Chirotonia: A Scalable and Secure e-Voting Framework based on Blockchains and Linkable Ring Signatures

Antonio Russo
Antonio Fernández Anta
IMDEA Networks Institute
Madrid, Spain
email: antonio.russo@imdea.org
email: antonio.fernandez@imdea.org

María Isabel González Vasco
Departamento de Matemática Aplicada
Universidad Rey Juan Carlos
Mostoles, Madrid, Spain
email: mariaisabel.vasco@urjc.es

Simon Pietro Romano
DIETI
University of Napoli, Federico II
Naples, Italy
email: spromano@unina.it

***Abstract*—In this paper we propose a comprehensive and scalable framework to build secure-by-design e-voting systems. Decentralization, transparency, determinism, and untamperability of votes are granted by dedicated smart contracts on a blockchain, while voter authenticity and anonymity are achieved through (provable secure) linkable ring signatures. These, in combination with suitable smart contract constraints, also grant protection from double voting. Our design is presented in detail, focusing on its security guarantees and the design choices that allow it to scale to a large number of voters. Finally, we present a proof-of-concept implementation of the proposed framework, made available as open source.**

***Index Terms*—E-voting, linkable ring signatures, blockchain, smart contracts, Ethereum.**

I. Introduction

Building an e-voting system represents a big challenge, due to strong requirements in terms of confidentiality, integrity and availability. Such a system must preserve these properties not only when facing external attackers, but also in the presence of malicious insiders, like regular or administrative users who misbehave and try to undermine the correct operation of the e-voting system. It is usually expected from an electronic voting scheme that it provides at least the same guarantees as those of a classic (paper-based) voting system, while adding the advantages of a modern electronic system, like remote participation and information processing speed.

In this paper we propose a decentralized, secure, and scalable framework for e-voting systems based on linkable ring signatures, a blockchain, and smart contracts. The blockchain provides a reliable repository for the data used by the e-voting system, and the support for the smart contracts that define the computation performed in each phase of a voting process. The blockchain also relieves from the need of having a centralized solution for a verified bulletin board. Then, the voting ballots are signed with linkable ring signatures, which is a cryptographic tool that allows only legal voters to sign, and detects duplicated signers without revealing identities or ballot contents. In this paper we also adopt a linkable ring signature scheme using Elliptic Curve Cryptography (ECC), which grants very short keys and signatures, hence making the framework scalable.

A. Related Work

There are a number of secure e-voting systems proposed in the literature. Many solutions are based on the Helios system [1] that is generally used as a reference due to its simple voting workflow, and its privacy and verifiability properties achieved through zero-knowledge proofs [10]. Its main drawback is that it requires a trusted central server, which is responsible for identifying voters, as well as for storing, shuffling, decrypting, and publishing ballots.

The D-DEMOS suite [5] (based on Helios [1]) takes a step further in the guarantees and properties that are ensured through Verified Secret Sharing [16], Zero Knowledge Proofs, and segregation of duties for the main tasks of the voting process. Most of the voting process is setup by an Election Authority for which only a few malicious behaviors can be detected. In our approach all the tasks necessary for the voting session setup are publicly verifiable, and any malicious behavior is detected before the vote start.

To our knowledge, in the distributed and decentralized setting, as we want to provide, there are two main relevant works. First, Liu and Wang [14] propose an e-voting system that uses a blockchain paired with blind signatures that needs to run identification of users multiple times for each vote and for each ballot. Moreover the blockchain only acts as an auditable communication mean.

Second, Lyu et al. [15] also combines a blockchain, smart contracts, and ring signatures, but there are some relevant differences. In their proposal, voters are responsible for running a distributed key generation protocol, thus involving a significant amount of processing and communication for large pools of voters, making it hard to go beyond a small number of users (the authors use up to 40 users in their tests). In contrast, as we will see, not only our design smoothly scales up to thousands of voters, but we also provide all the detail for a practical and working implementation.

Finally we discarded to adopt a traceable ring signature [9] approach like [4], since the traceability property is achieved at the cost of disclosing the signing identity.

B. Contributions

In this work we propose a framework to build e-voting systems based on a blockchain, smart contracts, and linkable ring signatures. The framework is realized as a set of protocols and tools that can be used in a wide range of e-voting scenarios. More precisely, we provide:

- A modular approach to build distributed, secure, verifiable, and scalable e-voting systems.
- A set of tools to implement such systems in different e-voting scenarios.
- A formal security analysis supporting our choice of a linkable ring signature scheme, as proposed in [13].
- A proof-of-concept implementation that can be run in any Ethereum-compliant blockchain, demonstrating the practical feasibility of our approach.

C. Paper Roadmap

In Section II we introduce the blockchain properties we leverage and our main cryptographic tool: linkable ring signatures. Section III-A presents the concrete signature scheme adopted, describing the security properties it satisfies (our full paper version provides a complete formal security analysis), while Section III-B focuses on its storage efficiency. Our e-voting framework is presented in Section IV, where we describe the involved actors and their interactions. Section V provides the details of the *identification* process and the proposed ways to guarantee *confidentiality* and *anonymity*. Next, a list of features achieved by our design is depicted in Section VI. Finally, Section VII presents our choices for the implemented proof of concept and the real deployment in univerity officers election, and Section VIII wraps up with some final conclusions.

