Distributed Trust, a Blockchain Election Scheme

Antonio M. Larriba[1], Aleix Cerdà i Cucó[2], José M. Sempere[1], and Damián López[1]

[1] VRAIN - Valencian Research Institute for Artificial Intelligence
Universitat Politècnica de València
{anlarflo@dsic.upv.es,jsempere,dlopez}@dsic.upv.es
[2] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
alcercu@inf.upv.es

October 29, 2020

Abstract

Voting systems are as useful as people are willing to use them. Though many electronic election schemes have been proposed through the years, and some real case scenarios have been tested, people still do not trust electronic voting. Voting is not only about technological challenges but also about credibility, therefore we propose a voting system focused on trust. We introduce political parties as active partners in the elections as a mechanism to encourage more traditional electors to participate. The system we here propose preserves elector's privacy, it operates publicly through a blockchain and it is auditable by third parties.

Keywords: Electronic Vote; Distributed Authority; Blockchain; Proof of Authority; Monero.

1 Introduction

Voting is a crucial part of democratic societies. There have been numerous approaches to improve and to automatize the voting process. First advancements were based on modification of the ballots (e.g. [38, 40]) and mechanical and optical readers to speed up the vote count. The advances of cryptography were key in the development of electronic voting methods. On the one hand, e-voting provided not only more accessible and faster ways to count votes, but also new properties unavailable to traditional voting systems such as: remote voting, universal verifiability and auditability.

On the other hand, e-voting also raised new security and privacy concerns. To address these concerns, e-voting makes use of cryptographic constructs. Therefore, most electronic voting protocols can be categorized in four large groups, defined by their cryptographic composition: those systems based on blind signatures (e.g. [10, 12]), based on mixnets (e.g. [11, 13]), based on homomorphic cryptography (e.g. [6, 16]) and those based on ring signatures (e.g. [15, 39]).

Usually, the later methods employ a public bulletin board to display the election process and the results. They seek a way of publishing the voting information to later allow to verify and audit the election. With the apparition of Bitcoin [44], Blockchain technology proved itself as an effective, distributed and decentralized public ledger [7, 36]. Since then, various voting protocols based on blockchain have appeared (e.g. [3, 29]). Typically, the voting process is embedded into the blockchain, which acts as the voting interface and a decentralized public bulletin board as well.

One of the goals of e-voting is to incentive participation. However, despite it provides remote and, in some cases, multi-platform access, people do not trust e-voting. Because it is difficult to trust in a system based on abstract mathematical results that people do not fully understand. In our opinion, e-voting should not be all about technological issues, but also about trust and about actively engaging all the parts in the democratic process. For this reason, we present an e-voting protocol in which traditional parties, to whom people trust to some extend, take a dynamic and responsible side. We introduce a new voting scheme inspired by Monero's blockchain and ring signatures. In our approach, each party is given partial power and full accountability in reaching a common interest despite their antagonistic nature. The contributions of this paper are two-fold: we develop a new voting scheme focused on trust, which is scalable, secure and preserves the anonymity and privacy of the electors, and we also present a new and cheaper consensus algorithm which we name Proof of Authoritys, based on the distribution of trust on the adversarial interests of the involved parties.

The rest of the paper is structured as follows: Section 2 reviews and describes significant works in the literature, Section 3 provides a brief background on the key concepts used on the protocol, Section 4 exposes and describes our proposal and Section 6 studies and analyses the properties of our election scheme. Finally, Section 7 reviews the contributions and conclusions of our work.

2 Related Work

In this section, we review relevant papers in the literature. We distinguish between three categories tightly related with our proposal. We inspect the usual problems these approaches face and assert the new contributions our

scheme brings.

2.1 Ring signature based voting systems

Exploring the security requirements of a democratic society, Salazar et al. proposed in [43] a voting system based on short-linkable ring signatures [48]. These signatures allow for a linking tag that permits to relate votes from the same elector without revealing her identity. The aim of the authors was to reduce the number third-parties involved in the voting process. Only a certification authority, responsible of issuing the keys and validating the certificates, and a recount authority are needed. Ring signatures grant signer ambiguity, and linking tags act as receipt and allow the removal of duplicates. Thus, only the public key of the recount authority is used for encryption, which gives too much sway to a single authority. The protocol was later implemented in [47], which provided a modular implementation in a multi-platform environment: a desktop client, an android native app and a Firefox extension were developed to boost usability and engagement in the voting process.

In [15], Chen et al. propose a electronic voting system based on a modified version of linkable ring signatures. In this modification, only designated verifiers can check the validity of the ring signature. Ring signatures are encrypted with the verifier's public key. Therefore, a designated verifier cannot convince a third party of the integrity of a signature without revealing its private key. This prevents designated verifiers from broadcasting information about a private signature. The system uses a (k, l)-threshold sharing scheme to generate and distribute the secrecy of the private key [24] between l tallying authorities. The public key is made public before the election starts. For an elector to cast a vote, she has to select the direction of her vote and encrypts it using the public key of the election. Then, the ballot is sent through an anonymous channel to an administrator. The administrator signs the ballot, adds a timestamp and returns the ballot to the elector. If the signature returned by the administrator is valid, the elector crafts the linked ring signature, encrypts it with the public key of a tallying authority of her choice and publishes it on the bulletin board. After the election, at least k tallying authorities cooperate to recover the private key. When the private key of the election is recovered, each tallying authority is responsible for checking the validity of the received ring signatures and publishing the votes.

2.2 Blockchain based voting systems

In [35], Noizat proposes a voting system based on blockchain and Merkle trees [31] for elector's verification. Each elector uses three public keys: a key from the candidate she decided to vote for (KeyC); a key from the

election organizers (KeyA), and a key from the voting application (KeyB). The system provides privacy through 2-of-3 multi-signature addresses [42]. To prepare the ballot, the elector prepares a 2-of-3 multi-signature address and a transaction. There is no way to guess the elector nor the candidate from a multi-signature address without knowing all three public keys and knowing to whom they belong. To verify the ballot is counted as intended, the elector can use block explorers to check his vote has been confirmed.

Lee describes in his proposal [29] a blockchain voting system. For an elector to vote for a candidate, she has to make a transaction to a candidate's address. To do so, the elector first needs to register as a valid elector. To avoid election organizers from knowing who voted for which candidate, the registration process is decoupled in two organizations (the registration organization itself and a trusted third party (TTP)). To cast a vote, the elector sends a hash of her secret to the registration organization and the TTP. The TTP asks the organization if that hash is linked to a valid elector. If the answer is positive, the TTP asks the elector for the secret and checks if it matches the one received. If they match the elector is registered as a valid elector. Then, the elector can make the transaction to vote for the candidate of her choice. At the end, transactions are inspected and checked against a permutation of the TTP verified elector's list. Multiple voting, and unverified electors are removed from the tally.

Ayed describes in [3] a simple blockchain based voting protocol. In his system, each candidate has its own blockchain and every block, but the first, represents a vote for the candidate. The first block of each chain contains information about the candidate. To vote, an elector must identify herself as a valid elector using her address and some private information. Then, she can decide the direction of her vote. When the vote is decided, some private elector's information is hashed alongside the hash of the previous block to constitute the hash of the new block. Finally, the new block is added to the correspondent blockchain. Ayed's proposal addresses a few of the issues of centralized voting systems, but it fails to properly define some of the crucial parts of the protocol: registration is not supported and identification is assumed to be solved. Also, the encryption and identification is managed by a centralized interface.

Tarasov and Tewari introduce in [46] a Zcash based voting protocol. Zcash [8] emerged as a Bitcoin fork with privacy as concern. Zcash supports two different kind of addresses: *t-addresses, which work like regular pseudonymous Bitcoin addresses and allow transparent transactions, and* z-addresses, which preserve the anonymity and privacy of the transactions. Private-anonymous transactions are based on special zero-knowledge proofs, zk-SNARKS [8]. These proofs permit the interchange of the secret values required to establish a private transaction between sender and receiver. After registration, the elector can make a transaction to the *z-address of the desired candidate. The elector is required to provide also a valid* t-address.

Thus, the direction of the vote is private, but the transaction is still public. When the voting phase ends, candidates send all the vote tokens to a central pool where the final tally is computed. This requires some trust on the system and the candidates, because the candidates are expected to send the tokens and collaborate, and a malicious candidate could interfere in the voting process.

Yang et al. propose in [53] a blockchain voting protocol for range voting, in which each candidate receives a score and the candidate with the highest score wins the election. To preserve elector's privacy they propose a novel encryption scheme based on El Gamal [19] and group-based encryption. Each vote is encrypted by using the public key of the elector and the public keys of the candidates. This way, even at the end of the election, when the candidates release their secret keys, the individual votes remain secret. To compute the final tally, they take advantage of the homomorphic properties of El Gamal to compute the final score without decrypting individual votes. Then, the final score is decrypted by the candidates. Each transaction contains a vote, and each vote is formed by a set of scores, one for each candidate. The only drawback oh this approach is that each individual score needs to be double encrypted, and accompanied by a zero-knowledge proof, and a partial-knowledge proof to ensure is a valid score from a valid voter. This affects both the time and spatial complexity of the system. However, the authors are able to provide a performance analysis that proves the validity of their election scheme.

Gao et al. present in [23] an anti-quantum e-voting protocol based on blockchain technology. To achieve quantum computing resistance, they base their method on code-based cryptography [33] (a NP complete problem) instead of using traditional public key cryptography based on the difficulty of number theory. Their protocol is equipped with an audit function, based on public key certificates [2, 26], that allows to detect fraudulent voters. To handle the certificates, the authors introduce the figure of the regulator, that, despite he does not participate in the election, he has the authority to revoke voter's privacy. The authors report a complete and exhaustive computational time analysis of their protocol.

FollowMyVote [1] and Bitcongress [41] are online projects of blockchain based e-voting. While they are not backed by research papers, they are pioneer on providing remote blockchain based voting to the public, being open source and supplying a real working service as an alternative to traditional voting. FollowMyVote is based on elliptic curves and blind signatures [12], and requires a central authority to deal with elector registration and ballot lending. BitCongress is another decentralized e-voting proposal, it is based on digital signatures and smart contracts. It is also compatible with distributed legislation which is enforced through the blockchain. It requires a timestamp server for the system to work. The developers provide a front-end wallet called AXIOMITY that handles the working modes of BitCongress.

