shall call the global ledger consensus (in the example above that would occur at 6 a.m. daily).

For now, we assume that every wallet is also a node, which is usually online and available, and which also provides basic verification services to the network. The central idea is that any honest node that wants to verify whether the funds it receives have not been double-spent can demand that the sender disclose the pedigree of the transferred funds, namely the sender's transaction history since the last global ledger consensus. In case the sender depends on incoming funds to have sufficient balance to cover the transaction, the receiver can recursively demand disclosure of the source of funds right back to funds that were available as of the last global ledger consensus.

Since querying all the nodes to verify each transaction is prohibitively expensive, an honest node will perform its due diligence on the pedigree of each inbound transaction with a random $k\sqrt{n}$ other nodes. If two nodes query $k\sqrt{n}$ random nodes, we will show that the probability of zero common nodes is extremely small for suitable k even if a substantial proportion of nodes are failing or malicious. Therefore, if two honest nodes receive the same double-spent coins, they may consult at least one common node and detect the fraud with a very high probability. This is the main idea of the algorithm, and it is based on the famous birthday paradox [29], where for example just 40 people ($\approx 2\sqrt{365}$) have about a 90% chance that at least two of them have the same birthday.

Each honest node provides verification services by keeping a history of the transaction pedigrees it was asked to verify; when two honest nodes query random nodes, any common queried honest node can immediately raise the alarm if the two receiving nodes are victims of a double-spending attempt, i.e. if they were given inconsistent transaction histories.

Let k be a small number greater than 1. We will generally choose $k = 10$. Assume that we are in an environment with Byzantine faults, say about 10% of nodes may not respond at any time due to node or network failure; assume that about 50% of the nodes are malicious, we would then have an effective $\underline{k} = 4.5$, i.e. $\underline{k}\sqrt{n}$ responsive and honest nodes. When each of two honest nodes receives funds and successfully each query $4.5\sqrt{n}$ honest nodes, this $\underline{k} = 4.5$ is sufficient to ensure an expected value of more than twenty common, honest and responsive nodes. There is a probability of just approximately 10^{-9} of zero common, honest and responsive nodes. Therefore, the chances of getting away with double-spending are negligible, and there is a probability very close to 1 that any double-spending can be detected as soon as both branches of the spend reach honest nodes.

The penalty for double-spending is forfeiting the wallet, so if each wallet has a minimum stake m of \$1 and each transaction is limited to well under \$1 billion, say to a maximum M = \$1,000,000, then there is a negative expected return from any double-spending attempt since there is a probability of just approximately 10^{-9} of not being caught.

A dishonest node may not be checking its inbound transactions or may be maliciously collaborating with other nodes. This is why the receiving honest node should check not only the transaction history of the immediate sender for forked history/double-spending, but also to recursively check any of sender's sender's transactions, to the extent that the immediate sender depends on the sender's sender (recursively) payment to have balance for covering the current transaction. This recursive tree of inbound transactions is called the pedigree of the transaction, that

is the recursive list of transactions that it depends on. This recursion is why the *k*-root-*n* algorithm is less efficient for long term use since the recursive pedigree of transactions may become large over a long time period.

As a motivation for the $O(\sqrt{n})$ algorithm, we briefly explore how a $10\sqrt{n}$ algorithm scales by assuming n = 10 billion people (the projected world population for 2050 [30] and much more than bitcoin's current 32*m* wallets [31]). Suppose people are each transacting once per hour on the average, i.e. 24-hours per day (higher than the average rate of commerce). Each transaction involves messages to $10\sqrt{n} = 10^6$ nodes. We will see that this gives a probability of just $p \approx 10^{-9}$ of getting away with double-spending even if half of the nodes are fraudulent and 10% of the nodes are unavailable (i.e. $4.5\sqrt{n}$ honest, responsive validating nodes). Thus, each transaction only burdens 1 out of 10,000 nodes, and with 10 billion transactions per hour globally, or 2.77million transactions per second globally, each node should be involved in just 278 transactions per second. This transaction throughput is feasible for a modern computer (especially in 2050).

Thus, it seems practical that the algorithm could securely handle not only Visa/Mastercard volumes, but in fact all the commerce in today's world and the foreseeable future. Visa's volumes have been widely misquoted in bitcoin articles as 24,000 per second, although that appears to be mythical [32] with apparently more reliable sources estimating about 78.95 billion Visa transactions in the first half of 2018 [33] which averages 5,000 per second, although peak time transaction rates would presumably be higher. In any event, the current algorithm could feasibly handle transaction volumes orders of magnitude larger than Visa.

Whilst the idea of depending on probability to secure commerce may at first seem strange, it is noteworthy that all commerce already depends on probability. For example, every credit card transaction is accepted based on a probabilistic evaluation that it isn't fraudulent.

We now introduce some definitions and formally present the *k*-root-*n* algorithm.

3. Preliminaries

Definition 1. A transaction $T = (x,t,S,R,s_S,s_r)$ is an agreement to transfer a balance from a sender node/wallet to a receiver node/wallet; the tuple comprises a positive amount $x = x[T]$, a time stamp $t = t[T]$, identifiers (public keys) of the sender user $S = S[T]$ and the receiver user $R = R[T]$. Additionally, s_S and s_r are the respective digital signatures of S and R of the data tuple (x,t,S,R).

A transaction T is only valid if the sender S had a balance (see next definition) of at least $m + x$ immediately before the transaction, where m denotes the agreed minimum wallet balance. No sender or receiver can participate in two transactions with identical time stamps t.

