Permissionless Consensus based on Proof-of-Eligibility

Geoffrey Saunois, Frédérique Robin, Emmanuelle Anceaume, Bruno Sericola

Permissionless Consensus based on Proof-of-Eligibility

Geoffrey Saunois, Frédérique Robin
Inria, Univ. Rennes, CNRS, IRISA
Inria, Campus de Beaulieu, France
firstname.lastname@inria.fr

Emmanuelle Anceaume
CNRS, Univ. Rennes, Inria, IRISA
IRISA, Campus de Beaulieu, France
emmanuelle.anceaume@irisa.fr

Bruno Sericola
Inria, Univ. Rennes, CNRS, IRISA
Inria, Campus de Beaulieu, France
bruno.sericola@inria.fr

***Abstract*—We propose a consensus algorithm whose objective is to decide on the same union of proposed values, such that with high probability all the values proposed by the honest nodes belong to the decision. Our algorithm has been designed to cope with an asynchronous and permissionless system. By relying on a proof-of-eligibility, our algorithm is tolerant to an adversary capable of instantaneously corrupting entities. A straightforward application of our algorithm is the design of permissionless distributed ledgers.**

***Index Terms*—consensus algorithm, proof of eligibility, aggregation, asynchronous and permissionless environment, adaptive adversary.**

I. Introduction

Permissionless blockchains aim at achieving the impressive result of being a persistent, distributed, consistent and continuously growing log of transactions, publicly auditable and writable by anyone. Despite the openness of the environment and thus the inescapable presence of malicious behaviors, security and consistency of permissionless blockchains do not demand the presence of a trusted third party [11]. In the seminal blockchains, i.e., Bitcoin and Ethereum, this achievement results from the tight combination of two ingredients: a randomized election of the next block of transactions to be appended to the blockchain and a short latency broadcast primitive. While the latter one relies on the properties of peer-to-peer networks, the former one has so far been commonly implemented by solving proof-of-work (PoW). A PoW is a cryptographic puzzle that is provably secure against a large proportion of participants that may wish to disrupt the system, and allows to keep the rate at which blocks are created parametrizable and independent of the size of the system. Unfortunately, resilience of PoW-based solutions fundamentally relies on the massive use of computational resources, which is a real issue today. Numerous investigations have been devoted to find a secure alternative to PoW, but most of them either rely on the intensive use of a large quantity of physical resources (e.g. proof-of-space [2], proof-of-space/time [10]) or make compromises in their trust assumptions (e.g. proof-of-elapsed-time [8], delegated proof-of-stake [6]). In contrast, solutions based on proof-of-stake (PoS) seem to be a quite promising way to build secure and permissionless blockchains. Indeed, proof-of-stake relies on a limited but abstract resource, the crypto-currency, in such a way that the probability for a participant to create the next block of the blockchain is generally proportional to the fraction of currency owned by this participant. It is an elegant alternative in the sense that all the information needed to verify the legitimacy of a stakeholder to create a block (i.e., crypto-currency possession) is already stored in the blockchain. Finally, by being a sustainable alternative (creating a block requires a few number of operations), scalability concerns, exhibited by PoW-based solutions, should be a priori more tractable. However, an important condition for PoS-based blockchains to be secure is randomness: the creator(s) of the next block must be truly random, and the source of randomness must not be biaised by any adversarial strategy. So far, this has been addressed by two main approaches: chain-based consensus and block-wise Byzantine agreement. In the former approach (e.g. [3]), a snapshot of current users' status is periodically taken, from which the next sequence of leaders is computed. In the latter one (e.g. [5], [7]), a Byzantine agreement per block, relying on the properties of verifiable random cryptographic schemes, is achieved. While [5] is tolerant to a weakly adaptive adversary (i.e., a targeted attack needs a given amount of time before being effective), node corruptions in [7] are effective once decided by the adversary. Algorand [7] handles such a strong adaptive adversary by implementing proof-of-stake with a proof-of-eligibility. Proof-of-eligibility allows us to decide who is in charge of executing an action. When the eligibility is found out by each one individually, it is called a *cryptographic sortition scheme* [7]. Note that in contrast to cryptographic sortition, proof-of-work cannot be attributed to a single user which explains the development of mining pools. In Algorand, participation to the agreement protocol is ephemeral and depends on the amount of stake owned on user account. However, by relying on account balance, Algorand may not defend against an adversary that will observe, during the very first rounds of the consensus protocol, the IP address of users with significant savings account balance and launch a targeted attack on these users. Since these users have a higher probability to be involved in the subsequent rounds of the protocol than those with low savings, by preventing these wealthy users from participating to these rounds, the adversary may strongly jeopardize rounds progress, and consequently prevent termination of the Byzantine tolerant agreement protocol from holding.