2.3 Ring Signatures & blockchain based voting systems

In [51], Wu develops a voting system based on ring signatures [39] and Bitcoin's blockchain. There are two authorities on the protocol: a Registration Authority (RA) and an Election Authority (EA). It is assumed these authorities are trustworthy and will not share information. Before the election starts, the EA is responsible for generating and managing a public pool of Bitcoin addresses. Since Bitcoin is not private, but anonymous, a two-phase registration is carried out by the RA. This process decouples the elector identity from the Bitcoin address and preserves elector's privacy. As a result of the registration process, the elector gets her public key certified as a valid key. For an elector to craft a ballot, she must perform a ring signature, using her private key and a list of public keys, on her desired vote. This results in a signer ambiguous signature that makes impossible to link the vote to her public key. Once the ballot is crafted, she selects a Bitcoin address from the pre-computed pool of addresses. The EA will provide the elector with the associated private key to that address. Then, the elector can send the ballot as a transaction from her own address to the EA's address, with the ballot encoded in the transaction itself. The EA is responsible of retrieving all the ballots and verify the integrity of ring signatures to compute the final tally. When the election is over, the list of public keys of electors is released to the public and everyone can verify the correctness of the tally. Since the ring signature used in this method allows double voting, EA checks the transactions coming from the same address. For each address, only the latest transaction will be considered valid. As a drawback of this method, it is to note the great sway of the authorities in the voting process. As a result, one of the requirements of the protocol is that authorities comply with the desired conduct. Wu also developed a complete implementation of the system on Bitcoin's testnet, with a great definition of classes and use cases.

Wei et al. propose in [50] an elegant voting system based on Ethereum smart contracts and one-time ring signatures [49]. A transaction is considered a vote, electors make a transaction to their selected candidate. The privacy of the elector is protected by ring signatures, which also prevent double voting. To keep the election fair and to avoid the leak of information until the tally phase, the electors should consider the stealth addresses of the candidates to cast the vote. This way, until the stealth addresses are revealed, votes are kept secret. To expose the stealth addresses, key managers are needed. Key managers share the first private key of a candidate through a Diffie-Hellman interchange. Key managers store some ETH in a deposit previous to the election, to recover it they must open their secret on the tally phase. Once the stealth addresses are exposed, the whole election process is public and auditable on the Ethereum blockchain. All the requirements of the protocol are contained in a smart contract.

3 Background

In this section we present a brief and concise review of the key concepts and protocols needed to define our proposal. We also introduce the notation that will be used along the rest of the work.

3.1 Notation

In this work, we use both RSA cryptosystem and Elliptic Curve Cryptography (ECC). The former is used by the electors to encrypt their vote and by the parties to sign the transactions. The latter is used for generating ring signatures in the context of a blockchain in order to allow correct and anonymous identification of electors. We assume some familiarity of the reader with these notions. We recommend [30] to the not accustomed reader. Now, we summarize the notation that will be used the rest of the paper.

- Modular product and exponentiation operations are expressed as $ab \bmod n$ or $c^d \bmod n$.
- Concatenation of binary sequences is expressed as $a \| b$.
- g represents a generator in a given finite group.
- A finite group defined by a prime number q is represented as $\mathbb{F}_q$.
- G represents the generator or base point of an elliptic curve E.
- In ECC, private keys are expressed in lowercase a, while public keys are expressed in uppercase A. In such way that $A = aG$.
- H_s is a hash function which maps a binary sequence into an element of a finite group $\{0,1\}^* \to \mathbb{F}_q$.
- H_p is a hash function which maps an element of a finite group into an elliptic curve point. $\mathbb{F}_q \to E$.

3.2 One Time Public Keys

Monero is a cryptocurrency based on the cryptographic protocol described in [49] by Nicolas Van Saberhagen. Its main focus is to achieve private and untraceable transactions over a public ledger. Here, we present a brief description of how Monero achieves those goals and an example of how an untraceable transaction works. For further details, we recommend the review done by Koe et al. in [28] to the interested reader.

In order to anonymize the sender, ring signatures are used to sign transactions with a group of keys, thus giving ambiguity to the sender. To anonymize the receiver, they use stealth addresses. One Time Public Keys

($OTPK$s) are derived from the receiver's stealth address, and allow the sender to use a new public key for each transaction (using a Diffie-Hellman-like [18] exchange). Sending transactions using these keys preserves the anonymity of the receiver. The receiver, and only the receiver, is able to recover the private key of the stealth address and use its funds.

In our proposal, we take advantage of Monero's $OTPK$s in order to allow elector identification. Thus, we here present a brief summary of how $OTPK$s are derived. Each user on the blockchain has two public keys (A, B). For any sender to make a transaction, he needs to compute the $OTPK$ that will be used in the transaction using the two public keys of the receiver. He also selects a random number r.

$$OTPK = H_s(rA)G + B$$

The derived $OTPK$ is used as the public address of the transaction. Note that R, being rG, is also added in the transaction information. Once the transaction is done, the receiver will be able to recover the private key x associated to the $OTPK$ using his own private keys (a, b):

$$x = H_s(aR) + b,$$

such that:

$$OTPK = xG,$$

indeed:

$$\begin{aligned} H_s(rA)G + B &= (H_s(aR) + b)G \\ (H_s(rA) + b)G &= (H_s(aR) + b)G \\ (H_s(raG) + b)G &= (H_s(arG) + b)G \\ (H_s(aR) + b)G &= (H_s(aR) + b)G \end{aligned}$$

3.3 Ring Signatures

Ring signatures are a special kind of cryptographic signature. They receive this name because they are created using a ring structure, where the public keys of several peers (electors in this framework) are used in order to provide a signer-ambiguous signature. No information about the signer is revealed, only his membership to certain group. Any person involved in a ring signature could be the signer.

Multiple versions of ring signatures exist. Originally, they were proposed by Chaum in [14] as group signatures. They were limited because a group coordinator was required to set up the signing scheme. To overcome this issue, Rivest introduced in [39] the first ring signatures. They provided unconditional signer ambiguity without group coordinator. Any user can define a set of possible signers, including theirselves, and sign the message

using their public and secret key and the others' public keys. Neither coordinator, nor the permission (or even the knowledge) of the owners of the third party public keys are required.

Ring signatures are a great and solid solution to achieve the anonymity needed in the voting protocols. Nevertheless, since it is impossible to know who is the actual signer, nothing would prevent the use of another ring to vote. In order to prevent that, we will use the Ring Signature Confidential Transaction algorithm described in [34, 49]. This algorithm is the result of combining the modifications for reducing space consumption described by Adam Back [4] and the introduction of Key Images.

Key images are a public commitment of the signer's private and public key, they do not reveal information about the signer's private key. As a result, we can anonymously link the private key of the signer, independently of the public keys on the ring, to the signature. This allows to prevent the use of the same key to sign two different rings, which would imply double spending the same resource.

Algorithm 1 details the steps an elector should perform to apply a ring signature to her vote. Since the ring signature generation has some random coefficients involved and depends on the signer's private key, votes in the same direction signed with the same ring of public keys are still different. Algorithm 2 specifies the process the authorities must follow in order to verify the correctness of a given signature.

Algorithm 1 Ring Signature Generation

Require: $N \leftarrow$ Number of keys that will participate in the ring.
$P = \{P_1, P_2, \ldots, P_N\} \leftarrow$ List of public keys that will be linked to the vote.
$s \leftarrow$ The index in the list P where the public key of the signer is located.
$x \leftarrow$ Private key associated to signer's public key (P_s).
$m \leftarrow$ Message to sign.

Let r be a list of random numbers empty in the s index.
Let α be a random number.
Let L, R, c be empty lists.
$K \leftarrow xH_p(P_s)$ //Key image. Note that $H_p(P_s)$ is a point of the curve.
$L_s \leftarrow \alpha G$
$R_s \leftarrow \alpha H_p(P_s)$
$c_{(s+1) \ mod \ N} \leftarrow H_s(m, L_s, R_s)$
$i \leftarrow (s+1) \ mod \ N$
while $i \neq s$ **do**
    $L_i \leftarrow r_i G + c_i P_i$
    $R_i \leftarrow r_i H_p(P_i) + c_i K$
    $c_{(i+1) \ mod \ N} \leftarrow H_s(m, L_i, R_i)$
    $i \leftarrow (i+1) \ mod \ N$
end while
$r_s \leftarrow \alpha - c_s x$
return (P, K, c_0, r)

Algorithm 2 Ring Signature Validation

Require: $N \leftarrow$ Number of keys in the ring signature.
$\{P_1, P_2, \ldots, P_N\} \leftarrow$ List of public keys linked to the vote.
$K \leftarrow$ key image associated to the signature.
$c_0 \leftarrow$ Seed of the ring that allows the start of the validation algorithm.
$\{r_1, r_2, \ldots, r_N\} \leftarrow$ List of random numbers used in the sign generation phase.
$ListK \leftarrow$ The list of all the key images that have already been used in valid signatures.
$m \leftarrow$ Signed message.

if K in $ListK$ **then**
  return $False$
end if
Let L', R', c' be empty lists.
$L'_0 \leftarrow r_0 G + c_0 P_0$
$R'_0 \leftarrow r_0 H_p(P_0) + c_0 K$
$c'_1 \leftarrow H_s(m, L'_0, R'_0)$
while $i < N$ **do**
  $L'_i \leftarrow r_i G + c'_i P_i$
  $R'_i \leftarrow r_i H_p(P_i) + c'_i K$
  $c'_{(i+1) \ mod \ N} \leftarrow H_s(m, L'_i, R'_i)$
  $i \leftarrow (i + 1)$
end while
if $c'_0 == c_0$ **then**
  return $True$
else
  return $False$
end if

4 Our Proposal

We devote this section to explain our voting scheme proposal. We present a fully auditable, public and distributed voting system based on Blockchain technologies. All the information is registered into the blockchain, a vote itself is a transaction, and all the blocks are free to be publicly consulted. As a mechanism to engage the elector in the election process, we give political parties a special role in our system: they act like miners, listening and processing transactions. Therefore, any political party willing to participate must supply computing power. Electors do not need to trust these parties, all the voting process is public and auditable, privacy and anonymity are provided. In our opinion, people feel confident if they have someone to blame if something goes wrong. People who feel reluctant about new electronic voting systems, may trust a scheme in which traditional factions take an

active part and are accountable for their actions.

We employ a Proof of Authority (PoA) consensus algorithm [5, 52], as an alternative to running a traditional Proof of Work. This variation reduces the computational cost required to run a functional blockchain. PoA is a permissioned blockchain where only some certified participants are allowed to carry out certain actions. In our system, parties are responsible for listening to transactions, verify the signature of the encrypted vote and make sure the vote is correctly written on the blockchain. Parties are the only partners with write access to the blockchain, while the rest of participants have read-only access. PoA can be seen as a variation of Proof of Stake [52], but instead of staking monetary tokens, parties stake their identity. Their reputation will be discredited if they misbehave. To our knowledge, no other adaptation of PoA to anonymous electronic voting has been presented.