A potential transaction $\underline{T}$ is a transaction that has not yet been signed by the receiver $\underline{T} = (x,t,S,R,s_S)$. □

Definition 2. The balance $b[u,t_1]$ of a user u at time t_1, given that s/he had a balance of b_0 at the time t_0 of the last known global ledger consensus, is defined by

$$b_0 + \sum_{t_0<t[T_i]<t_1, R[T_i]=u} b[T_i] - \sum_{t_0<t[T_i]<t_1, S[T_i]=u} b[T_i],$$

i.e. the last known global ledger consensus balances plus all received amounts, minus all spent amounts. □

Definition 3. The Lineage $LIN[T]$ of a transaction or potential transaction T is the transaction history for the sender $u[T]$ from the last known global ledger consensus at time t_0 before $t[T]$ and up to the time of T :

$$LIN[T] = \{T_i \mid t_0 < t[T_i] < t[T],\ S[T_i] = S[T] \vee R[T_i] = S[T]\}.$$ □

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 January 2020 doi:10.20944/preprints202001.0059.v1

Peer-reviewed version available at Information 2020, 11, 90; doi:10.3390/info11020090

These are the transactions relevant to establishing that the sender u has sufficient balance to afford T. However, not all of this history is necessarily required to establish sufficient balance, so for the sake of efficiency we now introduce a narrower transaction history.

Definition 4. The Critical Lineage *CLIN[T]* of a transaction or potential transaction T is a set of transactions whose elements are a subset of *LIN[T]*, comprising a minimal subset of inbound transactions critical to provide the balance that allows the sender to afford T. Formally, suppose that a sender S makes a payment of amount x in a transaction or potential transaction T; suppose that S's last known global ledger consensus balance was b_0. Suppose that the set of transactions in which S participated since the last global ledger consensus, *LIN[T]*, includes inbound payments, $T_1 \ldots T_n$, in descending order of amount (and ordered chronologically when amounts are equal) and outbound payments, $U_1 \ldots U_m$. Now, the critical inbound payments are the subset $CLIN[T] = \{T_1 \ldots T_j\}$ of inbound payments with j minimal such that $b + \sum_i^j T_i - \sum_i^m U_i \geq x + m$. □

Thus, given that the sender has opening balance b and has spent the Ui, then, $\{T_1 \ldots T_k\}$ is a minimal subset of inbound transactions that are enough to provide balance coverage for this payment of x. Even if any of the other inbound payments, $T_{j+1} \ldots T_n$, is derived directly or indirectly from fraud, the validation by the receiver of these critical inbound payments *CLIN[T]* of the sender is sufficient due diligence to ensure that the sender can afford x. Therefore, the minimum due diligence of the receive $R[T]$ is to check the following: (A) *LIN[T]* is complete and (B) the lineage of each transaction in *CLIN[T]* is complete. We now formalize this recursive set of transactions.

Definition 5. The Pedigree *PED[T]* of a transaction or potential transaction T is the recursive closure of the set $\{T\}$ under the CLIN operator. To compute this:

- Start with the set of $\underline{PED}[T] = \{T\}$.
- For each T_1 in $\underline{PED}[T]$, add any element of $CLIN[T_1]$ which is not already in $\underline{PED}[T]$ to $\underline{PED}[T]$. Repeat until there is nothing to add.
- $PED[T] = \underline{PED}[T]$. □

It is also helpful to think of *PED[T]* as the nodes of a directed acyclic graph for each transaction, recursively showing the inbound transactions that the sender depended on to cover the transaction since the last known global ledger consensus. Figure 1 depicts a $10 transaction and its *PED* pedigree. Here, the opening balances are shown on the nodes, and we assume a minimum balance of $1. The $10 transaction depends on $2 that the sender already had (over and above the $1 minimum) plus two received amounts of $4 each, of which one, in turn, depended on a received $3. The dashed lines represent other received amounts that are not critical to covering the transaction balances so are excluded from the *PED*.

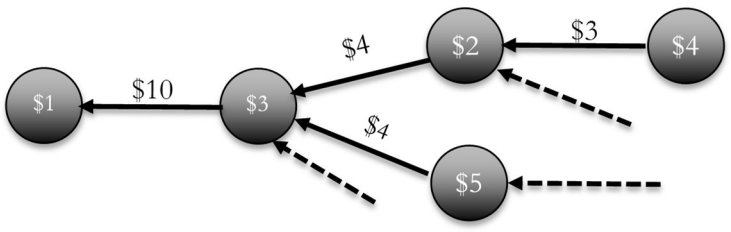

Figure 1. An example of the *PED[T]* pedigree of a $10 transaction

Definition 6. A Disclosure *DIS[T]* for a potential transaction $\underline{T}$ is a communication from sender *S*[$\underline{T}$] to receiver *R*[$\underline{T}$] of *PED[$\underline{T}$]* and *LIN[T_1]* for every T_1 in *PED[$\underline{T}$]*. □

Definition 7. A Fraudulent transaction *T* is any transaction wherein the sender *S*[*T*] provides the receiver with an incomplete *LIN[T]* in the disclosure. Additionally, a transaction is considered a fraudulent transaction in retrospect if, at some time between *t*[*T*] and the next global ledger consensus, the same sender *S*[*T*] is the sender of another transaction T_1 and fails to disclose *T* when disclosing *LIN[T_1]*. □

Thus, if Malory sends money to Alice and later double spends by sending the same money to Bob without disclosing to Bob the earlier payment made to Alice, then both payments are considered fraudulent. It is insufficient to cancel the second transaction; the one which was directly involved in the fraud like Alice may be a co-conspirator of Malory, while Bob is the only victim. The cancellation of both transactions ensures there is a significant penalty for fraud. In theory, this does mean that Bob would lose out since he was a victim of double-spending in retrospect, but practically this arrangement ensures that double-spending has a negative expected value and is very unlikely to occur at all.