The objective of this work is to go a step further by deeply relying on ephemeral but provable user identities. These identities allow any user to act under multiple identities without depending upon a trusted third party. We use the so-called *unspent transaction outputs (UTXOs)* to implement such identities. An UTXO can be roughly seen as a *single-use* account in the sense that once credited, an UTXO can be debited only once. An UTXO is uniquely characterized by a key pair (pk, sk) and its associated amount of stake. Each public key is related to the digital signature schema with the uniqueness property, which allows stakeholders to use the public keys (or a hash thereof) of their UTXOs as a reference to them, as demonstrated in the "Public Keys as Identities principle" of Chaum [4]. Any two UTXOs created by the same user are unlinkable. Hence user u is publicly known as pk_u if u owns the UTXO whose key pair is (pk_u, sk_u). An UTXO is created when it appears for the first time in the output set of a transaction, and once it is referenced in the input set of another transaction, it cannot be used anymore. Hence, by using UTXOs as user identities, a user owns as many verifiable identities as he wants, and can easily spread all its coins over multiple and unlinkable UTXOs. Since UTXOs are ephemeral, the number of users continuously varies with the activity of the system. Hence the rationale of using UTXOs in the proof-of-eligibility is to prevent the adversary from presuming on user wealthiness, and thus makes the above described targeted attack inefficient. This allows us to design a Byzantine tolerant consensus algorithm, which in presence of a rushing adversary, guarantees that all correct users of the permissionless system decide on the same union of proposed values with any high probability $1 - \varepsilon$, with $\varepsilon \in (0, 1)$, in a bounded number of rounds. To fit the context of blockchains, users propose their set of pending transactions as their input values, and the decision value is a union of pending transactions. When all the users propose the same set of pending transactions then with high probability the algorithm decides in three rounds, otherwise the number of rounds is upper bounded.

The remaining of the paper is orchestrated as follows. Section II presents the model of the system in terms of synchrony, communication, security, and user transactions. Section III specifies the problem addressed by this paper. Section IV describes the main tenets of our consensus algorithm. Section V presents an in-depth analysis of our algorithm.

II. System assumptions

a) Asynchronous and permissionless system: By permissionless we mean a distributed system in which *(i)* the number of participants for carrying out the protocol is not known before hand, is not even known during the course of the execution, and may change over time, *(ii)* the right to contribute and to participate is not controlled by a (trustworthy) third authority, i.e. we do not assume the presence of any public key infrastructure (PKI), and *(iii)* participants communicate over a weakly but reliable connected communication topology. We assume an asynchronous environment, that is our algorithm does not make any synchrony assumptions, i.e. does not assume any bounds on the time needed for a message to be received by its recipients, nor on the computation time of the processes, nor on the individual drifts of clocks.

b) Cryptographic functions: Users have access to basic cryptographic functions, including a cryptographic hash function $\mathfrak{h}$ of hash-value size h and an asymmetric signature scheme that allows a user to generate a public and secret key pairs (sk, pk), and compute a signature $\sigma_{sk,\mathfrak{h}}(d)$ of any message d. Function $\mathfrak{h}$ is modeled as a random oracle. Our algorithm relies on verifiable random functions (VRFs). A Verifiable Random Function (VRF) [9] is the public-key version of a keyed cryptographic hash. It is a pseudorandom function that provides a proof of its correct computation. Only the holder of the private key sk can compute the hash, but anyone with corresponding public key pk can verify the correctness of the hash.