We consider the process as five different stages: election setup, registration, vote casting, vote processing and tallying. The algorithm is discussed in detail in the next sections.

4.1 Election Setup

Before the election, parties must collaborate to generate the parameters of the election. To encrypt the votes electors send to the parties, we employ RSA, therefore, parties must generate the RSA parameters: the public modulo n, the public *verification key* v *and the private* signature key s. Since giving the private key s to a single party would result in giving up too much power, l parties apply the threshold RSA key generation protocol proposed by Damgård et al. in [17]. This protocol relies on the work of Frankel et al. [21] to remove the necessity of a trusted dealer. It introduces a (k, l)-threshold RSA sharing scheme in which the parameters are computed in a distributed way and the secret key is generated in l fragments. To recover the secret key, any subset of at least k parties can collaborate to find the original secret s. Even if some parties are corrupt, they would need to collaborate with at least k parties. The same applies to honest parties, only k are needed to recover the private key s. Now, we proceed to give an overview of the aforementioned key generation protocol.

In RSA, modulo n is computed as the product of two large primes p and q. To distribute the trust, p and q are computed as the sum of different shares chosen by the parties: $n = (p_1+p_2+\ldots+p_i+\ldots+p_l)(q_1+q_2+\ldots+q_i+\ldots+q_l)$. To avoid from parties maliciously changing its share, they are required to publish a commitment [37] during the process. The distributed computation of n is carried out using a variant of Shamir's secret sharing [45]. This variant [20], also employs random polynomials in which the independent term is the secret, but it allows the computation of n to work outside prime fields.

Once the modulus is agreed, parties can jointly derive the public exponent v in a similar way, a distributed test division is required to test the

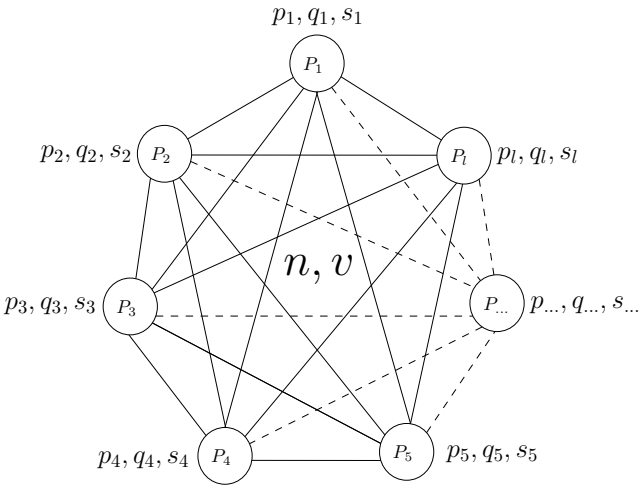


Figure 1: Parties collaborate to generate RSA parameters. Generation of n and v is made of parties additive shares p_i and q_i. Each party ends with a share of the private key s_i.

primality of the candidates. When computing s, to prevent from revealing the shares of the polynomial, the shares v_i are used as an exponent to compute: g^{v_i} , being g a generator of the group defined by v. This way each party gets an additive share of the private key s_i. Then, they proceed to construct a (k, l)-threshold polynomial of s. Thus, a distributed generation of RSA, in which parties only have partial information of the private key s is achieved. To decrypt or sign a message, parties must collaborate to retrieve the shared private key. The collaboration process can be seen as a graph (see Figure 1), in which each node represents a party, and each edge represents the interchange of messages. The retrieving protocol is detailed in the following sections.

We require all the traffic of messages as well as the commitments related to any vote to be published as transactions on the blockchain. Hence, the blockchain comprehends all the information related to the election. RSA parameters n and v are also released to the blockchain. Apart from the distributed key, each party has its own personal pair of public/private keys. They are used to sign all the transactions they make. All this information is published on the blockchain as the first block.

4.2 Registration

Electors must register before the election starts. For this purpose, a local administration (e.g: city hall) defines the census of potential electors. It will also store and manage elector's keys as well as generate $OTPK$s for each elector. The administration will declare the process through which the

electors will identify and register their public keys, and the termination of the registration term.

Any elector willing to participate will generate a two pairs of elliptic curve keys (a, A),(b, B). Then, the elector will follow the process specified by the administration, and, in case of correctly identifying, the administration will link their public keys (A, B) to their identity.

Once the elector registration term ends, the administration computes a $OTPK$ per elector using their public keys and a different random number r per $OTPK$. After this, the administration sends the information to the parties. Parties send a transaction through the blockchain, making public a list with the $OTPK$s of each public key pair and the associated R such that $R = rG$. The owners of a given $OTPK$ either will be able to recover the corresponding private key computing $H_s(aR) + b$, or can be notified by the administration in order to save computation time. Administrations can cooperate to compute and group $OTPK$s per districts in structure transactions and to simplify the search for the final elector.

The same transaction also includes basic configuration information for the voting such as which are the options in the voting, the codification of the vote. This second transaction contains all the remaining public parameters of the election. Figure 2 illustrates the process electors carry out in order to register themselves.

4.3 Vote Casting

Once the electors have decided what to vote for, they must follow three steps to cast a valid vote.

First they must encrypt it to protect its direction until the election is over. To do so, they read the public key v and the modulo n form the blockchain and apply a modular exponentiation to encrypt it using RSA. Before encrypting the vote, the elector must select a fixed length random $mask$ of her choice to concatenate to the vote. Otherwise, all the votes in the same direction will result in the same encryption. The elector will obtain an encrypted value such as $evote = (vote \| mask)^v \ mod \ n$.

Next, the elector must sign the vote to prove that it has been casted by an eligible elector. To perform the ring signature generation procedure described in Algorithm 1, the electors randomly take N $OTPK$s (including their own) from the list of public keys. The elector obtains a ballot conformed by the encrypted vote and its signature: $ballot = \{evote, \sigma(evote) = (P, K, c_0, r)\}$. Note that the number of public keys in the ring N is tightly connected with the ambiguity of the signer. N acts as a security parameter, larger values imply more ambiguity, thus more privacy. However, it makes the signature slower and more expensive in terms of computation. It might be beneficial to establish a fixed or a minimum value for this number.

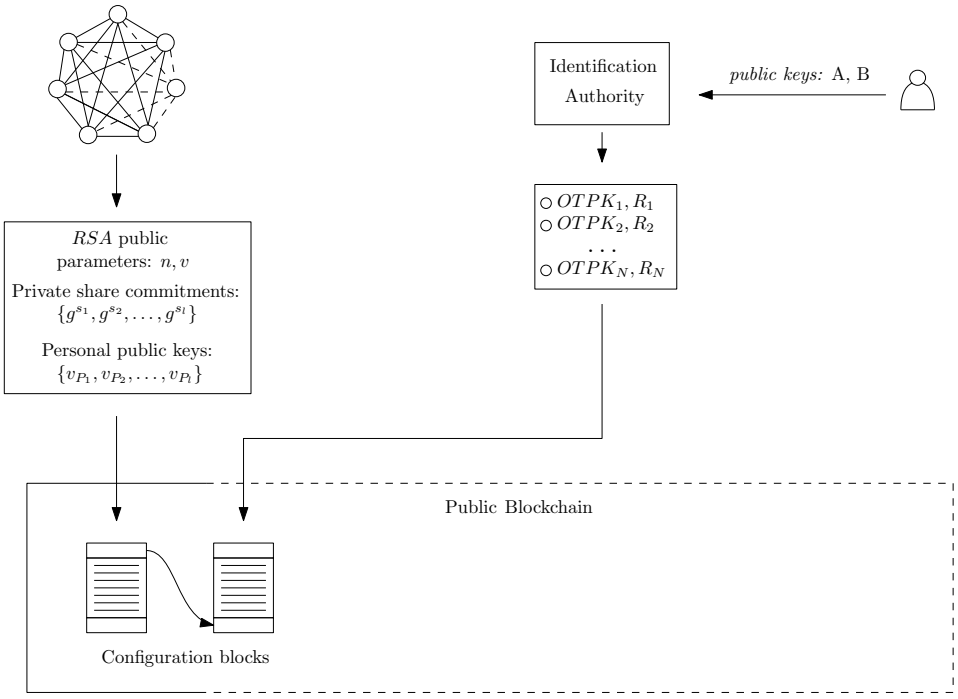


Figure 2: Electors register their public keys in a local identification authority. The identification authority computes the $OTPK$s. All the public parameters of the election are added as the first blocks of the blockchain.

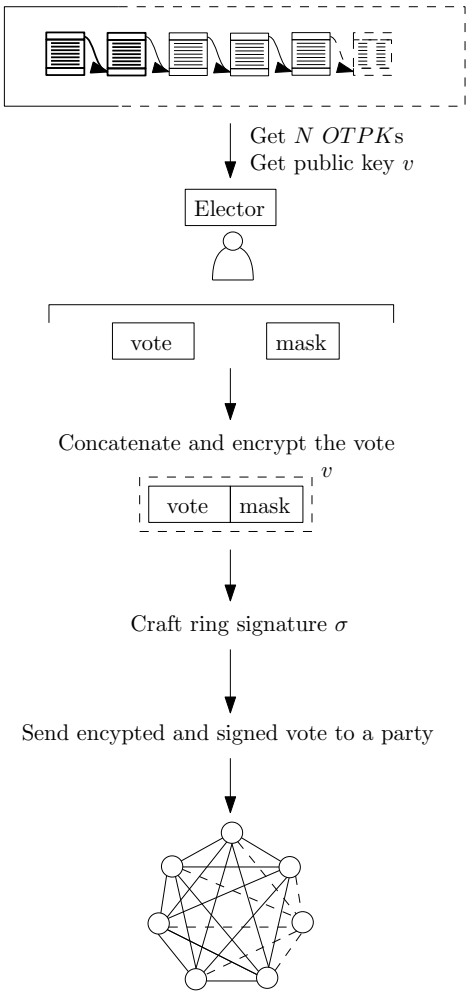


Figure 3: Electors consult election parameters from the blockchain. To cast a vote, they select a fixed length mask and concatenate it to the vote. Adjacent boxes represent concatenation, while the dashed box represents encryption using modular exponentiation. Electors craft a ring signature using the consulted $OTPK$s. When the ballot is properly signed and encrypted they can send it as a transaction to any party of their choice.

Finally, when the vote has been encrypted and signed, the elector sends the ballot through a blockchain transaction to a party of her choice or a random one. Figure 3 illustrates the casting process.