Definition 8. An Invalid transaction is a transaction that is not fraudulent but wherein the sender in retrospect did not have balance to cover the transaction after removing fraudulent transactions. Equivalently, these are transactions that turn out to have a fraudulent transaction in their pedigree. □

Definition 9. Due diligence for a potential transaction $\underline{T}$ is the process of receiver *R*[$\underline{T}$] communicating the offered disclosure *DIS*[$\underline{T}$] with a random selection of $k\sqrt{n}$ (strictly $\lceil k\sqrt{n} \rceil$ i.e. $k\sqrt{n}$ rounded up to the nearest integer) nodes called the validating nodes and confirms that none of them has seen an alternative version of *LIN*[T_1] for any T_1 in *PED[$\underline{T}$]*. □

4. *k*-root-*n* algorithm

Suppose we have *n* nodes, each of which is also a wallet, connected to a network, and the nodes achieve consensus on the global ledger (or at least on the balance of each node) at some time t_0 in the recent past; we shall call this time the global ledger consensus. Each global ledger consensus may become known at time $t_1 > t_0$, that is with some latency after the time t_0 which it relates to (e.g. in bitcoin, we may wait some hours for transactions to be included in a block and then for 6-block confirmation before trusting that consensus was achieved). When we refer to the last global ledger consensus before time *t*, we mean the last one known before time $t > t_1$.

The nodes transfer balances to each other by mutually digitally signing transactions. Based on the algorithm, each receiving honest node will perform the following steps before accepting and signing a potential transaction $\underline{T}$.

- Demand that the sender transmits the disclosure *DIS*[$\underline{T}$] that includes the recursive list of dependent transactions *PED[$\underline{T}$]* and the transaction history *LIN[T_1]* for each T_1 in *PED[$\underline{T}$]*.
- Validate that each transaction T_1 in *PED[$\underline{T}$]* was properly formed and signed by known nodes, and each node had the balance to cover T_1 based on the last known global ledger consensus balance of *S*[T_1] plus the provided *LIN*[T_1].
- Due diligence: Randomly choose $\lceil k\sqrt{n} \rceil$ validating nodes on the network; send each (directly or by communicating through a cascading tree of nodes) *DIS*[$\underline{T}$] and ask the validating nodes to validate that they have not seen any alternative *LIN* history for any transaction in *PED[$\underline{T}$]*.

- Any honest node receiving this due diligence request will validate that they have never previously seen a contradictory *LIN* history for any of the transactions T_1 in *PED*[*T*]. Else, they will confirm their validation by digitally signing *DIS*[*T*] and transmitting that back to the receiver *R*[*T*]. Each validating node must then store every *LIN* transaction history they are asked to validate until the next achieved global ledger consensus, for future validation.
- If any validating node has in fact seen an alternative *LIN* transaction history for some T_1 in *PED*[*T*], it informs the receiver *R*[*T*] who rejects potential transaction *T*. Proof of fraud, which comprises two alternative histories *LIN*[T_1] and *LIN'*[T_1], should be broadcast to all nodes on the network by the validating node. The wallet *S*[T_1] will be blacklisted and any balance forfeited.
- The receiver *R*[*T*] should also broadcast the list of the other $k\sqrt{n}$ validating nodes that failed to raise an alarm. In case *R*[T_1] or any other nodes had previously disclosed *LIN'*[T_1] to one of these validating nodes V, then this node should also be blacklisted with proof of validation fraud, that is proof that V signed the current validation which included *LIN*[T_1] while previously validating a transaction disclosure that included the forked lineage *LIN'*[T_1]. Thus, the validating nodes are also held accountable.
- If on the other hand all the validating nodes validate the transaction, the transaction is accepted and signed by the receiver.

All nodes periodically take time to reach a global ledger consensus on the distributed ledger, e.g. using Nakamoto consensus. All recipients will request that the transactions they received should be added to the global ledger. In the case that a node was caught in a fraudulent transaction, it will be disqualified. All fraudulent transactions are iteratively removed from the ledger.

After removing fraudulent transactions, invalid transactions must be iteratively identified until they are excluded from the global ledger. This process must be iterative since invalidating one transaction may cause the receiver not to cover for subsequent spends, thereby invalidating further transactions.

In the *k*-root-*n* algorithm, there is a need for honest nodes to be online almost all the time. It is recommended to have a protocol wherein an honest node commits to a service level agreement (SLA) (e.g. [34] [35]) of say u = 90% uptime, and a node that does not comply may receive warnings and eventually financial penalties or disqualification by consensus of all the nodes. A node which tries to consult $k\sqrt{n}$ nodes and receives less than $uk\sqrt{n}$ responses in a specified target latency time should pick other nodes and retry until it receives the target $uk\sqrt{n}$ validations.

5. An example of detection of double-spending using the *k*-root-*n* algorithm

Suppose during Monday morning the network reaches a consensus that as of Sunday midnight the balances on the distributed ledger after all valid transactions were as follows:

- Chuck (malicious) $100
- Mallory (malicious) $100
- Alice (honest) $100
- Bob (honest) $100

The ledger also showed that there were a total of n valid nodes, in which each has at least a minimum stake of m = $1.

We analyze the scenario where Chuck conspires with Mallory to double-spend, by giving the same money to Alice and also to Bob. To conceal the double-spend, the payment to Bob is passed by Chuck via co-conspirator Mallory.

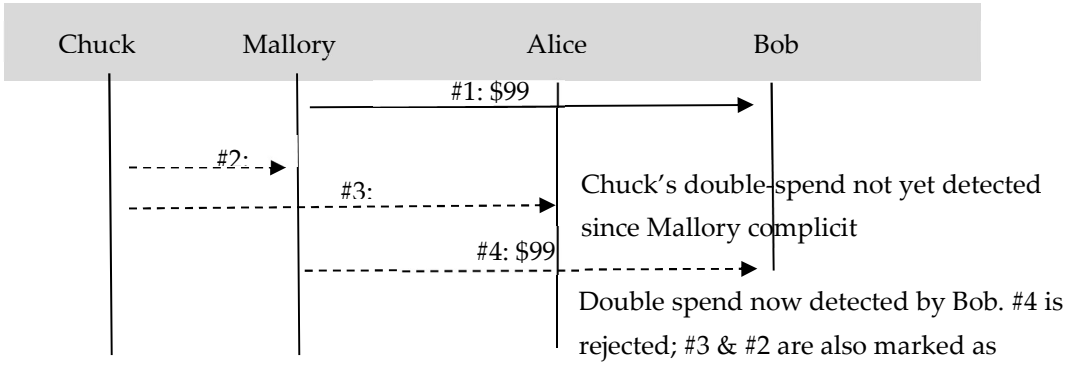

Figure 2. An example of transactions between two dishonest and two honest nodes, including attempted double-spending

