UTXOs as a proof of membership for Byzantine Agreement based Cryptocurrencies

Emmanuelle Anceaume, Antoine Guellier, Romaric Ludinard

UTXOs as a proof of membership for Byzantine Agreement based Cryptocurrencies

Emmanuelle Anceaume
CNRS / IRISA, France
emmanuelle.anceaume@irisa.fr

Antoine Guellier
CNRS / IRISA, France
antoine.guellier@irisa.fr

Romaric Ludinard
IMT Atlantique, France
romaric.ludinard@imt-atlantique.fr

***Abstract*—The presence of forks in permissionless blockchains is a recurrent issue. So far this has been handled either *a posteriori*, through local arbitration rules (e.g., "keep the branch which has required the most computational power") which are applied once a fork is present in the blockchain, or *a priori*, via a Byzantine resilient agreement protocol periodically invoked by a committee of well identified and online nodes. In the former case, local arbitration rules guarantee that if they are correctly applied by a majority of the users of the system, then with high probability forks are progressively resolved, while in latter case, the sequence of Byzantine resilient agreements decide on the unique sequence of blocks to be appended to the blockchain. The question we may legitimately ask is the following one: To prevent the period of uncertainty inherent to optimistic-based solutions, are we doomed to rely on the decisions made by a unique committee whose members are already actively involved in the creation of blocs ? We negatively answer this question by presenting a solution that combines the best features of optimistic and pessimistic approaches: we leverage the presence of users and the "public-key as identities" principle to make users self-organize in small Byzantine resilient committees "around" each new object (i.e., blocks and transactions) to decide on their validity. Once validated, objects can be pushed in the network, appended to the blockchain without fearing any fork nor double spending attacks: we guarantee a "0"-confirmation delay. Additionally, our solution mitigates selfish attacks. We are not aware of any solutions enjoying such features.**

I. Introduction

Blockchains, also called distributed ledgers, initially appeared as the technological solution for the deployment of the Bitcoin digital cryptocurrency system, a secure system, usable by anyone, in a peer-to-peer way, with no trusted third party whatsoever. Blockchains achieve the impressive result of constructing a persistent, distributed, append-only log of transactions, and publicly auditable and writable by anyone in case of permissionless (i.e., public) blockchains. Construction of distributed ledgers typically relies on a sophisticated orchestration of cryptographic primitives, agreement algorithms, and broadcast communication primitives.

To face recurrent double-spending attacks — which are inherent to digital cash systems — blockchains are built so that their records (i.e blocks of transactions) are totally and securely ordered. So far, two main designs exist to totally and securely order blocks: the optimistic and the pessimistic approaches. The optimistic approach mainly consists in regularly running elections among a subset of the nodes of the system (i.e., miners), at the outcome of which, leaders (i.e., successful miners) properly and securely gathers transactions into blocks, with the hope that no concurrent blocks already exist in the system. Transient inconsistencies (i.e., the presence of concurrent forks) are locally handled, but at the expense of a substantially long block confirmation time. For instance Bitcoin guarantees that if a block has been stored for more than one hour in any local copy of the ledger, then it will remain there forever, and at the very same position in the local copies maintained at all the nodes of the system. This holds with very high probability even if up to 10% of the miners are malicious. More recently Ethereum [31] and Spectre [28] have succeeding in decreasing the block confirmation time but at the expense of more involved arbitration rules. The second approach, which we call pessimistic, aims at preventing forks from happening so that once recorded in the ledger, a block will never be pruned. This is achieved by relying on Byzantine resilient agreement algorithms (*e.g.* [8], [20]) fed with all the currently submitted transactions. An already impressive amount of work has been focusing on the properties of those algorithms to securely and totally order transactions in distributed ledgers, but the foremost difference that exists among all these works is related to the essential notion of *identity*. In consortium blockchains, including RedBelly [9] and HyperLedger [5], participants - those who control and manage copies of the blockchain - form a clique of carefully selected institutions with appropriate permissions. In Byzantine-based permissionless blockchains, such as PeerCensus [10], Bizcoin [19], or BitcoinNG [12], those in charge of executing the Byzantine resilient agreement algorithms are selected among all the successful miners. Note that one may be careful in the way miners are selected to form the Byzantine algorithms committees not to violate the system safety [2].

To summarize, the pessimistic approach achieves an irrevocable decision on the next block to be appended to the permissionless ledger at the cost of running a Byzantine-tolerant algorithm among several hundred of nodes for each created block. The optimistic approach guarantees that in presence of forks on the ledger the probability that a given branch will remain in the ledger increases exponentially with the number of blocks appended after it.

Very recently, an elegant pessimistic-based approach to mitigate the presence of blockchain forks has appeared with

Algorand [14]. In Algorand, members of the Byzantine-tolerant algorithm are selected no more proportionally to their computational power, but proportionally to their stake. Among them, a leader is elected, the one with the largest stake, and handles all the currently submitted transactions. Algorand guarantees that in periods of strong synchrony, the blockchain correctly grows (absence of forks and double-spending), while in presence of variable communication delays, growth is not guaranteed.