4.4 Vote Processing

Parties act like miners, listening to the blockchain network and expecting transactions addressed to them. When a party finds a transaction addressed to himself in the pool of unprocessed transactions, proceeds to verify the

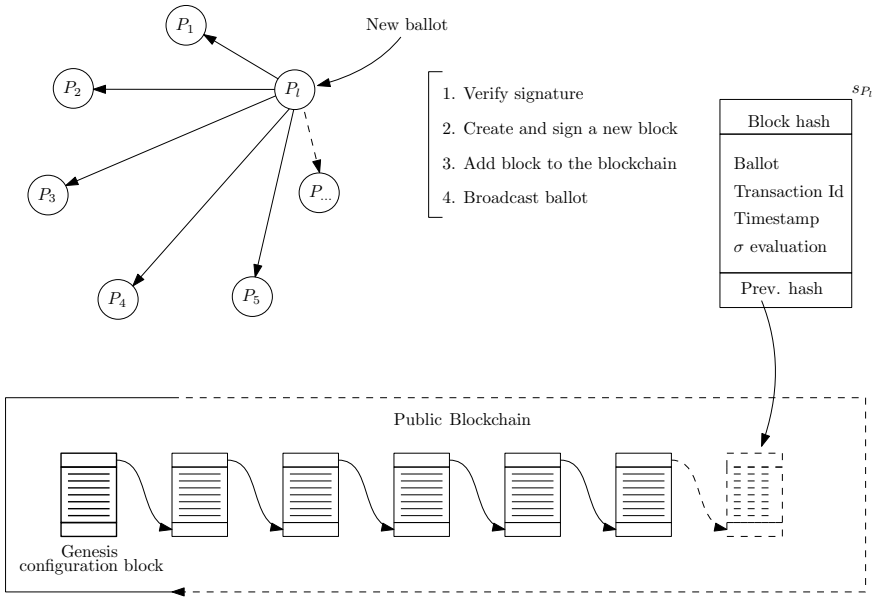


Figure 4: Process a party needs to perform to process a vote. Each party is responsible of evaluating received ballots and ensuring they are correctly added to the blockchain. All the posted blocks are signed with party's personal private key.

integrity of the ring signature. Figure 4 shows the processing of a vote. When it has received enough transactions, the party creates a block that is added to the blockchain and later broadcasted to the rest of parties:

1. It applies Algorithm 2 to the signature to verify its correctness.
2. It creates and signs a block with the following attributes:
   - Block Id.
   - Votes and transactions Id.
   - Timestamp.
   - Result of verifying the ring signature.
3. It adds the block to the blockchain.
4. It broadcast the new block to the rest of parties so they can also verify the votes.

For a more detailed and technical explanation about how the blocks are created and processed within the blockchain, we refer the reader to the Appendix A.

4.4.1 Consensus

How different nodes reach consensus in a distributed environment? Distributed systems, blockchains included, fall under the CAP theorem [9, 25] which states that a distributed system, can not provide a strong consensus (C), high service availability (A) and partition tolerance (P) simultaneously. Since availability and partition tolerance are binary properties (they are provided or not), consistency is usually the property degraded and results in different consistency models [32]. To reach consensus and provide consistency in blockchains, Bitcoin [44] popularized the Proof of Work algorithm [27]. When someone wants to write a new block in the chain, it must provide some proof of the expensive computational cost (e.g: a hash function with certain characteristics) invested on the block. If two blocks are validated near the same time and a bifurcation occurs, users must follow the longest chain, this is the chain with more work. If a block is left out of the longest chain, his proof of work needs to be recomputed in order to be added again.

Here we employ a cheaper consensus algorithm: PoA. The trust is distributed among a set of reduced parties with adversarial interests, only these parties have write access to the blockchain. Just like in game theory, a Nash equilibrium is reached in where different entities collaborate because there is no reward in following a different strategy. Indeed, since our approach is fully logged and auditable and the parties are linked with real life entities, the penalization for indecorous conduct is immense. Parties have write access, but since transactions are encrypted and anonymously signed, the user's privacy is not affected. By using PoA, our approach is faster since it does not require costly computation, environmentally friendly since it does not consume so much electricity and simpler to scale. In the improbable case of bifurcation, we follow the same principle as Proof of work, we follow the longest chain. Each party is responsible of making sure a block is properly written.

4.5 Tallying

Once the voting phases finish, parties stop accepting transactions from electors. Now they must proceed to compute the final tally.

First, parties must recover the secret key s to decrypt votes. To do so, a subset Λ of at least k honest parties collaborate to compute the original polynomial containing s. To recover a polynomial from k shares, defined as $s_0 = (x_0, y_0), s_1 = (x_1, y_1), \dots s_k = (x_k, y_k)$ we employ Lagrange polynomials:

$$\mathcal{L}(x) = \sum_{j=0}^{k} y_j l_j(x) \tag{1}$$

$$l_j(x) = \prod_{\substack{0 \leq m \leq k \\ m \in \Lambda \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \tag{2}$$

Once we recover the polynomial $\mathcal{L}(x)$, s can be obtained as:

$$s = \frac{L(0)\Delta}{\Delta^3} \tag{3}$$

being Δ a factor used when generating the RSA key. [1] When s is recovered, parties can decrypt the received votes and compute the final tally. Parties must analyze the received votes directly from electors and those received from another parties. This way, each party can compute a global and independent tally. If multiple votes contain the same key image, only the last vote will be considered valid.

When the tally is completed, each party has to publish a new block containing each received transaction, the result of checking the ring signature and the direction of the vote. At the end of the block they must add the results of the election and the private key of the election s. This message has to be signed by the parties. The private key is also published to allow electors to compute their own tally. If all the parties agree with the tally the election is finished. Figure 5 represents the tallying stage.

5 Computational time analysis

We devote this section to, first, analyze the asymptotic computational time complexity of our election scheme, both for the elector and the involved parties; and, second, to provide a comparison with the systems reviewed in Section 2.

Blockchain-based systems measure their throughput as the maximum number of transactions per second (TPS) they can process. TPS are heavily determined by the consensus algorithm employed: block proposal, block validation, and the mechanism used to solve forks among others. Here, we note that, by using PoA instead of Proof-of-Work, the limiting factor is no longer the block proposal but the block validation. Because no extensive hashing is required to propose blocks. Therefore, ring signatures, which are the main factor in block validation, determine the number of TPS.

Next, we summarize the main stages of a consensus protocol and determine the associated computational time complexity. For computing the asymptotic time complexity, we consider the modular exponentiation, employed in RSA and the crafting/verification of a ring signature, as basic units. This operation is the most expensive and the basic construction units

[1] We note that $\Delta = l!$. It is used for computing the random polynomials when using Frankel et al. proposal [20].

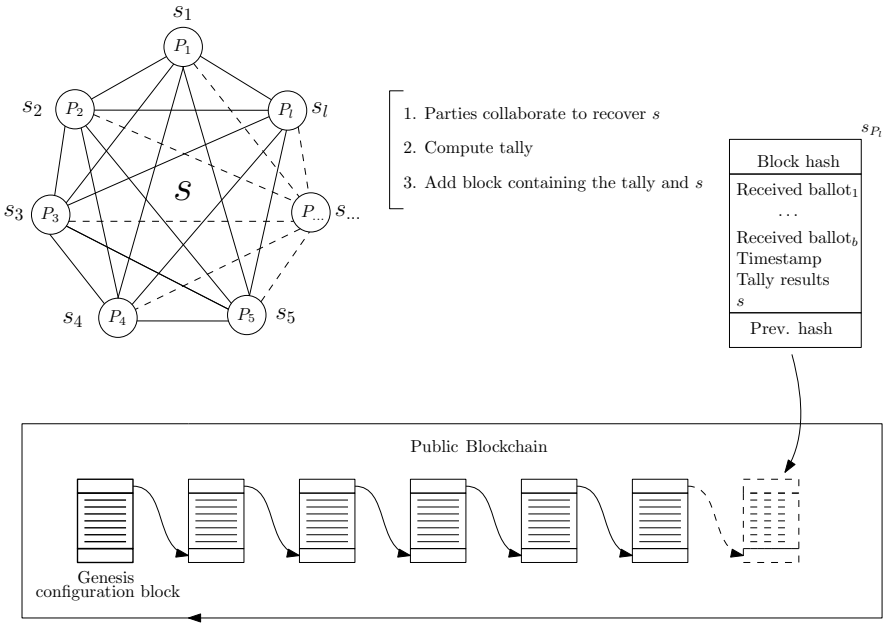


Figure 5: Parties collaborate to recover the private key s. Then, each of them individually computes a personal tally and adds it to the blockchain. It contains all the information needed by a third party to audit the tally.

of our scheme. Modular exponentiation has a time complexity of $\mathcal{O}(\log^3 n)$ bit operations [30], being n the input and $\log n$ its size in bits.

- **Block proposal:** Thanks to PoA, no intensive computation is required to propose a new block, no resource-consuming hashing is employed with respect to Proof-of-Work. Every identified party who receives enough transactions to craft a block may propose a new one. Only a modular exponentation is required to sign the proposed block. Then, the complexity to propose a block can be expressed as $\mathcal{O}(\log^3 n)$.

- **Block validation:** The validation of a block requires to check the ring signature of every transaction contained within it. The computational complexity of this validation varies depending on the elliptic curve used and the size of the ring signature employed. Also note that we require all parties to validate all the new blocks, therefore the validation process is also affected by the number of involved parties. To validate the block, the signature of the party who created the block must also be checked. This results in one additional modular exponentiation. The asymptotic complexity of block validation can be expressed as $\mathcal{O}(tr_v p + \log^3 n)$, where t represents the number of transactions, r_v the cost of verifying a ring signature and p the number of parties involved in the consensus.

- **Information propagation:** Message communication is another crucial factor to determine the throughput of a system. However, since we work with a permissioned blockchain, the number of nodes that send messages is minimal. A simple broadcast between parties is enough to communicate new blocks. Therefore, the number of messages remains linear with respect to the number of blocks b and the number of parties. Each message must be signed by the sending party, so the complexity can be regarded as: $\mathcal{O}(bp \log^3 n)$.