II. Building Blocks

A. Blockchain and Smart Contracts

The chain-of-blocks data structure, paired with a Byzantine Fault Tolerant consensus results in a powerful and nifty way to realise an untamperable Distributed Ledger [8]. We will not go into details of blockchain foundations and its wide range of implementations. Indeed, we model the blockchain properties as a replicated database and a deterministic execution environment, resilient to a bounded set of Byzantine failures. We refer to the code that is stored and executed in the blockchain as Smart Contract [22] and to the activation of a piece of it as a transaction. Any modification to the system state is audited in an untamperable manner on the blockchain itself as part of the state.

B. Linkable Ring Signatures

Ring signatures are cryptographic constructions, first introduced by Rivest, Shamir and Tauman [17], allowing for the creation of signatures on behalf of a group of signers, while hiding the identity of the actual signer among the group. In [13], so-called *linkable ring signatures* were introduced as an extension to ring signatures, adding the feature of providing a public way of determining whether two signatures have actually been produced by the same signer. As pointed out in [2], these constructions are particularly suited for e-voting schemes, as linkability prevents voters from casting multiple votes while anyonymity is preserved within the group of registered voters. Formally:

Definition 1 (Linkable ring signature scheme): A *linkable ring signature scheme* is defined by four probabilistic polynomial-time algorithms (KeyGen, Sign, Verify, Link) such that:

- $\mathsf{KeyGen}(1^\ell)$, the key generation algorithm, takes as an input the security parameter ℓ and outputs a pair (pk, sk) of public (verification) and secret (signing) keys,
- $\mathsf{Sign}(1^\ell, sk_\pi, pk_1, \ldots, pk_n, m)$, the signature algorithm, takes as input the security parameter ℓ, a secret key sk_π, for some $\pi \in \{1, \ldots, n\}$, a message m (which is assumed to belong to a public message space $\mathcal{M}_\ell$) and a list of public keys $pk_1, \ldots, pk_n$, and outputs a signature σ.
- $\mathsf{Verify}(1^\ell, pk_1, \ldots, pk_n, m, \sigma)$, the verification algorithm, takes as input the security parameter ℓ, a list of public keys $pk_1, \ldots, pk_n$, a message $m \in \mathcal{M}_\ell$ and a signature σ, and outputs a bit $b \in \{0, 1\}$ (namely, 1 if the signature is recognized as valid w.r.t. the list of public keys, and 0 otherwise).
- $\mathsf{Link}(\sigma_1, \sigma_2, m_1, m_2)$, the linking algorithm, takes as an input two signatures, σ_1, σ_2, and two messages, m_1, m_2, and outputs a bit $\beta \in \{0, 1\}$, indicating whether the two signatures have been produced by the same user or not.

There are two obvious correctness requirements to add to the above definition imposing that, indeed, properly constructed signatures can be verified and linked as intended. Furthermore, our usage of a linkable signature scheme requires that certain security properties are fulfilled, which we informally list below:

- *Unforgeability*: only those belonging to the designed group of voters can produce a valid signature (which will be validated with respect to the election census).
- *Anonymity:* the actual identity of a signer remains hidden within the signing group.
- *Linkability:* different uses of the same secret key can be identified through the public information.
- *Non-framability:* it is not possible for an adversary to interfere with the correctness of linkability, even if he/she is able to corrupt (a polynomial number of) group members. Namely, even seeing a number of signatures of a prescribed honest user, and controlling some legitimate signers, he/she should not be able to produce a new signature that will link with the honest user without his/her secret key.[1]

[1]Deviating from [13] we work with the (stronger, and more realistic) definition from [2] where the case of colluding malicious users is captured, as the adversary is allowed to query for (a polynomial number of) secret keys.

In our full version paper [19] we give formal definitions and security proofs for the concrete instantiation of the linkable ring signature scheme we use in our design, which we describe in the next section and have implemented in our proof of concept (see Section VII). While we follow the approach of [13], it is worth mentioning two main differences in our security analysis. First, we allow for corruptions in the unforgeability definition, which are not considered in [13]. This is particularly relevant for our use case, as a collusion of malicious users of the system should indeed not be able to produce a valid signature that can be verified using a set of public keys from honest users. Second, we include non-framability as a desirable property, which in particular for our use case implies that a collusion of voters may not produce a valid signature (e.g., vote) that can be linked with another (honest) user, thus preventing him/her from casting a vote.