In this paper, we propose a solution that borrows ingredients of both approaches to guarantee that once a block, mined in isolation, is declared *valid* by the system, then it cannot be confronted with any other conflicting block, and thus will irremediably be registered in the ledger. Our solution also guarantees that double-spending attacks are detected and prevented once any conflictual transaction is submitted to the peer-to-peer network. Hence, once a transaction is declared *valid* by the system, then it cannot be confronted with any other conflicting transaction, relieving sellers of any fraud.

The *block validation* protocol and the *transaction validation* one are essential in our design. The core of both protocols relies on Byzantine agreements (BA). BA allows the block validation protocol to reach an agreement on the unique block that can reference an earlier block in the blockchain, and allows the transaction validation protocol to reach an agreement on the unique transaction that can redeem all the unspent transaction outputs (UTXOs) referenced as its inputs. In contrast to all the pessimistic approaches described above (and to the best of our knowledge to all them), BAs are executed "around" the objects to be validated. It briefly means that, when a newly created transaction is submitted to the peer-to-peer network for validation, the transaction validation protocol is executed by a subset of users randomly chosen among those that are (logically) close to the UTXOs referenced in the input of the transaction. Participation of those randomly chosen users is publicly verifiable by anyone. Similarly, when a newly created block is submitted to the network, the block validation protocol is executed by a subset of users randomly chosen among those that are (logically) close to the predecessor block referenced by this block. Participation of those users is also publicly verifiable. As an additional consequence of our design, selfish attacks are mitigated. Since a selfish miner can not create a block without disclosing it, no one is able to built a private sequence of blocks in order to prune the tail of the blockchain. Finally, beyond giving users an incentive to correctly behave, relying on their participation to execute Byzantine agreement protocols allows for greater equality in the management of the blockchain. By the "locality" principle of the validation protocols, participation in the execution of a Byzantine agreement protocol is de facto temporary and unfrequent.

The remainder of the paper is organized as follows. Section II describes the computational and system model adopted in this work and then presents the main features of blocks and transactions in Bitcoin. Section III presents a brief survey of some of the attempts that have been made at solving Bitcoin issues. Section IV presents the key elements that we leverage to derive our solution. Section V describes the orchestration of these elements to validate blocks and transactions. Section VI and Section VII detail implementations of both protocols. Finally, Section VIII concludes.

II. Model, transactions and blocks

A. Model

We assume a large, finite yet unbounded set Π of nodes whose composition may change over time. Nodes communicate with each other through unreliable channels, meaning that messages can be lost, altered, duplicated or reordered. We assume the existence of a finite but unknown upper-bound on message propagation time, which fits the the partial synchrony model [11].

We suppose that a bounded proportion μ, with $\mu \leq \lfloor 1/3 \rfloor$ of the nodes in Π are Byzantine (i.e., behave arbitrarily, either in collusion or on their own to maximize some utility function of their choice). All the other nodes are said correct or honest. We assume that nodes have access to basic cryptographic functions, including a cryptographic hash function $\mathfrak{h}$ – modeled as a random oracle – and an asymmetric signature scheme – that allows nodes to generate public and secret key pairs (p_r, s_r), to compute signatures $\sigma_{r,\mathfrak{h}(d)}$ on messages d, and to verify the authenticity of a signature. These primitives are assumed to be safe – i.e., forging signatures, and finding hash collisions, pre-image or second pre-image is impossible. By these properties, each object o of the system – *i.e.*, UTXO (for unspent transaction output), transaction and block – is assumed to be uniquely identified.

We assume that correct nodes use their cryptographic keys in a safe way, *i.e.* they do not disclose, share or drop their secret keys. As a consequence, their identity can not be spoofed and their received coins cannot be stolen. We do not suppose the existence of any trusted public key infrastructure (PKI) to establish nodes identities. Finally, we assume that each object o is well-formed. For example, a transaction can be rejected (by a correct node) only if that transaction tries to double-spend inputs and not *e.g.* because a script is not correctly written.

B. Transactions and blocks

Prior to describing our solution, let us first recall some background on transactions and blocks manipulated in most of the cryptosystems that derive from Bitcoin. A transaction is made of two sets, the input set denoted by I and the output one denoted by O. Set I contains the set of outputs, credited by previous transactions, that the creator of the transaction wishes to spend, together with the proof that she is allowed to redeem each of those outputs. The output set O contains transferred coins, together with the challenges that will allow their owners to redeem those coins.

Transactions outputs are locked with a challenge and redeemed in subsequent transactions by providing the appropriate response. Different types of scripts exist in Bitcoin, but

the most common one is the PAY-TO-PUBKEY-HASH script.[1] In that script, the challenge embeds the hash value $h = \mathfrak{h}(p_r)$ of a public key p_r and the response of the challenge contains the public key p_r together with the signature σ_r signed with the secret key associated to p_r. Thus the only user able to provide the appropriate values of σ_r and p_r is the effective owner of the transaction output, that is the owner of the UTXO. As a consequence, double spending attacks can only be launched by (malicious) users that create distinct transactions redeeming exactly one of their transaction outputs.