- **Block finalization:** For a block to be finally accepted by all parties, it must be added in the longest chain. If two parties try to add a block at the very same time, a collision occurs. As we mentioned on the previous section, we follow the longest chain rule and each party is responsible for checking their blocks are correctly added to the blockchain. As the validation of a block does not depend on the previous block's hash, adding an already validated block for a second time does not require further computation. However, it is safe to assume that probably the block would be outdated and a new block proposal and its corresponding propagation should be carried out. The complexity can be expressed as $\mathcal{O}(qbp \log^3 n)$, being q the number of collisions requiring a new block re-added to the blockchain. The number of collisions is difficult to estimate since it depends on many empirical factors and varies from election to election. The worst case would be an election on which all the electors vote at the same time and the ballots sent are equally distributed between all parties. That would result in many concurrent collisions. Because of the unequal distribution of a real election, we can assume q will be low when compared with successful finalized blocks. Nonetheless, if the number of collisions becomes a problem, we could abandon following longest chain in favor of a byzantine fault tolerance consensus or a round-based writing [52].

- **Reward mechanism:** In our adaptation of PoA, the reward and the purpose of the election scheme are the same thing. Parties are, allegedly, interested in carrying out the election process. Parties are motivated in an adequate election since their public reputations are at stake. Since the reward is abstract it does not affect the computational time complexity.

- **Tallying:** For computing the final tally, parties must collaborate to recover the secret key and then proceed to decrypt all the votes. The collaboration requires p signed messages and the decryption needs one modular exponentiation per transaction(vote). The complexity of the tally can be expresses as $\mathcal{O}(pt \log^3 n)$.

Therefore, the total time complexity of the system can be expressed as $\mathcal{O}(\log^3 n) + \mathcal{O}(tr_v p) + \mathcal{O}(\log^3 n) + \mathcal{O}(bp \log^3 n) + \mathcal{O}(qbp \log^3 n) + \mathcal{O}(pt \log^3 n)$.

Given that the number of blocks b depends linearly on the number of transactions t and that t dominates b, we can substitute b for t. After grouping some terms, the final complexity can be simplified as $\mathcal{O}(tp(r_v + q\log^3 n))$. Thus, the system's time complexity depends on the cost of verifying ring signatures and on the cost of running modular exponentiations, which are both affected by the number of involved parties, the number of collisions and the total transactions processed by the system.

We did not consider the cost of distributely generating the keys for RSA encryption because it can be done offline before the election process and it is carried out off the blockchain. Apart from the initial genesis blocks, which would have only required a pair of modular exponentiations, this process does not affect the complexity of our scheme.

Besides, one elector willing to craft and cast a vote, only needs to perform the encryption of the vote and the associated ring signature. The time complexity for the final user can be expressed as $\mathcal{O}(\log^3 n + r_c)$, being r_c the cost associated to craft a ring signature.

5.1 Ring signature performance

Unlike modular exponentiation, ring signatures are not an operator. Ring signatures are a quite complex cryptographic construct that includes multiple basic operators, so the comparison is not straightforward and computational time complexity is dominated by ring signatures. Ring signatures time complexity also depends on some parameters apart from the input size such as the size of the ring, or the desired level of security. For these reasons, in Section 5, we chose to leave the computational complexity associated to craft/verify a signature as a variable to provide a clear view of the time complexity. However, now we also provide an implementation of these signatures to present an empirical result of what the real TPS of the system would be.

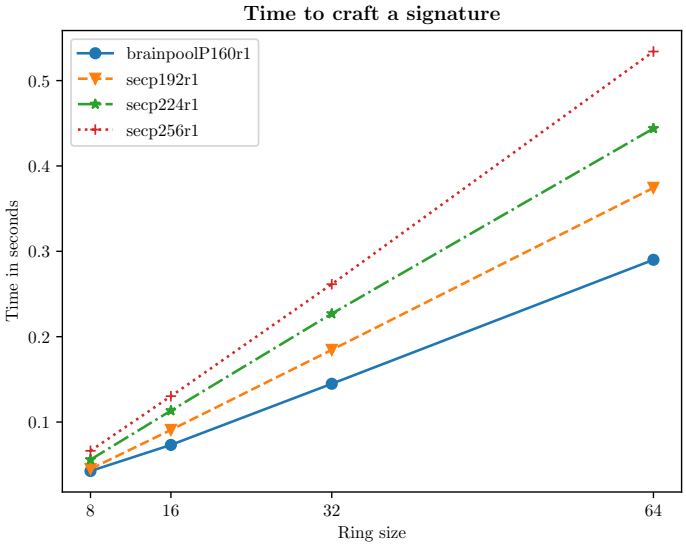
The code developed for this performance analysis was implemented using Python 3 and it is publicly available on GitHub[2]. We provide a framework to test how the different parameters and different elliptic curves affect the performance of ring signatures. Our goal with this framework is to provide a tangible implementation and to obtain real performance time measurements. A low-level implementation in a different language such as C++ would probably benefit the final throughput.

Figures 6a and 6b represent the elapsed time to craft or verify a signature under different parameters respectively. We chose 4 different elliptic curves for the comparative. Each one provides a different level of security: brainpoolP160r1 provides 80 bits of security; Curve-192 provides 91 bits; Curve-224 provides 112 bits and, Curve-256 provides 128 bits. Brainpool

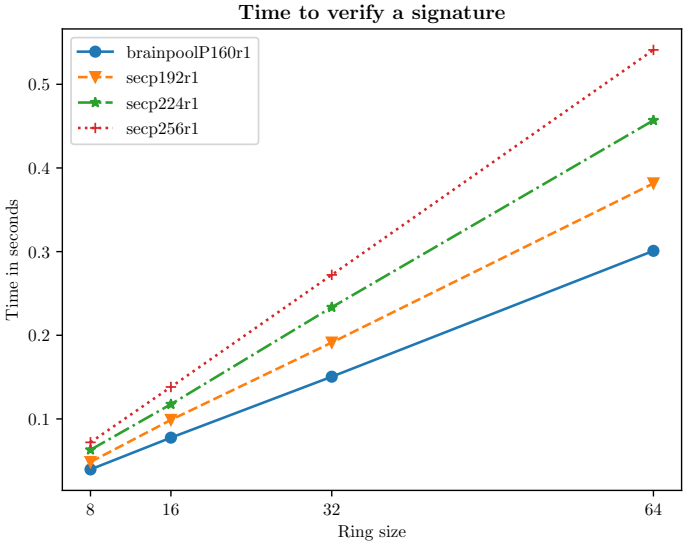
[2] `https://github.com/Fantoni0/RingCTPerformance`

curve provides a security level comparable to use RSA with a 1024 bits modulo [22], which is more than enough to protect the voter's privacy[3]. As expected, the required time grows linearly when we increase the size of the ring or the complexity of the curve. We believe that using brainpool and a ring size of 64 public keys is more than enough to achieve the security needed for our scheme. Specially due to the short term of security required by our proposal: the votes only need to be secret until the final tally is computed.

[3] A 1024-bit key in RSA is considered safe for the next decades. No key bigger than 829 bits has ever been factored.



(a) Time required to craft a ring signature under different ring sizes and elliptic curves.



(b) Time required to verify a ring signature under different ring sizes and elliptic curves.

Figure 6: Ring signature performance times for crafting and verifying the same message under different parameters. Experiments carried out on a AMD Ryzen 7 3700X with 8 cores and 16 threads.

The figures here presented were obtained using a personal computer (AMD Ryzen 7 3700X). Since there are no dependencies between transactions, the verification is a parallelizable task. We use multiple cores to take

advantage of this aspect. Using a professional server's processor and decentralizing the task among multiple servers would yield a great performance improvement. Nonetheless, a single personal computer is capable of verifying 3-4 TPS and 200 in a minute, using immoderate high standard security parameters.

Also note that the times for crafting/verification are very similar. This is because the verification algorithm (See Section 3.3) requires to re-create the signature to check its validity.

5.2 Comparative evaluation of systems

We devote this section to compare the performance of our proposal with those studied in Section 2. Ring signatures and modular exponentiations determine the time complexity of our approach. Modular exponentiations are a far common operator present in most of the related works (except if they are completely based on ECC). For this reason, we consider modular exponentiation, and its associated complexity in bit operations, as the atomic unit in the analysis of the time complexity.

Comparing different e-voting proposals is not trivial: some systems do no disclose all the details, not all the systems are directly comparable and not all the works provide a time complexity analysis. Thus, some of the reviewed works, are left out of the comparison: because they do not provide a performance analysis either enough information to obtain an asymptotic complexity [3, 29, 35, 46, 50]; because, despite providing a thorough analysis, they are based on different problems and the analogy would not hold [23] ;or, because they are implementation approaches and the authors only report user timings [51].

Let us note, that despite not providing a time anlaysis, Wei supplies a complexity analysis of the ring signatures employed under different ring sizes and the associated gas (unit for measuring cost of Ethereum transactions) cost of running their voting protocol on the Ethereum blockchain. Also, Wu provides empirical times and sizes of the ring signatures used with varying ring sizes. Unfortunately, neither of them detail the level of security engaged on those tasks.

We compare the asymptotic time complexity of the remaining works to prove the validity of our approach. When the methods do not specify a part of their protocol or do not provide enough information, we introduce a variable in the complexity analysis. Table 1 summarizes the associated complexity for the elector and for the protocol to process the received votes. For more details, we refer the reader to the original works. Protocols employing different kinds of ring signatures are compared, to provide a fair example we assume the time to craft r_c or to verify r_v a signature are comparable and can be agglutinated under the same variable despite their different implementations.

	Elector's Cost	System's Cost
Salazar et al. [43]	$\mathcal{O}(r(\log^3 n + r_c))$	$\mathcal{O}(vr(\log^3 n + r_v))$
Chen et al. [15]	$\mathcal{O}(\log^3 n + r_c)$	$\mathcal{O}(vp(\log^3 n + r_v))$
Yang et al. [53]	$\mathcal{O}(cs \log^3 n)$	$\mathcal{O}(cts \log^3 n)$
Our proposal	$\mathcal{O}(\log^3 n + r_c)$	$\mathcal{O}(tp(q \log^3 n + r_v))$

Table 1: Table representing the asymptotic complexity of the work performed by the elector and the system in number of bit operations. In the table: r refers to the number of rounds in the case of round voting,v represents the number of votes, c represents the number of candidates, s references the number of possible scores for each candidate in the case of ranked elections, t represents the number of transactions in a blockchain environment, and p the number of parties involved.

Note that the number of votes v and the number of transactions t are semantically equivalent, they both represent the number of processed votes. On the other hand, the number of candidates c and the number of parties p, not always represent the same partners. Not all protocols directly involve the candidates and some systems include extra authorities to handle credentials or distribute responsibility.

In Table 1, the results obtained by our proposal compete with the reviewed systems. Indeed, Chen's system and our proposal require the minimum effort for the final elector. Regarding system's complexity; it can be observed that, as for many works, it scales linearly with the number of involved parties and total number of votes. Salazar and Yang's works are also affected by other factors given that they support round and ranking e-voting respectively. Our proposed e-voting protocol is scalable due to its linear complexity and introduces the blockchain as a distributed public ledger without losing performance with respect to analogous works.