c) Public Keys as Identities principle [4]: Users own some minimal amount of stake (i.e., money), which gives them the right to participate to the algorithm. As said in the introduction, we adopt a simplified version of what is commonly known as the Bitcoin Unspent Transaction Output (UTXO) model. In the following, when we say that *UTXO* (pk_u, sk_u) *is selected to perform some action* we mean *the user that owns this UTXO is selected to perform some action*. Symmetrically, when we say that *some user* u *executes some action* we mean *the owner of UTXO* $< pk_u, sk_u >$ *executes some action*.

d) Threat model: An adaptive adversary: We assume the presence of Byzantine (i.e. malicious) users which controls up to $p^{\mathcal{A}} < 1/3$ of the total amount of stake currently available in the system. This model, named the "stake threshold adversary" by Abraham and Malkhi [1], is an alternative to the common threshold adversary model. It bounds the total number of parties the adversary controls relative to the total population of the system. It is an extension (or modification) of the computational threshold adversary introduced by Bitcoin, which bounds the proportion of the computational power owned by parties. Byzantine users can deviate from the protocol. They are modeled by an adversary. The adversary can perfectly coordinates all malicious users. It can learn the messages sent by honest users (i.e. non malicious users), delay them, and then chooses messages sent by malicious ones. Further the adversary is adaptive: it can select at any time which users to corrupt in replacement of corrupted ones (i.e. corruptions are "moving"). The adversary is computationally bounded so that it can neither forge honest nodes' signatures nor break the hash function and the signature scheme. Finally, we assume that all users (honest and malicious) share an initial knowledge that we call *genesis block* which contains an initial arbitrary UTXO set. We assume this block also shares the same properties as regular blocks. How to setup the genesis block is out of the scope of this paper.

III. The addressed problem: the Merged-Consensus

While the overall goal of this work is to build a permissionless blockchain, in this paper we will concentrate on the design of the consensus algorithm whose objective is to decide on a union of valid transactions. We will prove that all the honest users of the asynchronous and permissionless system decide on the same union of transactions and that the decision is reached in a finite and bounded number of rounds with any high probability. We will also show that our algorithm is tolerant to a strongly adaptive adversary capable of instantaneously corrupting entities. Such an adversary is also called *rushing* adversary. Clearly, by sequentially invoking an instance of the consensus with a sequence number, this will allow the construction of a permissionless blockchain.

The task that honest users want to solve is the Merged-Consensus which is formalized by the following properties.

- **Termination:** Each honest user u eventually outputs one decision value dec_u;
- **Agreement:** For any honest users u and v that respectively decide dec_u and dec_v, then $dec_u = dec_v$;
- **Validity:** Any decision value dec contains at least the set of transactions proposed by an honest user.

IV. Main principles of our Merged-Consensus algorithm

The algorithm we propose to implement the Merged-Consensus specification in a permissionless system consists of several asynchronous rounds. By adopting the nice idea of user replacement [7], each asynchronous round r, $r \geq 0$, is run by a dynamically created committee whose members are selected among all the users (i.e. UTXO owners) of the system at round r. As said in the introduction, selection is achieved in a random, unpredictable, and non-interactive way with a cryptographic sortition mechanism. Correctness of our cryptographic sortition is ensured by guaranteeing that no obvious Byzantine strategy such as a concentration of the stake on a single manipulated UTXO or a massive sub-division of stake on a multitude of compromised UTXOs (Sybil attack), can bias user random selection. Mitigating the impact of the former strategy is achieved by assuming an upper bound on the amount of stake credited on UTXOs. Note that this is not a constraint since each user may create an arbitrary large number of UTXOs and the number of transaction outputs is not upper bounded. To prevent an adaptive adversary from manipulating committee members during round r, $r \geq 0$, the action of each committee member is limited to a unique step of computation followed by a unique step of communication. Hence, if the adversary eavesdrops a message from a committee member u, it is too late for him to manipulate u since u will not execute any more steps in round r, and possibly in any other rounds of the algorithm. Recall that users are "short-lived" (a user is alive as long as its UTXO has not been spent).