Once created, a transaction is submitted to the peer-to-peer network. Each node of the network should check the validity of the transaction prior to propagating it to its neighborhood. Informally, a transaction $T = (I, O)$ is *locally valid* at node p if p has received all the transactions that have credited all the inputs in I and for all $i \in I$, i is not in a *double-spending situation*. Input $i \in I$ is in a double-spending situation if p is aware of transaction $T' = (I', O')$ such that $i \in I \cap I' \neq \emptyset$.

Transaction $T = (I, O)$ is *conflict-free* if none of the inputs of T is involved in a double-spending situation and all of the transactions that credited T's inputs are conflict-free. By construction, the induction is finite at least in Bitcoin, because money is created only through coinbase transactions, which are by definition conflict-free [2].

Blocks are created by *successful miners*, a subset of the nodes involved in the *proof-of-work* competition. The incentive to participate to such a competition is provided by a reward given to each successful miner. This reward is made of a fixed amount of coins (in Bitcoin, the reward is currently equal to 12.5 bitcoins) and a fee associated to each transaction contained in the newly created block. This reward is inserted in the output of a particular transaction, called the coinbase transaction. Note that coinbase transactions do not have inputs.

A block is made of two parts: the header and the payload. The payload contains a unique coinbase transaction and a list of valid transactions. The header of the block contains several fields among which the reference to its parent block (hence the block*chain*), a *proof-of-work*, that is a nonce such that the hash of the block matches a given target (in Bitcoin, this target is calibrated so that the mean generation time of a block is equal to 10 minutes), and the fingerprint of the payload. In the following, we refer by $b = (\mathfrak{h}(b'), c(b))$, a block with a parent block reference $\mathfrak{h}(b)$ and a payload $c(b)$.

When a transaction T is included in a block b, it is said *confirmed* by all the peers that accept that block in their local copy of the blockchain. The *level of confirmation* of transaction T is the number of blocks included in the blockchain starting from b; by extension, a 0 confirmation level means that the transaction has not yet been included in the blockchain. To limit double-spending attacks, Bitcoin recommends that sellers do not provide their goods in exchange of a transaction before it becomes *deeply-confirmed*. Actually, Nakamoto [25] as well as subsequent studies [13], [18], [24] have shown that if the computational power of malicious miners is equal to 10% of the whole computational power, then with probability less than 0.1%, a transaction can be rejected if its level of confirmation in a local copy of the blockchain is less than 5. In the following, we say that a transaction is *deeply confirmed* once it reaches such a confirmation level.

Most of the permissionless blockchain-based cryptosystems guarantee the following two properties:

- **Safety** If a transaction T is deeply confirmed by some correct node, then no transaction conflicting with T will ever be deeply confirmed by any correct node.
- **Liveness** A conflict-free transaction will eventually be deeply confirmed in the blockchain of all correct nodes at the same height in the blockchain.

I case of a blockchain fork, some blocks can be invalidated and the level of confirmation of their transactions can decrease, especially if the conflicting branch contains a conflicting transaction. This deters the use of Bitcoin for fast payment, as the expected time for a deep confirmation is approximately one hour. Fast payment are used in most everyday life situations, where the time between buying and consuming the goods is in the order of minutes. This impracticality motivates this work.

In the present paper, by preventing double-spending attacks and blockchain forks we aim at strengthening the safety property as follows:

- **Strong Safety** If a transaction T is confirmed by some correct node, then no transaction conflicting with T will ever be confirmed by any correct node.

The strong safety property ensures that whenever a transaction T has been included in a block, no other block will ever contain a transaction conflicting with T. An immediate and important consequence of this property is the capability blockchain-based cryptosystems to safely handle fast payments. The remaining of the paper is devoted to the implementation of this property.

III. Related work

Bitcoin [25] is seen as the pioneer of cryptocurrencies. Since its inception, several altcoins [1] have emerged. The GHOST protocol [29] proposes a different rule to solve blockchain forks, based on the number of blocks contained in each blockchain subtree (in case of consecutive forks). Recent works have focused on Bitcoin modeling and evaluation. Authors of [24] prove that the Bitcoin protocol achieves consensus with high probability, while [13] show that peers participating in the Bitcoin network agree on a common prefix for the transaction history, both in failure-free environments. In contrast, authors of [17], [18] focused on adversarial environments. These works study the feasibility of double spending attacks and their detection. Several studies have shown that Bitcoin behaves quite well in failure-free environments [24] but is vulnerable to some attacks such as the double-spending one [18]. Several attempts to fix it have been published, using a leader [10], [12], [19], or forming local committees to run consensus

[1]Note that different Bitcoin verification scripts exist and some of them do not rely on cryptography (ANYONE-CAN-SPEND) or do not ensure a unique recipient (ANYONE-CAN-SPEND, TRANSACTION PUZZLE). In the following, we will consider only PAY-TO-PUBKEY-HASH scripts.