6 Properties

A voting scheme can be described by its properties. These properties define what it can provide and under which circumstances. Usually, the desired properties to be held by any electronic voting system are: verifiability, accuracy, democracy, privacy and robustness. We now discuss and prove that our proposal fulfills all of them. Let us note our proposal is based on well-known cryptographic primitives, therefore most of the proofs rely on the underlying problems of those primitives.

Verifiability

Verifiability implies the existence of auditing mechanisms for the election, ensuring that the voting process has been correctly developed. We here

distinguish three types of verifiability:

- Casted-as-intended: the ballot is sent with the desired vote direction.
- Recorded-as-casted: the ballot is recorded in the blockchain as it was sent.
- Tallied-as-recorded: the ballot will be tallied with the same vote direction as recorded.

Theorem 1: Our e-voting protocol is end-to-end verifiable.

Proof: Key images (see Section 3.2) work as a private receipt. Thus, allowing the elector to read from the blockchain to check her ballot was casted, recorded and tallied properly. As mentioned above, key images are anonymous and do not compromise elector's privacy.

Concerning universal verifiability, we note that any person, participant or not in the election process, is able to ensure that every vote has been tallied-as-recorded. This is achieved thanks to the public nature of the blockchain. Note that this does not ensure that the vote has neither been casted-as-intended nor recorded-as-casted because that would violate the privacy property.

Summarizing, our proposal provides universal verifiability by posting in the blockchain the key to decrypt the orientation of every vote recorded. Anyone can take the key and the votes stored in the blockchain to compute a tally by themselves. Thus, allowing to audit the final result of the election.

Accuracy

Also known as *correctness*, it demands that the tally corresponds with the actual outcome of the election. To achieve this: no one can change anyone else's vote; all valid votes are included in the final tally; and, no invalid vote will be included in the tally.

Theorem 2: As RSA encryption system remains secure, the system is auditable by third parties, and, ring signatures are unforgeable, then our voting protocol is accurate.

Proof: We divide the proof in three parts:

(a) *No one can change anyone else's vote:* Votes are encrypted using RSA: assuming that no elector shares his secret key; that at last k parties are honest; and, that the Discrete Logarithm Problem (DLP) has no efficient solution for carefully selected parameters, the votes remain unaltered until the final tally.

(b) *All valid votes are included in the final tally:* Parties are responsible for processing and tallying all received valid votes. Given universal verifiability proved on Theorem 6, not including all valid votes would result in an early finalization of the election.

(c) *No invalid vote will be included in the tally:* For a vote to be included in the final tally, its ring signature must be valid. Assuming that the Elliptic-Curve-DLP(ECDLP) has no efficient solution, no signature can be forged.

Democracy

Democracy guarantees that only eligible electors are allowed to cast a vote, and that they can only do it once.

Theorem 3: If the ECDLP is semantically secure, no one can impersonate a valid elector or perform double voting.

Proof: The proof can be separated in two parts:

(a) *Eligibility:* Only electors in the census are able to register their public keys in the registration administration. The list of public keys and elector's identifier will be stored in the blockchain to prevent the administration from creating fake electors. If the ECDLP problem is computationally secure. Then, only the registered elector can recover their personal $OTPK$ from the public list.

(b) *Double voting:* Key images work as a commitment of the private and public key of a elector. As stated previously, if the ECDLP has no efficient solution, then no modification of the key image can be made. Therefore, each elector is authorized, without revealing her real identity, to vote only once.

Privacy

Privacy refers to the inability of linking a elector's identity to the direction of her vote.

Theorem 4: If the ECDLP has no efficient solution, elector's identity remains private.

Proof: Key images, ring signatures and $OTPK$s are based on ECDLP. These are the only cryptographic constructs related to elector's identity. Assuming that, under the right parameters, no efficient solution exists for ECDLP, we can conclude there is no method to expose the elector's identity. Therefore, the elector is protected by the size of the ring signature, because any member of the ring is a possible signer with the same probability. We also stress that even if two equal votes were encrypted using the same ring of public keys, their signature will differ and privacy will be granted.

Robustness

Robustness implies that no reasonably sized coalition of electors nor authorities would be able to significantly disrupt the election. As mentioned above, in a (k, l)-threshold RSA key sharing scheme l represents the total number of involved parties, and k represents the minimum of collaborating parties needed to recover the secret key.

Theorem 5: In a (k, l)-threshold RSA key sharing scheme, if at least $l - k + 1$ parties are honest, our voting protocol is robust.

Proof: Parties are the only ones with write access, thus, electors cannot directly interfere in any of the processes stages. To recover the private key and compute the final tally at least k parties must cooperate. If a party (or any subset of them lower than k) misbehave, their actions are publicly auditable through the blockchain. Therefore they can be sanctioned and left out of the process without compromising the running election nor the final tally. Let us note, that even in the worst case of k malicious parties colliding to recover the secret key before the election ends, elector's privacy and the integrity of the vote will prevail since they depend on the ring signature bounded to each vote. They will be only capable of knowing the directions of the votes before the tally phase.

Other properties

Uncoercibility refers to the impossibility of an elector of being coerced to change her vote. It is tightly related with the concept of receipt-freeness: if a voter can not craft a receipt to prove how she voted, the coercers will not get any reassurance. Our system does not provide receipt-freeness, an elector can use her key image as receipt the direction of her vote. Receipt-freeness can be obtained by letting the authorities generate or manage some part of the credentials. However, this somehow contradicts some basic properties of voting-schemes. We have decided to prioritize and emphasize the properties based on the individual confidence with the system, allowing electors to have a receipt. Nevertheless, note that our proposal allows for multiple-voting, allowing to consider the last vote as the only valid. While this does not provide complete uncoercibility, it provides a bribed or coerced elector with a mechanism to later change her vote.

7 Conclusions

In this paper, we present a new voting protocol. Our proposed scheme, is secure and focuses on providing arguments to promote participation and trust issues. Without renouncing to secure and solid cryptographic proofs, we put the traditional parties of the voting process inside the system. Each political party was given limited capabilities to reach a shared goal. They are accountable for their actions and every misconduct can be detected and audited. The trust is therefore distributed and local misbehaviors are logged and do not compromise the robustness of the system. The signer ambiguity provided by ring signatures and the public and decentralized blockchain, make possible a secure, public and universally verifiable voting system. All the process is articulated through the blockchain, all the related information is contained on it. When the election ends, the private key is made public

and anyone can review the votes, the parties' actions and the final tally.

We adapt PoA to the problem of electronic voting, as a cheaper consensus algorithm to distribute trust. PoA is intended for situations when because of the problem definition, a small group requires a special role. It fits perfectly in the electronic voting paradigm. PoA allowed us to introduce political parties as active partners in the voting process to increase confidence in the system as ledger criterium. PoA is also more efficient in terms of computational work and makes it easily scalable to different types of elections.

Apart from the (k, l)-threshold RSA key sharing scheme, which grows loglinear with the number of involved parties, the rest of the system is linear, since ring signatures scale linearly with the size of the ring and all the computational work done by the parties scales linearly with the number of votes. Note that the threshold RSA sharing is done off-line and its precomputation does not affect the election performance.

References

[1] Follow my vote. *https://followmyvote.com/*, 2018.

[2] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer, 2003.

[3] Ahmed Ben Ayed. A conceptual secure blockchain-based electronic voting system. *International Journal of Network Security & Its Applications*, 9(3):01–09, 2017.

[4] Adam Back. Ring signature efficiency, 2015. Available at `https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684`.

[5] Igor Barinov, Viktor Baranov, and Pavel Khahulin. Poa network white paper. *URL: https://github. com/poanetwork/wiki/wiki/POA-Network-Whitepaper*, 2018.

[6] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, PODC 2001, Newport, Rhode Island, USA, August 26-29, 2001*, pages 274–283, 2001.

[7] Marianna Belotti, Nikola Bozic, Guy Pujolle, and Stefano Secci. A vademecum on blockchain technologies: When, which, and how. *IEEE Commun. Surv. Tutorials*, 21(4):3796–3838, 2019.

[8] Sean Bowe, Ariel Gabizon, and Matthew D Green. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. *LNCS*, 10958:64–77, 2018. Proceedings of the International Conference on Financial Cryptography and Data Security.

[9] Eric Brewer. CAP twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

[10] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Blind signatures based on the discrete logarithm problem. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 428–432, 1994.

[11] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[12] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*, pages 199–203, 1982.

[13] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 118–139, 2005.

[14] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[15] Guomin Chen, Chunhui Wu, Wei Han, Xiaofeng Chen, Hyunrok Lee, and Kwangjo Kim. A new receipt-free voting scheme based on linkable ring signature for designated verifiers. In *2008 International Conference on Embedded Software and Systems Symposia*, pages 18–23. IEEE, 2008.

[16] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5):481–490, 1997.

[17] Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application*

of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding, pages 152–165, 2001.

[18] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6):644–654, 1976.

[19] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, 1985.

[20] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Optimal resilience proactive public-key cryptosystems. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 384–393, 1997.

[21] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed rsa-key generation. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 663–672, 1998.

[22] Patrick Gallagher. Digital signature standard (dss). *Federal Information Processing Standards Publications, volume FIPS*, 186, 2013.

[23] Shiyao Gao, Dong Zheng, Rui Guo, Chunming Jing, and Chencheng Hu. An anti-quantum e-voting protocol in blockchain with audit function. *IEEE Access*, 7:115304–115316, 2019.

[24] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[25] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[26] Yang Hua-jie, Miao Xiang-hua, Zhu Hai-tao, and Li Yi-ran. Efficient certificateless ring signature scheme with identity tracing. *Information Security and Technology*, (7):9, 2014.

[27] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), September 20-21, 1999, Leuven, Belgium*, pages 258–272, 1999.

[28] Koe, Kurt M. Alonso, and Sarang Noether. Zero to monero: Second edition. Available at `https://web.getmonero.org/library/Zero-to-Monero-2-0-0.pdf`, 2020.

[29] Kibin Lee, Joshua I James, Tekachew G Ejeta, and Hyoung J Kim. Electronic voting service using block-chain. *Journal of Digital Forensics, Security and Law*, 11(2):8, 2016.

[30] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[31] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, pages 122–134, 1980.

[32] Francesc D. Muñoz-Escoí, Rubén de Juan-Marín, José-Ramón García-Escrivá, José Ramón González de Mendívil, and José M. Bernabéu-Aubán. CAP theorem: Revision of its related consistency models. *Comput. J.*, 62(6):943–960, 2019.