Rule 1. *Any UTXO stake amount is bounded by constant U.*

Cryptographic sortition exports two functions: the DRAW and VERIFYDRAW functions. Function DRAW is a private function which allows users *(i)* to determine by themselves whether they are selected as a committee member of a given round r of the consensus algorithm, and *(ii)* to provide later and if necessary a proof of soundness of this selection. The DRAW function when invoked by UTXO $< sk_u, pk_u >$, has five arguments: the secret key sk_u of the UTXO, a seed *seed* from which comes the randomness of the sortition, the expected number of stake τ selected by the current sortition, the amount of stake w_u credited on UTXO $< sk_u, pk_u >$, and the total current amount of currency units W owned by all the users of the system. Computation of the seed is based on the most recent pieces of information known by every user. In our case, this is the hash of the last created block, that is the hash of the decision value decided by the preceding instance of the Merged-Consensus. The knowledge of W is obtained by successively reading all the blocks of the blockchain. Function DRAW is made of two steps, first a call to a VRF function which calculates a random and verifiable number h_u, and second a weighted random sampling seeded with h_u. The weighted random sampling computes the voting weight $vote_u$ of UTXO $< sk_u, pk_u >$ during round r according to the amount of stake credited on this UTXO. This voting weight is used by UTXO $< sk_u, pk_u >$ during round r of the consensus algorithm. If $vote_u$ is equal to 0, user u cannot participate to the r-th round of the consensus algorithm. Otherwise, $vote_u$ gives u the right to belong to the committee of round r, and represents the weight of u's vote during round r. Specifically, let $vote_u$ be the random variable representing the voting weight of UTXO $< sk_u, pk_u >$ as computed by function DRAW. Random variable $vote_u$ has a binomial distribution $\mathcal{B}(w_u, p)$ where probability p is equal to τ/W. The probability that UTXO $< sk_u, pk_u >$ is selected is thus $1-(1-\frac{\tau}{W})^{w_u}$. These random variables being independent, whatever the sub-division $w_1 + w_2 = w$ of w, the distribution of the weight associated with w is the same as the sum of the weight associated with w_1 and w_2: $\mathcal{B}(w, p) = \mathcal{B}(w_1, p) + \mathcal{B}(w_2, p)$. This guarantees that an adversary has no advantage in launching a Sybil attack: an adversary can create as many accounts as he wants, what will influence the probability of winning is the total amount of stake, not the number of accounts.

Function VERIFYDRAW is a public function that allows each user to verify the legitimacy of UTXO $< sk_u, pk_u >$ to get a voting weight *vote*. This function is similar to DRAW except that it is called with the public key pk_u of the UTXO and proof π_u.

Any honest [1] user of the system invokes the Merged-Consensus algorithm (invocation of Propose() with the set of locally pending valid transactions [2]), but participation to round r, $r \geq 0$, depends on the outcome of the Draw function.

[1] Recall that we cannot compel Byzantine user to follow the protocol.

[2] We can legitimately talk about the validity of a transaction. A transaction is valid if none of its referenced UTXOs have already been spent in some transactions belonging to a previous decision value, i.e., in a block belonging to the blockchain.

The following parameters are public knowledge: the *seed* of the current instance of the consensus, the expected number of stake τ selected by the lottery, the total amount of stake W in the system, $\mu \in (0,1)$, and $\lambda \in (0,1)$ whose values are analyzed in the full version of the paper. From above, in expectation, for each round r, τ stakes are randomly selected among the W stakes of the system.