algorithms at the local level [22] but these proposals encounter various scalability or security issues which make them unusable. Specifically, Bitcoin-NG [12], PeerCensus [10], and BizCoin [19], have proposed to rely exclusively on miners to take in charge the full process of validation and confirmation to guarantee that all the operations triggered on the transactions are atomically consistent. Atomic consistency guarantees that all the updates on shared objects are perceived in the same order by all entities of the system. In all these protocols, time is divided into epochs. An epoch ends when a miner successfully generates a new block. This miner becomes the leader of the subsequent epoch. Each of these solutions rely on a dedicated set $\mathcal{E}_\ell$, with $\ell \in \{1, w, \infty\}$. This set is built along consecutive epochs as follows. At epoch k, if $|\mathcal{E}_\ell| < \ell$, the new leader is added to $\mathcal{E}_\ell$. Otherwise, the leader at epoch $k + 1 - \ell$ is removed from $\mathcal{E}_\ell$ and the new leader is added. Once set $\mathcal{E}_\ell$ reaches size ℓ, it remains at constant size ℓ. Strong consistency is implemented in these protocols by different means. In Bitcoin-NG, it is achieved by delegating the validation process to $\mathcal{E}_1$, *i.e.* the leader of the current epoch. In PeerCensus it is implemented by relying on Byzantine Fault Tolerant consensus protocols (*e.g.* [8], [15], [20]) run by $\mathcal{E}_\infty$ (recall that it contains all the miners that successfully generated a block). Finally, BizCoin leverages both ideas by using the leader and a consensus run by $\mathcal{E}_w$. In all these protocols, members of $\mathcal{E}_\ell$, with $\ell \in \{1, w, \infty\}$, are entitled to validate and confirm issued transactions and blocks and to disseminate them so that each peer integrates them in its local blockchain. It has been shown in a previous paper [2] that none of the studied solutions enhances Bitcoin's behavior. Beyond the complexity introduced by the consensus executions, the main issue comes from the fact that all important decisions of Bitcoin are solely under the responsibility of (a quorum of) miners, and the membership of the quorum is decided by the quorum members. This magnifies the power of malicious miners.

Very recently, an elegant pessimistic-based approach to mitigate the presence of blockchain forks has appeared with Algorand [14]. In Algorand, members of the Byzantine-tolerant algorithm (BA*) are selected no more proportionally to their computational power, but proportionally to their stake. Among them, a leader is elected, the one with the largest stake, and handles all the currently submitted transactions. Algorand guarantees that in periods of strong synchrony, the blockchain correctly grows (absence of forks and double-spending), while in presence of variable communication delays, growth is not guaranteed.

The current paper improves upon a previous work in which the idea of validating transactions and blocks as early as possible was introduced [21]. To cope with the risk of Sybil attacks, participants to BA committees in [21] have to solve a computational puzzle to create their current identities [22], which in expectation makes the number of identities per node proportional to its computational power. In the present solution, we rely on UTXOs owners to participate to BA committees for the following reasons. First by relying on the public key as identity principle, anyone can easily verify BA committees membership, and second by relying on the fact that UTXOs are one shot objects (i.e., once debited an UTXO does not exist anymore), an induced churn is generated allowing honest participants to escape poisoning attacks by moving to a new region of the system, and preventing malicious nodes from staying indefinitely long in the same region of the system, healing the system from eclipse attacks.

IV. A SET OF INGREDIENTS

The main objectives of our solution are twofold: *(i)* the guarantee that only non conflictual objects – blocks and transactions – are validated and propagated in the system in order to prevent arbitration rules from being applied a posteriori and, *(ii)* the execution of the block and transaction validation process over distinct committees to mitigate adversarial behaviors (collusion and eclipse attacks) and to improve the system scalability. Our solution relies on the orchestration of the following ingredients.

Byzantine agreement (BA). The first ingredient we use to implement the validation process is Byzantine agreement. Informally, Byzantine agreement (BA) is a communication protocol enabling a set of committee members, each of which holds a possibly different initial value, to agree on a single value v. Such an agreement is reached by all honest members, that is, by those who scrupulously follow the protocol despite the fact that a minority of the members are malicious and can deviate from the protocol in an arbitrary and coordinated manner.

Distributed Hash Table (DHT). The second ingredient of our solution is a distributed hash table. Recall that DHTs build their topology according to structured graphs, and for most of them, the following principles hold: the identifier space, e.g., the set of 256-bit strings, is partitioned among all the nodes of the system, and nodes self-organize within the graph according to a distance function D based on node identifiers (*e.g.* two nodes are neighbors if their identifiers share some common prefix), plus possibly other criteria such as geographical distance. Example of DHTs are [26], [30], [23], [27]. For resiliency reasons, each vertex of the graph can be a set or a cluster of nodes. Basically, nodes sharing a common prefix gather together into clusters, and clusters self-organize into a graph topology, for instance an hypercube. By running distributed algorithms inside each cluster, cluster-based DHTs can be made robust to high churn [16] and adversarial attacks [3].