[33] Harald Niederreiter. A public-key cryptosystem based on shift register sequences. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 35–39. Springer, 1985.

[34] Shen Noether. Ring signature confidential transactions for Monero. *IACR Cryptol. ePrint Arch.*, 2015. Available at `https://eprint.iacr.org/2015/1098`.

[35] Pierre Noizat. Blockchain electronic vote. In *Handbook of digital currency*, pages 453–461. Elsevier, 2015.

[36] Remigijus Paulavicius, Saulius Grigaitis, Aleksandr Igumenov, and Ernestas Filatovas. A decade of blockchain: Review of the current status, challenges, and future directions. *Informatica*, 30(4):729–748, 2019.

[37] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

[38] Ronald L Rivest. The threeballot voting system, 2006. Available at `http://theory.csail.mit.edu/~rivest/Rivest-TheThreeBallotVotingSystem.pdf`.

[39] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *Proceedings Of The 7th International Conference On The Theory And Application Of Cryptology And Information Security: Advances In Cryptology*, pages 554–567. Springer-Verlag, 2001.

[40] Ronald L Rivest and Warren D Smith. Three voting protocols: Threeballot, vav, and twin. *USENIX/ACCURATE Electronic Voting Technology (EVT 2007)*, 2007.

[41] Morgan Rockwell. Bitcongress process for blockchain voting & law. *https://cryptochainuni.com/wp-content/uploads/BitCongress-Whitepaper.pdf*, 2017.

[42] Tim Ruffing and Pedro Moreno-Sanchez. Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. *LNCS*, 10323:133–154, 2017. Proceedings of the International Conference on Financial Cryptography and Data Security.

[43] José Luis Salazar, Joan Josep Piles, José Ruíz-Mas, and José María Moreno-Jiménez. Security approaches in e-cognocracy. *Computer Standards & Interfaces*, 32(5-6):256–265, 2010.

[44] Nakamoto Satoshi. Bitcoin: a peer-to-peer electronic cash system. Available at `https://bitcoin.org/bitcoin.pdf`, 2008.

[45] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[46] Pavel Tarasov and Hitesh Tewari. Internet voting using zcash, 2017. Available at `https://dblp.org/rec/bib/journals/iacr/TarasovT17`.

[47] José Luis Tornos, José Luis Salazar, Joan Josep Piles, Jose Saldana, Luis Casadesus, José Ruíz-Mas, and Julián Fernández-Navajas. An evoting system based on ring signatures. *Network Protocols & Algorithms*, 6(2):38–54, 2014.

[48] Patrick P. Tsang and Victor K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. *IACR Cryptology ePrint Archive*, 2004:281, 2004. Available at `http://eprint.iacr.org/2004/281`.

[49] Nicolas Van Saberhagen. Cryptonote, 2013. Available at `https://cryptonote.org/whitepaper.pdf`.

[50] Wei-Jr Lai Ja-Ling Wu. An efficient and effective decentralized anonymous voting system, 2018. Available at `https://arxiv.org/abs/1804.06674`.

[51] Yifan Wu. An e-voting system based on blockchain and ring signature. Master's thesis, University of Birmingham, 2017.

[52] Yang Xiao, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Commun. Surv. Tutorials*, 22(2):1432–1465, 2020.

[53] Xuechao Yang, Xun Yi, Surya Nepal, Andrei Kelarev, and Fengling Han. Blockchain voting: Publicly verifiable online voting protocol without trusted tallying authorities. *Future Gener. Comput. Syst.*, 112:859–874, 2020.

A Blockchain Technical Specification

We devote this appendix to provide the technical specification of the blockchain described in the article. Our purpose is to provide the specifics and the structure needed to implement the blockchain required to run our voting protocol.

We divide the specification into two sections. First, we detail the data structures that define the blockchain. Secondly, we determine the methods the nodes (parties) will need to operate in a functional blockchain. Some implementation details such as node discovery, size in bytes or network particulars are not considered because they are not in the scope of this appendix. However, we provide enough information for an interested reader to build a functional blockchain.

A.1 Blockchain data structures

Here we define the structure for transactions and blocks. Our protocol is designed for e-voting. Therefore, we could get rid of the monetary tokens, however we decided to provide the blockchain specification with support for a token. Parties are responsible for supplying the $OTPK$s during the registration phase with enough tokens to create transactions. There must be enough tokens to allow for multiple votes to prevent coercion. If needed, the change to a version without support for tokens is straightforward.

A.1.1 Transactions

Transactions are the basic unit of information on the blockchain. They are broadcasted into the public network and defined by the set of referenced inputs and outputs. They are used to register votes in our e-voting protocol. In our e-voting scheme, the inputs represent the elector's $OTPK$, while the outputs represent the parties' addresses. The elector decides to which party sends the vote by configuring the outputs of the transaction. Table 2 discloses the structure of transactions and Table 3 defines the structure of inputs/outputs.

Field	Definition
Version	Number version of the protocol
Inputs	List of referenced inputs
Outputs	List of outputs
Vote	Encrypted elector's vote
Signature	Ring signature of the transaction
Transaction ID	Hash identifying the transaction

Table 2: Transaction structure.

Field	Definition
Amount	Amount of tokens in the key
Address	Public key identifying the address

Table 3: Input and output structure.

A.1.2 Blocks

A block is a set of bundled and validated transactions. Blocks constitute the basic building component of a blockchain, because as its name implies, a blockchain is an ordered set of blocks. Table 4 shows the structure of an ordinary block in our blockchain.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Signature	Digital signature of the block creator
Transactions	List of the transactions contained
Merkle Root	Merkle's root hash of the transactions
Block ID	Hash identifying the block

Table 4: Block structure.

Besides from general blocks containing votes, our scheme contains three special blocks. The first two configuration blocks, whose structure is detailed in Tables 5 and 6 respectively. And the last tallying block as described in Table 7. The first two blocks contain all the public information and parameters needed to properly run the election. The last tallying block contains the secret key needed to decrypt the votes and provides a summary of the results of the election.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Height	Distance to the first block
RSA parameters	Public key v and modulus n
Private shares	Commitments g^{s_i} of the private keys for each party
Public keys	Parties' public keys
Block ID	Hash identifying the block

Table 5: First block in the chain describing configuration parameters.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Start Time	Start time of the election
End Time	End time of the election
Ring Size	Minimun accepted ring size for ring signatures
Options	List of options to vote for in the election
$OTPK$s	List of $OTPK$s and their corresponding random number R
Block ID	Hash identifying the block

Table 6: Second block in the chain describing configuration parameters.

Field	Definition
Version	Number version of the protocol
Timestamp	Time of the block creation
Previous Hash	Hash identifying the previous block
Height	Distance to the first block
Secret key	Recovered secret key to decrypt votes
Results	Final election tally
List of results	Referenced transactions for each result
Block ID	Hash identifying the block

Table 7: Last block in the chain describing the tally results.

In Figure 7, we can see the structure of a complete blockchain. We can appreciate how the different blocks work and how the different data structures are engaged to build the blockchain.

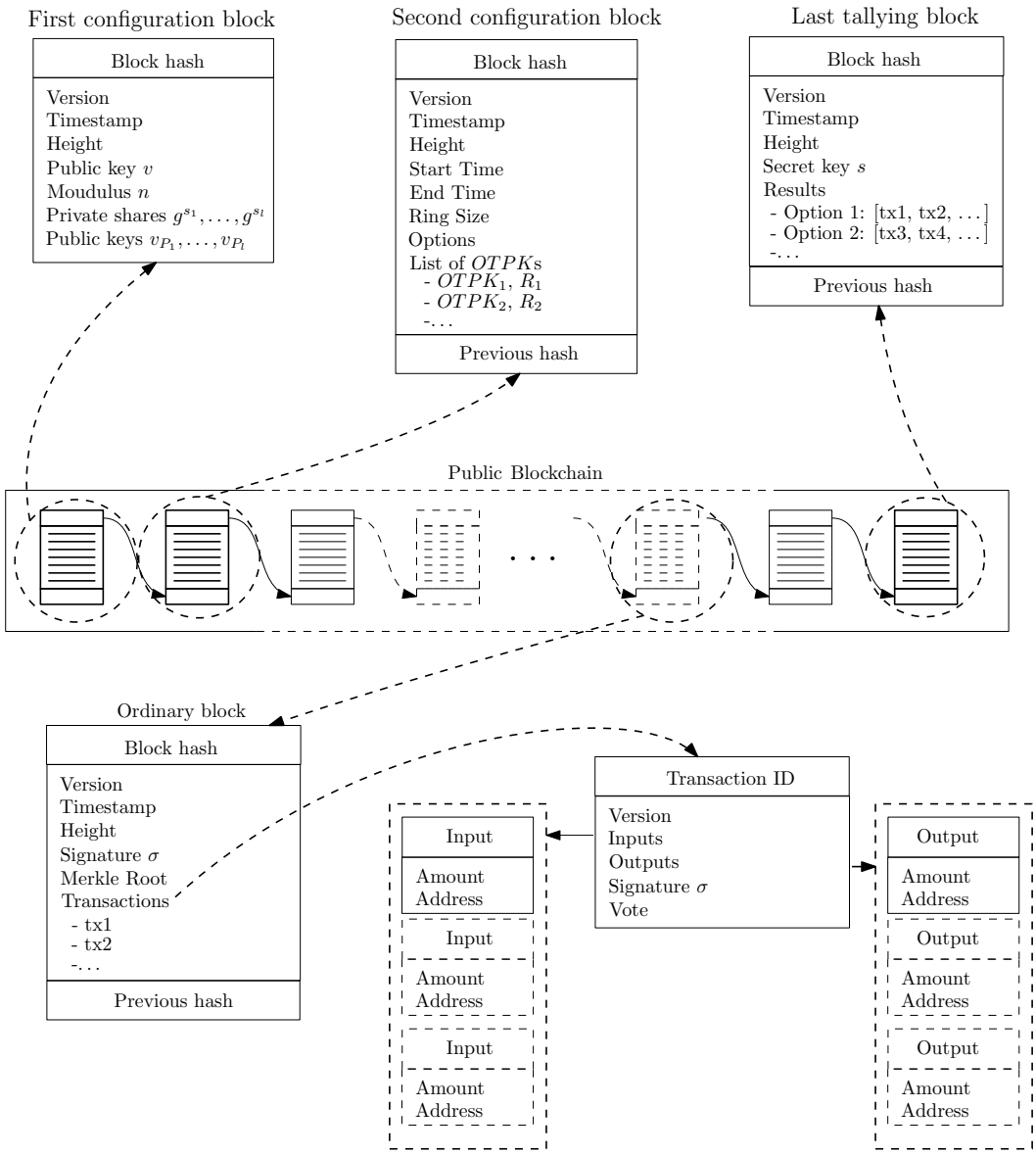


Figure 7: Holistic view of all the data structures involved in the blockchain. We can appreciate the 4 different block types and its inner architecture.