- Mallory sends $99 to Bob (in exchange for some goods, services or other currency). Mallory discloses her transaction history from the last consensus, which is empty, so she has $99 to spend. Bob first confirms that Mallory had $100 as of the last known global ledger consensus. Then, Bob being honest does his due diligence and queries $k\sqrt{n}$ network nodes (either directly or through a cascading tree of nodes) to confirm that none of them has seen Mallory signing any other transactions since consensus. They have not, and Bob, therefore, accepts the $99 and counter-signs the transaction and submits it for eventual inclusion on the main distributed ledger.

- Chuck sends $99 to Mallory. Mallory being malicious and complicit with Chuck tells no one about this transaction. They both sign the transaction and may or may not submit it to the main ledger. Chuck sends this $99 through Mallory attempting to mask the double-spending he is planning. He might potentially pass this money through further nodes.

- Chuck now sends $99 to Alice in exchange for some value. This is a fraudulent double-spend. He informs Alice fraudulently that he has no other transactions since the last consensus. Alice being honest does her due diligence and queries $k\sqrt{n}$ random validating nodes with the declared disclosure. They all inform Alice that they are unaware of any forked transaction lineages (since transaction #2 was not broadcast) for Chuck, and so Alice accepts the payment. Thus, the double-spend is not yet detected (until both instances of double-spent money reach honest nodes).

- Mallory sends another $99 to Bob in exchange for some value. Bob being honest demands disclosure, and Mallory provides Bob with a copy of her transaction history since consensus, namely transaction #1 (-$99 that Bob already knows about) and transaction #2 (+$99), thereby evidencing Mallory's balance of $100 allowing Mallory to spend $99. At this juncture, Chuck's double-spent money has, via Mallory, reached the honest Bob.

    - Bob notices that Mallory has $100 but contingent on the money from Chuck (so transaction #2 is in the critical lineage *CLIN*[#4] and therefore in *PED*[#4]). Thus, Bob validates transaction #2 by requiring Mallory to provide Chuck's transaction history *LIN*[#2] as part of the pedigree. (Further, in case Chuck, in turn, was depending on the incoming transactions for his balance in transaction #2, which is not the case here, Bob would recursively require the sender's sender's sender's transaction history until he has a transaction history for every transaction since the last consensus which is sufficiently required to justify Mallory's balance to cover the current transaction).

- Chuck now does his due diligence and queries $k\sqrt{n}$ random network nodes by asking them to validate the pedigree, which includes both *LIN*[#4] and *LIN*[#2].

- Some of these nodes (k^2 on average, but at least 1 with an extremely high probability) had previously been told about Chuck's alternative transaction history of transaction #3 where he gave \$99 to Alice. Then, they raise the alarm of double-spending and broadcast a fraud-proof, namely that transaction #3 was not disclosed in *LIN*[#4]. The fraud-proof comprises two divergent transaction histories that were both signed by Chuck.

- Bob rejects the fraudulent transaction.

    - Chuck has his wallets blacklisted and forfeits his \$1 minimum stake.

    - The fraudulent transaction #2 from Chuck to Mallory (which was later hidden from Alice) is also rejected from the distributed ledger. Therefore, transaction #4 is invalid since it depends on a fraudulent transaction #2.

    - The network preferably should ask Mallory to show that she queried $k\sqrt{n}$ validating nodes. When she fails to do so, Mallory should also be blacklisted and forfeit her balance. (This is optional extra protection called no due diligence fraud although this is not required for the algorithm to work and cannot be enforced in case Mallory had $k\sqrt{n}$ co-conspirators and pretends to select them at random).

    - Alice and Bob compare notes and find all the $\sim k^2$ common validating nodes they had consulted in #3 and #4 and ensure none of them failed to report the double-spending. If any common node failed to raise the alarm, then such node would also be blacklisted for fraud, with the validation fraud-proof showing that the node received two alternative histories from Chuck and in both cases approved them (such approvals being signed by the verifying node comprises verification fraud-proof).

The next morning consensus is established again around the following end balances:

- Chuck (malicious) \$100 (blacklisted with balance forfeited for double-spending)
- Mallory (malicious) \$1 (may be blacklisted for failing to do due diligence on #2)
- Alice (honest) \$100
- Bob (honest) \$199

Once this new global ledger consensus is known, future senders only need to provide shorter transaction histories back to the newer global ledger consensus. In this scenario, Alice has clearly lost out as a victim of fraud, but the algorithm ensures that the fraudsters have significant losses with very high probability, meaning that such fraud is very unlikely in the first place.

6. Algorithm correctness

Theorem 1. *For any two honest nodes receiving and successfully validating payments with $k\sqrt{n}$ random nodes each, there are an average of k^2 common nodes queried by both honest nodes (any one of which can detect double-spending and raise the alarm).*

Proof of Theorem 1. The first honest node randomly queries $k\sqrt{n}$ nodes representing a proportion $k/\sqrt{n}$ of all n nodes. Therefore, when the second honest node queries $k\sqrt{n}$ random nodes, a proportion of $k\sqrt{n} * (k/\sqrt{n}) = k^2$ on average will overlap. □

This result is the key strength of the algorithm since both transactions involve O($\sqrt{n}$) validating nodes, and the expected value of the number of overlapping nodes is significant, thereby allowing any double-spending to be detected.

However, since it only requires one common node to detect fraud, what we are not really interested in the expected number of common validating nodes but rather in the probability of at least one node in common, versus that of zero common nodes, which we require to be very small.