Cluster-based DHT. In the following we use PeerCube, a cluster-based DHT to implement the validation protocols [3]. Briefly, PeerCube is a DHT that conforms to an hypercube. Vertices of the hypercube are clusters of nodes. Each cluster is dynamically formed by gathering nodes that are close to each other according to a distance function D applied on the bit string identifier space. Distance $\mathcal{D}$ consists in computing the numerical value of the "exclusive or" (XOR) of bit strings.

Thus identifiers that have longer prefix in common are closer to each other, and for any point p and distance Δ there is exactly one point q such that $\mathcal{D}(p,q) = \Delta$ (which does not hold for the Hamming distance). Nodes whose identities share a common prefix gather together within the same cluster. Each cluster is uniquely identified with a *label* that characterizes the position of the cluster in the overall hypercubic topology. The label of a cluster is defined as the shortest common prefix shared by all the users of that cluster such that the *non-inclusion* property is satisfied. The non-inclusion property guarantees that a cluster label never matches the prefix of another cluster label, and thus ensures that each identifier belongs to at most one cluster. The length of a cluster label, i.e. the number of bits of that label, is called the *dimension* of the cluster. In the following, notation d-cluster denotes a cluster of dimension d. Dimension determines an upper bound on the number of links a cluster has with other cluster of the DHT, i.e. the number of its neighbors. Clusters self-organize into a hypercubic topology, such that the position of a cluster into the hypercube is determined by its label. Ideally the dimension of each cluster $\mathcal{C}$ should be equal to some value d to conform to a perfect d-hypercube. However, due to the fact that nodes join and leave the system at anytime, and their identifiers are random bit strings, then cluster membership evolve, and thus clusters may grow or shrink more rapidly than others. In the meantime, cluster size are bounded. Whenever the size of $\mathcal{C}$ exceeds a given value S_{max}, $\mathcal{C}$ splits into clusters of higher dimensions, and whenever the size of $\mathcal{C}$ falls under a given size of S_{min} nodes, $\mathcal{C}$ merges with other clusters into a single new cluster of lower dimension. Members of each cluster run Byzantine agreement protocols to guarantee that the functioning of the DHT is correct despite targeted attacks [4]. This is achieved by partitioning each cluster into two sets, core members and spare members. The number of core members is at any time kept constant to handle a proportion μ of malicious nodes, while the spare set gathers all the other node of the cluster, and by doing so handle the churn without impacting the topology of the hypercube [3].

"Public keys as identities" principle. The third ingredient of our solution is to use (verification) public keys as user identities. This means that users can use their public keys as a reference to them. Digital signatures enables this because one has the ability to verify the validity of an information based on the public key, information, and signature. This principle is at the core of challenges present in transactions to redeem coins of UTXOs, and as detailed below we deeply use the unique association UTXO/identity as a proof of membership for BA.

V. Distributed Hash Table has a support for running distinct instances of Byzantine agreements

We now describe how the above ingredients are orchestrated to validate transactions and blocks.

BA committee members are the owners of UTXOs. BA committee members are the owner of UTXOs, that is the users of the cryptocurrency system – this clearly differs from most of the BA-based blockchains in which BA protocols are executed by the successful miners. Note that as each user may own a multitude of UTXOs (i.e. may have a multitude of identities), a user may belong to several distinct BA committees. As will de described in the following, any committee member may prove its right to belong to a given committee by exhibiting a digital signature that verifies an UTXO that has never been redeemed so far.

BA committees as vertices of the DHT. To cope with the thousands of transactions to be validated per day, a multitude of BA protocols are run in parallel, each one sitting at the vertices of PeerCube. Thus a BA committee is identified by a unique label, which is the shortest common prefix shared by all the users (i.e. UTXOs owners) of that BA committee. Validation of each transaction T is handled by a specific BA committee, the one whose label is a prefix of T identifier – the identifier of a transaction is equal to its hash. Recall that by construction, PeerCube guarantees that any identifier belongs to a single cluster. In the following the BA committee to which T is affected is called T referee. Similarly, validation of a block is handled by a unique BA committee. However, in contrast to transactions, the referee of a block B is the BA committee whose label is a prefix of B predecessor (Recall that blocks form a chain by pointing to a predecessor).

UTXOs to prevent targeted attacks. It is very important to understand that, since UTXOs are one shot objects – that is UTXOs are debited once and then disappear – the presence of users in committees is verifiable. Anyone in the system can check that some user is allowed to participate to the execution of BA in a given cluster by just checking in her blockchain that the identity of that user, that is its public key, and thus its UTXO has never been redeemed so far. This feature is important as it is a very efficient way to prevent targeted attacks, attacks in which collusion of malicious nodes devise strategies to progressively take the leadership of a targeted region by staying longer than honest nodes [6], [4]. By the second pre-image property of public keys, it is also very difficult for malicious nodes to generate identities that allow them to choose their positions so as to form collusions inside committees.