A.2 Methods

In this section, we provide a set of algorithms describing the methods the nodes in the blockchain must follow to operate. These methods provide a high-level overview for the different scenarios nodes have to face, such as: process new blocks, handle transactions or follow the longest chain.

We assume the existence of some predefined functions, when no algorithm describes the function is because we believe the name it is self-explanatory (e.g: hash() assumes a function that generates a hash for a given input, verifySignature() assumes a function that verifies the RSA signature of the given input). We also assume some predefined variables such as: time, localBlockchain, and other evident and presumed values.

Now we describe the different triggers and methods that incur from the casting of a vote to its final addition in the blockchain.

A.2.1 Casting a vote

The process for an elector node in the blockchain to craft and cast a vote is described on Algorithm 3. The transaction containing the vote is then added to the pool of unprocessed transactions, where it will be stored there until is properly verified.

Algorithm 3 Voting process(Craft vote and send transaction)

Require: voteDirection ← Vote direction.
partyPublicKey ← Public key of the authority designated to add the block to the blockchain.
publicKeyList ← List of public keys to form the Ring Signature.
pivateKey ← Private key of the user.
hiddenIndex ← Index in the list publicKeyList where the public key of the signer is.
rsaPublicKey ← RSA public key to encrypt the vote direction until the tally.
tokens ← Inputs containing he required tokens.

1: mask ← randomMask()
2: maskedVoteDirection ← (voteDirection ‖ mask)
3: encryptedVoteDirection ← $(maskedVoteDirection)^{rsaPublicKey}$
4: ringSignature ← sign(publicKeyList.length, publicKeyList, hiddenIndex, privateKey, encryptedVoteDirection) (see Algorithm 1)
5: broadcast(Transaction(
Version: version,
Inputs: tokens,
Outputs: [partyPublicKey],
Signature: ringSignature,
Vote: encryptedVoteDirection,
TransactionID: hash(Version, Inputs, Outputs, Signature, Vote)))

A.2.2 Processing transactions

Parties must listen for new transactions addressed to them in the pool of unprocessed transactions. They are responsible for validating new transactions addressed to them, as described on Algorithm 4. This process includes checking the public keys used in the ring signature are included in the census of authorized electors, verifying the validity ring signature and checking the inputs have enough tokens to operate on the blockchain.

Algorithm 4 Validate Transaction

Require: transaction ← Transaction to validate.
1: **if** transaction.ID ≠ hash(transaction) **then**
2: **return** False
3: **end if**
4: **if** ∃ pk ∈ transaction.Signature.PublicKeys | pk ∉ census **then**
5: **return** False
6: **end if**
7: **if** ¬ verifyRingSignature(transaction) **then**
8: **return** False
9: **end if**
10: **if** $\sum_{i \in transaction.Inputs}$ < requiredTokens **then**
11: **return** False
12: **end if**
13: validatedTransactions.append(transaction)
14: **return** True

Once they have received and validated enough transactions, nodes can generate and broadcast new blocks to the rest of parties as described on Algorithm 5.

Algorithm 5 Generating new blocks

Require: listTransactions ← List of validated transactions.
Require: version ← Version of the used protocol.
Require: lastBlock ← Previous last block added to the longest chain.
Require: localBlockChain ← Blockchain in the node's memory.
1: version ← version
2: timestamp ← time.now()
3: previousHash ← lastBlock.ID
4: height ← lastBlock.height + 1
5: transactions ← listTransactions
6: merkleRoot ← merkleTreeRoot(transactions)
7: ID ← hash(version, timestamp, previousHash, height, transactions, merkleRoot)
8: signature ← (version, timestamp, previousHash, height, transactions, merkleRoot, ID)^{s_P}
9: newBlock ← Block(version, timestamp, previousHash, height, transactions, merkleRoot, ID, signature)
10: localBlockChain.append(newBlock)
11: $\forall_p \in$ Parties send(p, newBlock)

A.2.3 Handling blocks

Blockchain nodes have to deal with the received blocks from other parties. They have to ensure they follow the longest chain as well as validate the received block and the contained transactions. On the one hand, Algorithm 6 shows the process the nodes follow to validate received blocks. As it can be seen in Line 9, the verifying node alerts the rest of nodes when he detects and invalid transaction on the block.

Algorithm 6 Validate block

```
Require: block ← Block to validate.
 1: if ¬ verifySignature(block.signature) then
 2:     return False
 3: end if
 4: if merkleRoot(block.transactions) ≠ block.merkleRoot then
 5:     return False
 6: end if
 7: for t ∈ block.Transactions do
 8:     if t.ID ∉ validatedTransactions ∨
        ¬ validateTransaction(t) then
 9:         AlertError(block.ID, t.ID)
10:         return False
11:     end if
12: end for
13: return True
```

On the other hand, Algorithm 7 describes the process for adding the block to the longest chain. The algorithm shows a logical simplification of how the nodes should follow the longest chain. For a real implementation, we refer the reader to Bitcoin's open source code. As Lines 10 and 12 exemplify, the node should employ a set of buffers for storing multiple chains. In Line 14, when the node receives a block ahead of his own chain, the node has to ask for the missing blocks to other nodes in the network.´

Algorithm 7 Add Block

Require: block ← Block received through the network.
Require: localBlockChain ← Longest blockchain in the node's memory.

```
1: if ¬ validateBlock(block) then
2:    return False
3: end if
4: last ← localBlockChain.lastestBlock
5: if last.ID = block.previousHash then
6:    localBlockChain.append(block)
7: else
8:    if block ∉ localBlockChain then
9:       if block.height < last.height then
10:         addToSecondaryChain(block)
11:      else
12:         saveAsSecondaryChain(localBlokchain)
13:         localBlockchain.append(block)
14:         ∀ b | b.height > last.height ∧
            b.height < block.height askForBlock(b)
15:      end if
16:   end if
17: end if
```

Finally, Figure 8 illustrates the process for casting and processing a vote as an interaction diagram. The algorithms described above are referenced in the interaction diagram. Some simplifications were made to addressed the complex interactions between parties and the blockchain on a single image. Despite not being perfectly accurate, the figure succeeds on representing the different election stages, the multiple partners and their interaction during the whole election.

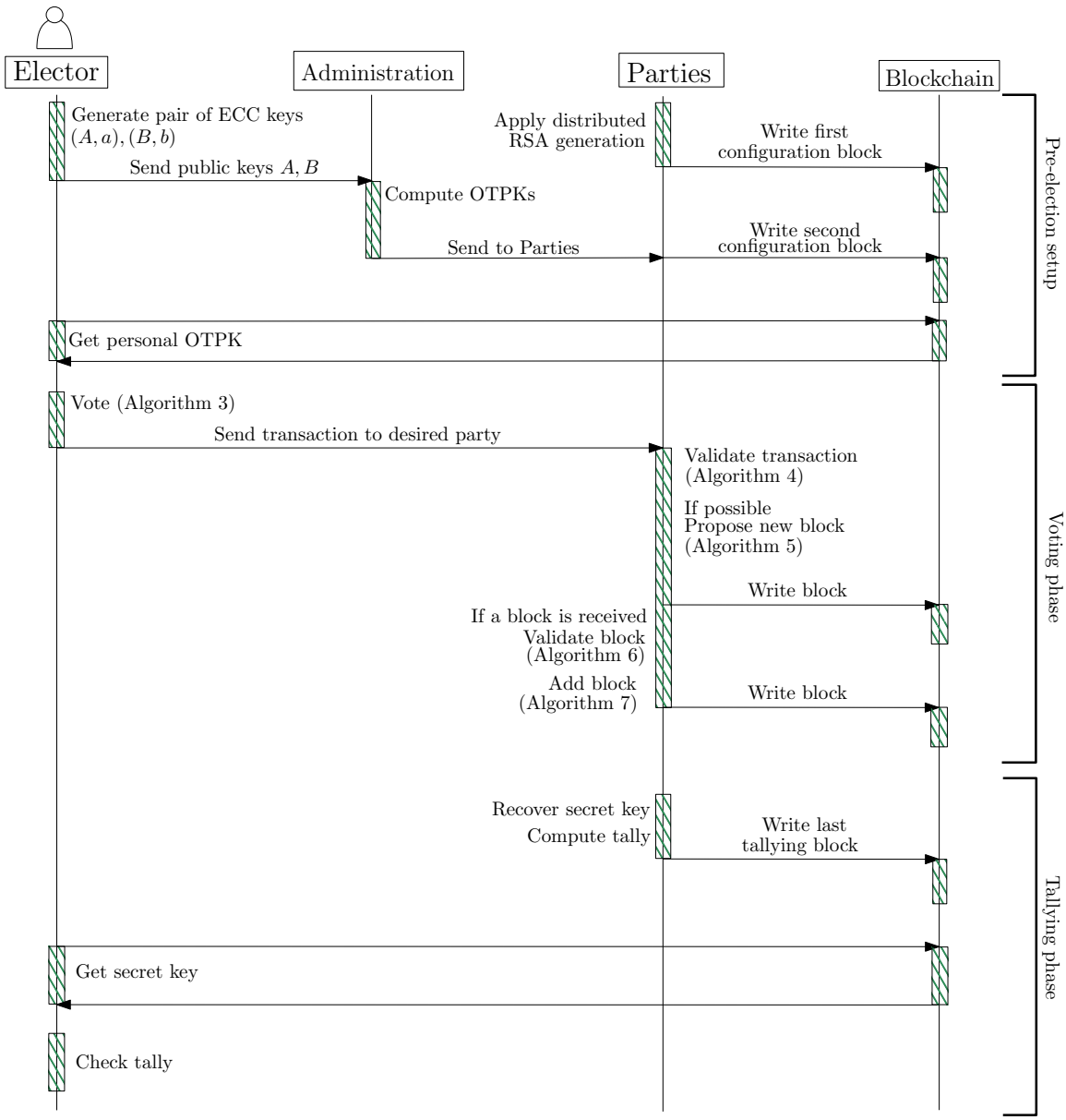


Figure 8: Timing and partners' interaction of the proposed voting scheme. The image shows the different election phases and the processes triggered at different stages. It shows the computations and the interactions needed as a time-interaction diagram.