The Merged-Consensus algorithm is made of a succession of rounds $r = 0, 1, \ldots$ During round $r = 0$, committee members (i.e., those that successfully passed the DRAW function for round $r = 0$) propose their sets of pending transactions $\mathcal{T}$ by broadcasting the message $m_u = \langle \perp, 0, (pk_u, \pi_u), \mathcal{T}_u, \emptyset, \text{vote}_u \rangle$ to all the users. In rounds $r > 0$, committee members collect and merge proposed transactions. Each round $r > 0$ is made of the following two steps: collect of broadcast messages, and construction of the final set of transactions, together with the proof that those sets of transactions have been initially broadcast in round $r = 0$. The collect of broadcast messages step consists for committee members of round r in collecting sufficiently many messages broadcast during round $r - 1$. By sufficiently many, we mean that the total number of votes of the senders of these messages must be larger than $\mu\tau$. Conditions on the value of $\mu\tau$ are provided in the full version of the paper. To guarantee the convergence on a unique set of transactions with high probability, we first need to guarantee that messages only contain transactions that have been initially sent in round $r = 0$, and second we need to prevent the adversary from withholding the transactions it sent during round $r = 0$, and then progressively make honest committee members discover them during subsequent rounds. The first case is guaranteed by providing in message m_u a proof (data structure $\mathcal{M}_u$) asserting that all collected transactions have been initially proposed by the committee members of the previous round, and thus by induction by those of round $r = 0$. The second case is handled in round $r = 2$ and is detailed below. The second step consists in building the final set of transactions. Once valid messages have been collected, any committee member u tries to build the final set of transactions. Specifically, if $r = 1$, then the preliminary final set of transactions $\mathcal{T}_u$ contains the union of all the transactions received by u during the round and variable $IsFinalCnt = 0$. It may happen that all the committee members of round $r = 0$ broadcast the same set of transactions, in which case the final set of transactions will not evolve anymore (Boolean $IsFinal$ is set to true). Set $\mathcal{T}_u$ together with the set of collected messages $\mathcal{M}_u$ (which acts as a proof for $\mathcal{T}_u$) is broadcast to all the users (Recall that by the proof-of-eligibility property, users determine by themselves whether they are selected as a committee member of a given round r which explains why any message must be broadcast to all users to be received by the intended recipients). Round $r = 2$ is particular and its objective is twofold: achieving faster convergence to the final set of transactions and preventing the adversary from withholding the transactions it sent during round $r = 0$, and then progressively making honest committee members discover them during subsequent rounds. This is achieved by keeping in the final set of transactions $\mathcal{T}_u$ only transactions that have received sufficiently many votes (i.e. $\lambda\mu\tau$). Parameter $\lambda \in (0,1)$ can be seen as a broadcast factor, whose value is analyzed in the full version of the paper. As for round $r = 1$, set T_u together with the set of collected messages $\mathcal{M}_u$ (which acts as a proof for $\mathcal{T}_u$) is broadcast to all the users for the next round of the algorithm. Subsequent rounds $r > 2$ are run until convergence to the same set of transactions is reached. Convergence to the same final transaction occurs when some user u receives sufficiently messages from the previous round where $IsFinal$ argument is set to TRUE. Again, by sufficiently, we mean that the cumulative votes of these messages is larger than $\mu\tau$.

V. ANALYSIS

Theorem 1 (Agreement)**.** *For any honest users u and v that respectively decide* dec_u *and* dec_v*, then* $dec_u = dec_v$ *with probability* $1 - \varepsilon$*.*

Theorem 2 (Validity)**.** *Any transaction* $\mathfrak{t}$ *belonging to an honest set broadcast at round* 0 *will appear into the final decision with probability* $1 - \varepsilon$*.*

Theorem 3 (Termination)**.** *Our algorithm completes in a finite number of rounds r, and r is upper bounded by r_{max} with $r_{max} = 3\,(\lfloor \alpha/\lambda \rfloor + 1)$ with probability $1 - \varepsilon$.*

All the proofs are provided in the full version of the paper.

REFERENCES

[1] Ittai Abraham and Dahlia Malkhi. The blockchain consensus layer and bft. *Bulletin of the European Association for Theoretical Computer Science*, (123), 2017.

[2] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi. Proofs of Space: When Space Is of the Essence. In *International Conference on Security and Cryptography for Networks (SCN)*, 2014.

[3] D. Bernardo, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *International Conference on the Theory and Applications of Cryptographic (EUROCRYPT)*, 2018.

[4] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1988.

[5] A. Durand, E. Anceaume, and R. Ludinard. Stakecube: Combining sharding and proof-of-stake to build fork-free secure permissionless distributed ledgers. In *Proceedings of the International Conference on Networked Systems (NETYS)*, 2019.

[6] EOS.IO. Technical white paper v2, 2019. Accessed: 2019-03-10.

[7] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, SOSP 17, page 5168, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Intel. Hyperledger Sawtooth description, 2019. Accessed: 2019-03-10.

[9] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science*, 1999.

[10] Tal Moran and Ilan Orlov. Proofs of space-time and rational proofs of storage. In *Cryptology ePrint Archive, Report 2016/035*, 2016.

[11] S.Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *www.bitcoin.org*, 2008.