VI. Transaction validation protocol

The purpose of the transaction validation protocol is to prevent double spending attacks by ensuring that concurrent transactions do not try to use common inputs. Say differently, its objective is to guarantee that at any time at most one transaction can redeem all the UTXOs referenced in its input set.

If we make an analogy between *transaction inputs* and *objects*, and an analogy between *using an input* and *writing an object*, then we can refer to database systems, in which exclusive access to objects is obtained by asking each transaction to explicitly lock objects it accesses using some single

object locking mechanism. Yet, unless care is taken, locking objects one by one may cause deadlocks. As the application we consider involves different nodes spread over a large area, it is not advisable to rely on having all of them conform to the same locking strategies. Moreover, from a performance viewpoint, it may be impossible to run deadlock detection and prevention protocols assuming independent object locking. In the following we propose a transaction validation protocol that provides the equivalent of an atomic locking mechanism for all of the inputs of each issued transaction.

Formally, our transaction validation protocol implements two methods, `grantInputs` and `release`, that both accept a transaction $T = (I, O)$ as parameter. The `grantInputs` method returns with GRANTED or DENIED. When an invocation returns with GRANTED, we say that *the method exclusively grants the inputs in I to T* or, in short, that *T has been* GRANTED. Once T has been GRANTED, for any subsequent transaction T' conflicting with T, the service returns DENIED. T can invoke the `release` method only if T has not been granted. Otherwise it has no effect.

The transaction validation protocol prevents double spending attacks if the following three properties are met:

- **Safety:** If a transaction $T = (I, O)$ is exclusively granted the inputs in I, then no other transaction $T' = (I', O')$ is exclusively granted the inputs in I' with $I \cap I' \neq \emptyset$.
- **Liveness:** Each invocation of the `grantInputs` method eventually returns.
- **Non triviality:** If there exists an invocation of the `grantInputs` method with $T = (I, O)$, and no other transaction $T' = (I', O')$ with $I \cap I' \neq \emptyset$ is exclusively granted the inputs in I' then T is granted exclusively all the inputs in I.

As evoked above, the referee of each transaction $T = (I, O)$ is the BA committee whose label share a common prefix with T. Let us call it pi_T. pi_T is in charge of invoking the `grantInputs` method for T and possibly the `release` method if T has not been granted. Granting a transaction means granting all the inputs of the transactions. Thus similarly to the referee of a transaction $T = (I, O)$, each input $i \in I$ of T has a referee. we call it UTXO referee, which is the BA committee whose label is a prefix of $i \in I$. Let us call it pi_i.

Therefore, when a user creates a transaction $T = (I, O)$, it submits T to PeerCube that routes T to its referee pi_T. For each input $i \in I$ of T, pi_T asks an exclusive lock at the referee pi_i of each input $i \in I$, in an order that corresponds to the lexicographical order of the input IDs. If the lock is DENIED for at least one of these inputs, pi_T releases all previously obtained locks (by proving to each of these referees that a conflicting transaction T' has already been GRANTED). Otherwise, after obtaining all locks, a GRANTED status is returned to pi_T. Thus, similarly to transaction referees, UTXO referees are characterized by the following properties:

- **Safety:** if an UTXO u is spent by a transaction $T = (I, O)$, $u \in I$, then no other transaction $T' = (I', O')$, $u \in I'$ can spend u, i.e. T' is considered as invalid.
- **Liveness:** Each invocation of the `grantUTXO` method eventually returns.
- **Non triviality:** If there exists an invocation of the `grantUTXO` method for an UTXO $u \in I$ with $T = (I, O)$, and no other transaction $T' = (I', O')$ with $u \in I \cap I'$ is exclusively granted the inputs in I' then u is granted exclusively for T.

Protocol 1 is the pseudo-code run by the referee of a transaction to orchestrate the grant requests on the involved UTXOs referees as described in the pseudo-code of Protocol 2.

The correctness and, in particular, the lack of deadlocks, result from the fact that objects are always obtained in lexicographical order. A lock can be implemented using a combination of Test-and-Set and Reset primitives. The referee pi_i that wishes to lock input $i \in I$, first checks the value of a binary register. When this value is 0, it modifies the register to 1 and uses the lock. Releasing a lock is done by resetting to 0 the register value. The fact that T has been granted the lock on each input $i \in I$ is proven by pi_i's signature. Each signature is bundled with the identity of the signer. Note that Bitcoin transactions can easily be extended to accommodate this process: the referee pi_T of transaction $T = (I, O)$ computes a group signature S (*e.g.* [7]) using the signatures of each input referee $pi_{i_{i \in I}}$ and its own signature and appends it, along with everything needed to verify this group signature, to a specific *validation* output o added to the set of outputs O of transaction $T = (I, O)$. Any node can easily verify that transaction $T = (I, O)$ has been GRANTED by checking the signatures S added by referee pi_T.

Referees are incentivized by introducing a *validation fee*. A fair and easy way to share the *validation* output is to randomly pick one of the referees and give it the entire reward. This requires seeding a random number generator in a publicly verifiable way, and for example with an information that can only be published after the transaction validation protocol has returned, like the hash of the block in which the transaction is included.