Lemma 2. *The probability* $\boldsymbol{p_0(n,r)}$ *of zero clashes (zero common nodes) between two random sets of* $r = k\sqrt{n}$ *nodes satisfies* $p_0(n,r) < e^{-k^2}$ *with* $p_0(n,r) \approx e^{-k^2}$ *for large n and* $r \ll n$.

Proof of Lemma 2. First, $p_0(n,r) = \frac{\binom{n-r}{r}}{\binom{n}{r}}$ since there are $\binom{n}{r}$ ways for the second node to choose r validating nodes from n nodes in which $\binom{n-r}{r}$ combinations involve zero of the r validating nodes that the first node chose. Thus,

$$p_0(n,r) = \frac{\binom{n-r}{r}}{\binom{n}{r}} = \frac{(n-r)!r!(n-r)!}{r!(n-2r)!n!} = \frac{(n-r)\ldots(n-2r+1)}{n\ldots(n-r+1)} < \left(\frac{n-r}{n}\right)^r = \left(1-\frac{r}{n}\right)^r \quad \text{with} \approx \text{for } r \ll n.$$

Now, let $r = k\sqrt{n}$. Then, we can approximate

$$p_0(n,r) = \left(1-\frac{k\sqrt{n}}{n}\right)^{k\sqrt{n}} = \left(\left(1-\frac{k}{\sqrt{n}}\right)^{\sqrt{n}}\right)^k < (e^{-k})^k = e^{-k^2}$$

again with $\approx$ for the limit of large n □

Therefore, e^{-k^2} is a safe upper bound for p_0 and a good approximation in the realistic case of large n and small k. By substitution, we can see that $\underline{k}$ = 4.5 gives $p_0 \sim 10^{-9}$ for all large n. For convenience, we typically recommend that $\underline{k}$ = 4.5. k must be large enough to allow for the level of Byzantine faults in the network. A typical practical value would be k = 10 to allow for 10% unavailable nodes and 50% fraudulent nodes, so we have $\underline{k}$ = 4.5 of honest available nodes. 10% unavailability seems generous for most modern networks, while 50% of fraudulent nodes is typically the most supported by the distributed ledger technology used for global ledger consensus.

Appendix A shows values of $p_0(n,r)$ and confirms that the approximation is excellent for large n and small k while providing a valid upper bound in all cases.

Now, double-spending with co-conspirators is in itself of no value as the co-conspirators will not provide any value in return for a payment that they know is fraudulent and may be later rejected from the global ledger. Therefore, the algorithm depends on ensuring a negative expected value when double-spent money directly or indirectly arrives at honest nodes, catching the fraud in time before honest nodes naively provide value in exchange for fraudulent payments.

Theorem 3. *Let M be the maximum transaction amount, m be the minimum wallet balance, n be the number of valid nodes as of the last global ledger consensus, h be the proportion of nodes assumed to be honest and u be the proportion of uptime required from nodes. Assume that the network is designed such that*

$p_0(n, \underline{k}\sqrt{n}) < \frac{m}{M+m} \approx \frac{m}{M}$,

where $\underline{k} = khu$. *Then, the expected return on any combination of double-spending to honest nodes is negative.*

Proof of Theorem 3. The maximum amount of a double-spend transaction is the maximum transaction amount M. By utilizing Lemma 2, the probability of two honest nodes not detecting a double-spend transaction is $p_0(n, \underline{k}\sqrt{n})$, in which case there is a gain of M. In the case that a double-spend transaction is detected, the double-spender will at least forfeit the minimum wallet balance m.

Therefore, the maximum expected gain is given by

$$p_0\left(n, \underline{k}\sqrt{n}\right)M - \left(1 - p_0\left(n, \underline{k}\sqrt{n}\right)\right)m.$$

Given $p_0\left(n, \underline{k}\sqrt{n}\right) < \frac{m}{M+m} \approx \frac{m}{M}$, this expected value is negative. □

As discussed, practical values are m = \$1, M = \$1,000,000 and $\underline{k}$ = 4.5 which give $p_0 \sim 10^{-9} \ll \frac{m}{M}$. Assuming h = 50% honest nodes (the algorithm can handle even fewer than 50% honest nodes but the blockchain cannot) and u = 90% uptime, we need k = 10 to ensure a negative expected value of any double-spend.

7. Algorithm message space complexity

The number of messages per transactions is $k\sqrt{n}$. These may be transmitted directly or cascaded through a tree of nodes to avoid the receiving node becoming a network bottleneck.

Before discussing message size, we present some definitions.

Definition 8. The critical inbound transaction size $j[T]$ is the number of inbound transactions (since the last global ledger consensus) which a sender depends on for their balance when spending money in a transaction t, that is $j[T] = |CLIN[T]|$. □

An upper bound for $j[T]$ is the total number of inbound transactions $|LIN[T]|$ that the node has participated in since the last global ledger consensus. In most transactions, j will likely be zero. A person spending money most often already had the money that morning. Although in extreme cases, v may be large. For example, a grocery store starting the day with zero balance, accepting hundreds of small transactions, and spending all their accumulated money on a large capital item or payroll that same evening. The large spend may depend on every one of the small inbound transactions.

Definition 9. The critical velocity of money $v[T]$ of a transaction T is the maximum depth of the recursion in $PED[T]$. □

$v[T]$ denotes the number of nodes that the specific balance of coin circulated through in the period between global ledger consensuses until it landed in transaction T, where it is only considered circulation if the nth transaction depended on its balance for the $(n-1)$th. v is generally assumed to be small, most often 1 and rarely more than 2. It is defined more narrowly than the economic concept of the velocity of money [36] that includes all circulation of currency whether critical to the spender's balance or not. The velocity of fiat money tends to average a very modest rate of 4–11 per year [37], so $v > 2$ in a single day between global ledger consensuses would be rather rare. That is, in say 24 hours of real commerce, a specific amount of currency will rarely change hands more than once or twice and at most a few times.