VII. Block validation protocol

By following exactly the same validation transaction principle, BA committees are exploited to prevent blockchain forks, that is ensure that any validated block has at most one valid block as immediate successor. It is achieved by providing a method `grantBlock` that accepts a block $b' = (\mathfrak{h}(b), c(b'))$ as parameter. This method returns with GRANTED or DENIED. When an invocation returns with GRANTED, we say that *the method validates* block b' as the unique successor of block b, i.e. block b' is granted for $\mathfrak{h}(b)$. This method has to satisfy the three following properties:

- **Safety:** If a block $b' = (\mathfrak{h}(b), c(b'))$ is granted for $\mathfrak{h}(b)$, then no other block $b'' = (\mathfrak{h}(b), c(b''))$ is granted for $\mathfrak{h}(b)$.
- **Liveness:** Each invocation of the `grantBlock` method eventually returns.
- **Non triviality:** If there exists an invocation of the `grantBlock` method with $b' = (\mathfrak{h}(b), c(b'))$, and no

Protocol 1: Transaction validation protocol

```
1  Upon reception of (GRANT, T=(I,O), user) begin
2      if ∃T' = (I', O') ∈ B, I ∩ I' ≠ ∅:
           // a conflicting transaction
           // exists in the blockchain
3          Send (DENIED, T, T');
4      BA_Propose ((GRANT, T, user));
5  Upon reception of BA_Decision((GRANT, T, user)) begin
       // transaction committed by PBFT
6      pending[h(T)] ← (T, user);
7      signature[h(T)] ← multisign (T);
8      inputs ← sort(I);
9      for i in inputs:
10         granted[h(T)][i] ← false;
11     current[h(T)] ← 1;
       // starts asking input grants
12     DHT_Route ((GRANT, h(T), inputs[current[h(T)]],
         signature[h(T)]));
13 Upon reception of (GRANTED, h(T), i) begin
14     (T = (I,O), user) ← pending[h(T)];
15     if current[h(T)] < |I|:
16         granted[h(T)][i] ← true;
           // try to grant the next input
17         current[h(T)] ← current[T] + 1;
18         DHT_Route ((GRANT, h(T), inputs[current[h(T)]],
             signature[h(T)]));
19     else:
           // all inputs have been granted
           // execute the transaction
20         BA_Propose ((GRANTED, (T, signature[h(T)]));
21 Upon reception of (DENIED, h(T), T', i) begin
22     (T = (I,O), user) ← pending[h(T)];
23     granted ← {i ∈ I | granted[h(T)]};
       // release previously granted inputs
24     for input in granted:
25         Sends (RELEASE, i, h(T), signature[h(T)]);
26     Send (DENIED, T, T', signature[h(T)]) to user;
27 Upon reception of BA_Decision( (GRANTED, (T=(I,O), σ_T) ) begin
28     broadcast (GRANTED, T,σ);
29     if I∩ BA_View ≠ ∅:
30         BA_ViewChange (I∩ PBFT_View);
```

Protocol 2: UTXO conflict handling protocol

```
1  Upon reception of (GRANT, h(T), input, σ) begin
2      if granted[input] = (h(T'), σ'):
           // previously granted... deny !
3          Reply (DENIED, h(T), h(T'), input)
4      BA_Propose ((GRANT, h(T), input, σ));
5  Upon reception of BA_Decision((GRANT, h(T), input, σ)) begin
6      if granted[input] = ⊥:
           // store the grant
7          granted[input] ← (h(T), σ);
8          Reply (GRANTED, h(T), input);
9      else:
           // any subsequent request is denied
10         Reply (DENIED, h(T), h(T'), input)
11 Upon reception of (RELEASE, i, h(T), σ) begin
12     if granted[input] = (h(T), σ):
           // release the grant on the input
13         BA_Propose (RELEASE, i, h(T), σ);
14 Upon reception of BA_Decision( (RELEASE, i, h(T), σ) ) begin
15     if granted[input] = (h(T), σ):
           // release the grant on the input
16         granted[input] ← ⊥;
```

Protocol 3: Block validation protocol

```
1  Upon reception of (GRANT, b : block, minerAddr)
2  begin
3      if granted[b.prev] = b' :
           // a block b' referring to b.prev
           // have already been granted, it thus
           // belong to the local blockchain
4          Send (DENIED, b.prev, b') to minerAddr;
5      else:
           // not aldready granted, try!
6          BA_Propose (GRANT, b, minerAddr);
7  Upon reception of BA_Decision (GRANT, b : block, minerAddr) begin
8      if granted[b.prev] = ⊥ :
           // update local state
9          granted[b.prev] ← b;
           // multi signature : f+1 out of S_min
10         σ_b ← multisign (b);
11         broadcast (GRANTED, (b, σ_b)) ;
12     else:
           // Any subsequent request is denied
13         Send (DENIED, b.prev, b') to minerAddr;
```

invocation of `grantBlock` with $b'' = (\mathfrak{h}(b), c(b''))$ has ever been granted, then block b' is granted as the unique successor of block b.