The number of transactions in the pedigree of a transaction T is the size of all transactions in the pedigree, and we can observe that $|PED[T]| = o\left(\bar{j}^{\bar{v}}\right)$, where $\bar{j}$ and $\bar{v}$ denote the maximum values of j and v for transactions in the pedigree recursion, respectively.

In the majority of the transactions, we expect v = 1 and occasionally v = 2 but rarely more, while j is most often 0 but may occasionally hit a few hundred. Thus, the message sizes in realistic commerce may typically be small; on rare occasions, we may have tens thousands of transactions and require some megabytes of message size, which is still quite a practical message size for a modern network.

However, if the algorithm is continuously used for days and weeks without a global ledger consensus, the message size v may grow prohibitively large.

8. Sybil attack

In a Sybil attack, a fraudster develops numerous fraudulent nodes hoping to reduce the chance of two honest nodes detecting the fraudster's double-spending. We already saw that a 50% attack does not provide a positive expected value of double-spending with the recommended network parameters. What about a still larger attack?

It is noteworthy that the global ledger consensus algorithm will typically fail with a 51% attack [38]; regardless we investigate whether such an attack could pay off in the k-root-n algorithm.

Suppose again that m = \$1, M = \$$10^6$ and k = 10. Assume that there are initially n honest nodes, and the fraudster creates another n fraudulent nodes to control 50%, and suppose further that 10% of the nodes are unavailable. We already saw that $\underline{k}$ = 4.5 and the fraudster has successfully reduced $p_0 \approx 10^{-44}$ to $p_0 \approx 10^{-9}$. But with $M/m=10^6$ there is no incentive to double-spend with $p_0 \approx 10^{-9}$.

In fact, the appendix shows that the fraudster needs to approximately get $\underline{k} < 3.5$ to obtain $p_0 > 10^{-6}$ and achieve a positive expected value for double-spending. Therefore, the criminal would need approximately $2n$ fraudulent nodes. However, the fraudster then faces another challenge. The loss from a single unsuccessful double-spending is not limited to forfeiting the double-spending wallet, but also the loss of all the nodes that failed to detect the double-spend and thereby committed a verification fraud. According to Theorem 1 the expected number of common nodes is $(10 - 3.5)^2 = 42.25$ nodes for an average loss of at least 42.25 m. Thus, even in this case, double-spending will have a negative expected value.

Consider more generally that a fraudster creates $(f - 1)n$ fraudulent nodes for a total of $n = fn$ nodes. When a user consults $k\sqrt{n} = k\sqrt{(fn)}$ nodes, a proportion of $1/f$ of the nodes or $k\sqrt{(n/f)}$ nodes, will be genuine, this being a proportion $k/\sqrt{(fn)}$ of all the n honest nodes. Two honest nodes will therefore have an expectation of consulting $(k/\sqrt{fn})(kn/\sqrt{f}) = k^2/f$ common honest nodes, i.e. $k = k/\sqrt{f}$. They will also consult on average $k^2 - k^2/f$ dishonest nodes, and if the double-spending is caught these $k^2(1 - 1/f)$ nodes will be disqualified.

Therefore, the expected payoff from a single double-spending is $p_0(k, n)M \approx e^{-k^2/f}M$, against the expected cost of $(k^2(1 - 1/f) + 1)m \approx k^2m$ (the fraudulent nodes which fail to report the double-spending, plus one for the double-spending wallet) for every unsuccessful transaction against a setup cost of $(f - 1)nm$.

Practically, an extremely large number of fraudulent nodes are required for the double-spending to payoff. Since k = 10, we can find numerically that we need approximately $f > 10.5$ for a positive payoff. Suppose f = 11; the fraudster has to create a massive $10n$ fake nodes to control ~91% of the network. This is done at a cost of \$$10nm$, assuming \$$10m$ for n = 1 million. Now, $k = k/\sqrt{f} \approx 3$ and $p_0 = e^{-k^2/f} = e^{-9} \approx 0.00012$. Thus, a double-spending of M = \$1,000,000 would have an expected value of \$120, while the expected cost would be losing $k^2(1 - 1/f) + 1 = 91$ nodes at a cost of $91m$ = \$91, giving an expected profit of \$29. After such an extreme attack, a single fraudulent transaction would have a positive expected value.

However, even this strategy is doomed to fail in realistic scenarios. If $n = 10^6$, it costs \$10 million to setup 10 million nodes. The user would have to repeat the double-spending three hundred thousand times to recoup the initial investment. However, they would lose 91 nodes on average each time they fail, meaning that they would lose the vast majority of the fraudulent nodes before recovering their investment, so the whole scheme is not feasible.

Now, if we increase f further say $f \approx k^2 = 100$, then the fraud can pay off. p_0 gets closer to 1 and the fraudster earns a payback that tends to M as f increases. However, this requires creating $O(100n)$ nodes to dominate ~99% of the network. Various strategies that can help to defend against such an extreme attack include the following.

Reducing M/m: Reducing M can force honest people to have more wallets that increase n.

- Increasing k.
- Biasing the $k\sqrt{n}$ random nodes towards the nodes that have been around for longer or have higher balances.
- Monitoring for suspicious behavior such as the creation a huge number of wallets with close to a minimum stake.

In summary, with the recommended parameters of k, m and M, we observed that the algorithm is immune to 51% attacks and even 90% attacks, and in fact resilient to all but the most extreme of Sybil attacks.

9. Variations on the algorithm for further research

9.1. k-root-n without global ledger consensus

There would be an option of running k-root-n as the sole algorithm without any common consensus on a ledger. As time goes on, j slowly increases, and the verifications may exponentially become heavier. Each node can should cache everything it knows about other nodes' verified transaction histories. Over time, if money circulates throughout the entire network, every node will end up verifying every transaction at some time or another just once with $k\sqrt{n}$ other nodes, creating in the long term a complexity of $kn\sqrt{n}$ per transaction that seems unattractive. However, there is room for optimizations which could make this approach of standalone k-root-n feasible.