The referee of block B is the BA committee whose label share a common prefix with B predecessor. Let us call it π_B.

Therefore, when a miner creates a block B it submits a `grantBlock` request for B to PeerCube that routes it to its referee pi_B. The request is granted if pi_B has never granted such a request before. Protocol 3 is the pseudo-code of the block validation protocol.

To summarize, the validation protocols guarantee via BA committees tessellated at the vertices of a DHT that that a transaction is validated only if it is the only one to redeem each of its inputs, and a block is validated if it is the unique successor of its predecessor. This is achieved by relying on the ephemeral participation of UTXOs owners as BA committees members. Indeed once an UTXO has been granted by a committee, that UTXO does not exist anymore, and thus its owner leaves the BA committee, and joins the BA committees whose labels prefix the newly created UTXOs. This is very important to prevent eclipse attacks, that is strategies which allows the adversary to stay forever at the same position in order to progressively eclipse honest nodes around it.

VIII. Conclusion

In this paper we have presented a new idea to prevent both blockchain forks and double spending attacks, without relying on successful miners to impose a total ordering on the blocks they have mined. Our design relies on the ephemeral participation of UTXO owners, and exploits both the scalability and robustness properties of cluster-based DHTs and the "public key as identity" principle. Those ingredients allow us to introduce a small amount of synchronization, soon enough

in the validation process, to guarantee that sellers can to deliver their good as soon as the transaction has been validated by the system. Fast payments transactions are thus no more an issue. The same level of local synchronization heals the Bitcoin system from blockchain forks, and selfish mining. We are currently evaluating the performance of our design through an implementation that we have deployed over several hundred nodes. Preliminary results are very promising.

REFERENCES

[1] S. Ahamad, M. Nair, and B. Varghese. A Survey on Crypto Currencies. In *Proceedings of the International Conference on Advances in Computer Science (AETACS)*, 2013.

[2] E. Anceaume, T. Lajoie-Mazenc, R. Ludinard, and B. Sericola. Safety Analysis of Bitcoin Improvement Proposals. In *15th IEEE International Symposium on Network Computing and Applications (NCA)*, 2016.

[3] E. Anceaume, R. Ludinard, A. Ravoaja, and F. Brasileiro. PeerCube: A Hypercube-Based P2P Overlay Robust against Collusion and Churn. In *2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO 2008.

[4] E. Anceaume, R. Ludinard, and B. Sericola. Performance evaluation of large-scale dynamic systems. *ACM SIGMETRICS Performance Evaluation Review*, 39(4), 2012.

[5] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. https://arxiv.org/pdf/1801.10228v1.pdf.

[6] B. Awerbuch and C. Scheideler. Group spreading: A protocol for provably secure distributed name service. In *Proceedings of the 31rst International Colloquium on Au- tomata, Languages and Programming (ICALP)*, 2004.

[7] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *6th International Workshop on Practice and Theory in Public Key Cryptography*, PKC 2003.

[8] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[9] T. Crain, V. Gramoli, M. Larrea, and M. Raynal. (Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains. http://csrg.redbellyblockchain.io/doc/ConsensusRedBellyBlockchain.pdf, 2017.

[10] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin Meets Strong Consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN)*, 2016.

[11] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the Presence of Partial Synchrony. *J. ACM*.

[12] I. Eyal, A. E. Gencer, E. Gün Sirer, and R. Van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'16, 2016.

[13] J. A. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques - Advances in Cryptology (EUROCRYPT)*, 2015.

[14] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP, 2017.

[15] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić. The Next 700 BFT Protocols. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2010.

[16] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *24th USENIX Security Symposium*, USENIX Security'15.

[17] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending Fast Payments in Bitcoin. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, 2012.

[18] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun. Misbehavior in Bitcoin: A Study of Double-Spending and Accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1), 2015.

[19] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium*, USENIX Security '16, 2016.

[20] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07.

[21] T. Lajoie-Mazenc, R. Ludinard, and E. Anceaume. Handling bitcoin conflicts through a glimpse of structure. In *32nd ACM Symposium on Applied Computing (SAC)*, 2017.

[22] L. Luu, V. Narayanan, K. Baweja, C. Zheng, S. Gilbert, and P. Saxena. SCP: a computationally-scalable Byzantine consensus protocol for blockchains. Technical report, Cryptology ePrint Archive, Report 2015/1168, 2015.

[23] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings for the International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[24] A. Miller and J. J. LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. http://bravenewcoin.com/assets/Whitepapers/Anonymous-Byzantine-Consensus-from-Moderately-Hard-Puzzles-A-Model-for-Bitcoin.pdf, 2014.

[25] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[26] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-addressable Network. *SIGCOMM Computer Communication Review*, 31(4), 2001.

[27] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[28] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016, 2016.

[29] Y. Sompolinsky and A. Zohar. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive*, 2013, 2013.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM SIGCOMM, 2001.

[31] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/Paper.pdf.