9.2. Nodes versus wallets

The assumption so far is that every wallet is a node, and the provision of node verification services is part of the cost of being a wallet. This may be feasible as we rapidly move to a world where all devices are online almost all the time, but it could also be a limitation.

There could be an alternative variation of the algorithm in which not every wallet is a node. This may be helpful since people may want their wallets to be offline or to be stored on a machine with limited processing power, bandwidth or memory. In this scenario, nodes may be paid a fee to provide verification services with a penalty for failing to report a double-spend they were aware of. Moreover, the nodes could be the same machines as the nodes of the underlying blockchain. Further research is required to formally define such a network.

9.3. Forced validation

An alternative idea may be considered where even dishonest nodes are forced to consult $O(\sqrt{n})$ nodes. In this situation, the dishonest nodes are not given the opportunity to select which nodes they consult since they could pick collaborating dishonest nodes. Therefore, we may introduce a pseudorandom formula to dictate the nodes that are consulted, as well as also ensuring balancing the load between all nodes. In this situation, the idea is that if Alice sends money to Bob and Bob sends the same money to Charlie, then Charlie will again ask $k\sqrt{n}$ nodes to validate that Bob did not double-spend. However, Charlie will not need to ask the network to validate the transaction from Alice to Bob. Charlie can instead ask Bob to see the $k\sqrt{n}$ digital signatures for the appropriate nodes that signed

off on the transaction with Alice. Thus, Bob can verify that Bob indeed consulted the right set of $k\sqrt{n}$ and received all their approval, creating less traffic and processing demands on the network.

We therefore propose that when two people do a transfer, they must notify a formulaically determined pseudorandom selection of $k\sqrt{n}$ other nodes and obtain each of their digitally signed approval. Moreover, the pseudorandom selection is based on a predetermined formula which is known to all and takes as input e.g. sender's ID and the time of the transaction. To reduce the sender's ability to pick and choose a specific time, we preferably take time stamps to be at a resolution of a second or minute (rather than a more fine-grained time slot) when the pseudorandom formula happens to limit their ability to select numerous fraudulent nodes. As before, each of those nodes that are honest may check that the sender has not double-spent.

Now for the recursive check of sender's sender and so on, the receiver can check that all the recursive transactions have the necessary sign-off from all the nodes as determined by the pseudorandom formula. Thus, the receiver does not have to burden the network by validating the recursive transactions. Such an algorithm may scale better over longer periods.

This algorithm suffers from some clear vulnerabilities. In a real network, there is a high chance that some nodes may not be available, so the sender could feasibly calculate which k^2 nodes would detect his double-spending and simply claim that those particular nodes were not available. This would have to be mitigated by common monitoring of node availability or the honest nodes may self-monitor, so anyone can later validate the claim that a certain node was unavailable at a certain time.

The sender may also have multiple wallets and multiple available time slots allowing them some choice of the sending node and time slot, giving them some leeway to plan a double-spend without any clashes of verifying nodes by choosing the particular sending wallet and time slots wherein the pseudorandom formula happens to pick many of their own complicit nodes. If we choose k large enough, we can make this infeasible, for example with $k = 10$ and $p_0 = 3.70 \times 10^{-44}$, the user must consider $O(10^{44})$ combinations of wallets and time slots to obtain one with no clashes to an earlier transaction, which is not feasible.

Therefore, it should be feasible to design an algorithm wherein each node (honest or not) is forced to consult a particular set of $k\sqrt{n}$ nodes based on a function of the sender and time slot, and wherein there is no feasible way to create a positive expected value of double-spending.

10. Conclusion

For a distributed ledger, reaching consensus is expensive and may involve a long lag time. In this paper, we have explored a two-tier system in which the primary algorithm ensures that the global ledger consensus is reached for the distributed ledger, but perhaps only periodically and with high latency. In the meantime, a secondary k-root-n algorithm allows parties to transact rapidly and protect against double-spending with a more efficient $O(\sqrt{n})$ probabilistic algorithm which involves validating each transaction and recursively the transactions it depends on (the transaction's pedigree) with a random selection of $k\sqrt{n}$ nodes. Moreover, we showed that it is feasible for such a network to handle all the world's commerce, while always having a negative expected value of double-spending and being resistant to even aggressive Sybil attacks.

Further research is required to investigate the practicality of each wallet being a highly available node or develop the idea of separating wallets and nodes. Further research is required to check the feasibility of the alternative idea of formulaically dictating validating nodes.

Acknowledgments My thanks to Joshua Fox, Benjamin Fox, Aviv Zohar, for their valuable, constructive feedback.

11. Appendix A: Table of k and p_0

This table shows p_0, the chances of zero clashes when two honest nodes each consult $k\sqrt{n}$ random nodes, for various values of k and a couple of values of n.

Table 1. Values of $p_0(n,k\sqrt{n})$

k	$p_0(n=10^4, k\sqrt{n})$	$p_0(n=10^{10}, k\sqrt{n})$	e^{-k^2}
1	0.36	0.37	0.37
1.5	0.1	0.11	0.11
2	0.017	0.018	0.018
2.5	0.0016	0.0019	0.0019
3	9.30E-05	1.20E-04	1.20E-04
3.5	3.10E-06	4.80E-06	4.80E-06
4	5.80E-08	1.10E-07	1.10E-07
4.5	6.10E-10	1.60E-09	1.60E-09
5	3.70E-12	1.40E-11	1.40E-11
5.5	1.20E-14	7.30E-14	7.30E-14
6	2.30E-17	2.30E-16	2.30E-16
6.5	2.30E-20	4.50E-19	4.50E-19
7	1.30E-23	5.20E-22	5.20E-22
7.5	3.70E-27	3.70E-25	3.70E-25
8	5.60E-31	1.60E-28	1.60E-28
8.5	4.60E-35	4.20E-32	4.20E-32
9	1.90E-39	6.60E-36	6.60E-36
9.5	4.10E-44	6.30E-40	6.40E-40
10	4.40E-49	3.70E-44	3.70E-44

12. References

1. *S*. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2009.
2. *S*. *S*. *R*. Network, "The Limits to Blockchain? Scaling vs. Decentralization.," Cybersecurity, Privacy & Networks eJournal, 2019.
3. "Scalability https://en.bitcoin.it/wiki/Scalability," Bitcoin wiki, 2015.
4. J. Garzik, "Making decentralized economic policy http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf," 2015.
5. A. d. Vries, "Bitcoin's Growing Energy Problem," Joule, vol. 2, no. 15, pp. 801-805, 2018.
6. *R*. Canneti and *T*. Rabin, "Optimal Asynchronous Byzantine Agreement," Technical Report #92-15, Computer Science Department, Hebrew University, 1992.
7. *M*. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, 1999.
8. J. *R*. Douceur, "The Sybil Attack," Peer-to-Peer Systems. Lecture Notes in Computer Science., vol. 2429, *p*. 251–60, 2002.

9. L. S.A., "Blockchain Unconfirmed Transactions. https://www.blockchain.com/btc/unconfirmed-transactions".

10. D. Azzolini, F. Riguzzi and E. Lamma, "Studying Transaction Fees in the Bitcoin Blockchain with Probabilistic Logic Programming," Information, 2019.

11. "Irreversible Transactions https://en.bitcoin.it/wiki/Irreversible_Transactions," Bitcoin Wiki.

12. L. e. a. Luu, "SCP: A Computationally-Scalable Byzantine," IACR Cryptology, 2015.

13. Y. Gilad, R. Hemo, S. Micali, G. Vlachos and N. Zeldovich, "Algorand: Scaling Byzantine Agreements," SOSP '17 Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51-68, 2017.

14. I. Eyal, A. Efe Gencer, E. Gün Sirerr and R. van Renesse, "Bitcoin-NG: A Scalable Blockchain Protocol," e Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16), 2016.

15. L. Apeltsin, "A CryptoCubic Protocol for Hacker-Proof Off-Chain Bitcoin Transactions," Cornell University, vol. arXiv:1408.2824, 2014.

16. J. P. a. T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2019.

17. U. W. Chohan, "The Double Spending Problem and Cryptocurrencies," Banking & Insurance Journal, Social Science Research Network (SSRN), 2017.

18. E. Heilman, A. Kendler, A. Zohar and S. Goldberg, "Eclipse attacks on Bitcoin's peer-to-peer network," SEC'15 Proceedings of the 24th USENIX Conference on Security Symposium, pp. 129-144, 2015.

19. M. Apostolaki, A. Zohar and L. Vanbever, "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies," IEEE Symposium on Security and Privacy, 2017.

20. C. Pinzón and C. Rocha, "Double-spend Attack Models with Time Advantange for Bitcoin," Electronic Notes in Theoretical Computer Science, vol. 329, 2016.

21. E. Ittay, G. Peter, J. Aljosha, M. Sarah, S. Nicholas, T. Itay, W. Edgar and Z. Alexei, "Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies," IACR Cryptology ePrint Archive, 2019.

22. I. Stewart, D. Ilie, A. Zamyatin, S. Werner, M. F. Torshizi and W. J. Knottenbelt, "Committing to quantum resistance: a slow defence for Bitcoin against a fast quantum computing attack," Royal Society Open Science, 2018.

23. M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang and A. Mohaisen, "Exploring the Attack Surface of Blockchain: A Systematic Overview," arXiv, vol. 1904.03487, 2019.

24. G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais and S. Čapkun, "Misbehavior in Bitcoin: A Study of Double-Spending and Accountability," ACM Transactions on Information and System Security (TISSEC), vol. 18, no. 1, 2015.

25. C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas and J. Herrera-Joancomartí, "Double-spending prevention for Bitcoin zero-confirmation transactions," International Journal of Information Security, vol. 18, no. 4, p. 451–463, 2019.

26. K.-Y. Kang, "Cryptocurrency and Double Spending History: Transactions with Zero Confirmation," Munich Personal RePEc Archive (MPRA), vol. 96875, 2019.

27. J. Göbel and A. E. Krzesinski, "Increased block size and Bitcoin blockchain dynamics," 27th International Telecommunication Networks and Applications Conference (ITNAC), 2017.

28. N. R. Ericsson, D. F. Hendry and S. B. Hood, "Milton Friedman and Data Adjustment," IFDP Notes, Board of Governors of the Federal Reserve System.

29. W. W. R. Ball, "Probability," in Mathematical Recreations and Essays, Macmillan, 1960, p. 45.

30. "The World Population Prospects 2019: Highlights," Population Division of the UN Department of Economic and Social Affairs, 2019.

31. A. Lielacher, "How Many People Use Bitcoin in 2019?" Bitcoin Market Journal, 2019.

32. K. Sedgwick, "No, Visa Doesn't Handle 24,000 TPS and Neither Does Your Pet Blockchain," https://news.bitcoin.com/no-visa-doesnt-handle-24000-tps-and-neither-does-your-pet-blockchain/.

33. "Global General Purpose Cards - Midyear 2018," The Nilsen Report, vol. 1140, no. https://nilsonreport.com/upload/issues/1140_0321.pdf, p. 7, October 2018.

34. J. Skene, D. D. Lamanna and W. Emmerich, "Precise Service Level Agreements," ICSE '04 Proceedings of the 26th International Conference on Software Engineering, pp. 179-188, 2004.

35. D. Lamanna, J. Skene and W. Emmerich, "SLAng: A Language for Defining," The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS), 2003.

36. I. Fisher, The fPurchasing Power of Money, 1911.

37. Federal Reserve Bank of St. Louis, "Velocity of M1 Money Stock," Fred Economic Data, no. https://fred.stlouisfed.org/series/M1V, 2019.

38. M. Bastiaan, "Preventing the 51 %-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin," no. https://www.hmbastiaan.nl/martijn/docs/2015/preventing-the-majority.pdf, 2015.