III. Proposed Linkable Ring Signature Scheme

A. Signature Scheme

We describe now our modification of the proposal in [13] based on the elliptic curve cryptography. Once the security parameter ℓ is fixed, we assume there is a public directory specifying a cyclic group E of order g within the group of points of an elliptic curve over a finite field $\mathbb{F}_p$ of size p (g, p are polynomial in ℓ). Moreover, there is also a fixed generator G for E that we assume publicly known. Furthermore, two different (public) hash functions H and $H2P$ are chosen and made public:

$$H : \{0,1\}^* \to \mathbb{F}_p,\ H2P : \{0,1\}^* \to E.$$

With these ingredients, a linkable ring signature scheme can be defined as follows:

Key Generation. $\mathsf{KeyGen}(1^\ell)$ takes as an input the security parameter ℓ and selects uniformly at random an integer sk within $\{1, \ldots, g-1\}$. Then, it sets $pk = sk \cdot G$ and outputs the resulting pair (pk, sk) of public (verification) and secret (signing) keys.

Signature Generation. $\mathsf{Sign}(1^\ell, sk_\pi, pk_1, \ldots, pk_n, m)$, given a public key tuple $PK_n = (pk_1, \ldots, pk_n)$, a pair (sk_π, pk_π), where $\pi \in \{1, \ldots, n\}$, starts by setting

$$L = H2P(HPK(PK_n)),\ T = sk_\pi \cdot L.$$

Here, HPK is a hashing procedure (that could actually be replaced by others, as in the generic construction of [13]) defined for public key tuples as follows. Given

$$PK_i = (pk_1, ..., pk_i) \qquad 0 < i \leq n,$$

we set ($||$ stands for the string concatenation operator),

$$HPK(PK_i) := \begin{cases} H(pk_1) & \text{if } i = 1 \\ H(HPK(PK_{i-1})||pk_i) & \text{if } i > 1. \end{cases}$$

Let us denote by $x \in_R X$ the selection, uniformly at random, of an element x from the set X. Values $s_1, \ldots, s_n$ are computed, starting from index π as follows:

$$u \in_R \{1, \ldots, g-1\},\ s'_\pi \in_R \{1, \ldots, g-1\}$$

$$A_\pi = s'_\pi \cdot G + u \cdot pk_\pi,\ B_\pi = s'_\pi \cdot L + u \cdot T$$

$$c_\pi = H(m||T||A_\pi||B_\pi)).$$

while for coefficients $i = \pi + 1, ..., n, 1, ..., \pi - 1$:

$$s_i \in_R \{1, \ldots, g-1\},\ A_i = s_i \cdot G + c_{i-1} \cdot pk_i$$

$$B_i = s_i \cdot L + c_{i-1} \cdot T,\ c_i = H(m||T||A_i||B_i)).$$

Finally, s_π is computed as $s_\pi = s'_\pi + sk_\pi(u - c_{\pi-1})$ and $c = c_n$ (All computations in the subindexes i above are done $mod\, p$). The output signature σ is defined as $\sigma = (PK_n, T, s_1, \ldots, s_n, c)$.

Signature Verification. $\mathsf{Verify}(1^\ell, pk_1, \ldots, pk_n, m, \sigma)$ first performs the following computations from the input values m, PK_n and σ

$$L = H2P(HPK(PK_n)),\ c_0 = c$$

. Furthermore, it builds, for $i = 1, ..., n$,

$$A_i = s_i \cdot G + c_{i-1} \cdot pk_i,\ B_i = s_i \cdot L + c_{i-1} \cdot T,$$

$$c_i = H(m||T||A_i||B_i).$$

Finally, if c_n equals c_0, it outputs 1. Otherwise, it outputs 0 and rejects the signature as invalid.

Linkability Check. $\mathsf{Link}(\sigma_1, \sigma_2, m_1, m_2)$, given

$$\sigma_1 = (PK_n, T_1, s_1, \ldots, s_n, c),\ \sigma_2 = (PK_n, T_2, \hat{s}_1, \ldots, \hat{s}_n, \hat{c}),$$

outputs 1 if an only if $T_1 = T_2$.

We are able to formally assess the following result:

Theorem 1: The ring signature scheme presented attains unforgeability, anonymity, linkability, and non-framability in the random oracle model[2] and under the DDH assumption in the underlying cyclic group E generated by G.

B. Efficiency (storage)

Our choice of an ECC linkable ring signature scheme is due to economy in keys and signatures sizes. Assuming private keys of 256 bits (32 bytes), Table I summarizes the space required for public keys and signatures compared to the same implementation in a group of prime order (reaching the same security level as with ECC).

We also assume that the list of public keys PK_n does not change once registration ends, so it can be saved once for all the signatures referring to the whole list. For the ECC case, it is useful for smart contract computations to have public keys (that are points in the curve) saved in an uncompressed form, so both coordinates are stored. Hence, for n voters, the length of the signature is the tag T length, n times the ring coefficient s_i length, plus the anchor coefficient c_n length. For the ECC case, in bytes, the signature length is $64+32(n+1) = 32(n+3)$ (because T is an elliptic curve point, while the coefficients have the same size as the private key); the public key is an elliptic curve point, so its length is $64 \cdot n = 32 \cdot 2n$ bytes.

[2]proofs in the *random oracle model* work under the assumption that the hash functions involved behave as idealized/truly random functions (see [3])

TABLE I
STORAGE REQUIREMENTS IN BYTES OF LINKABLE RING SIGNATURES OVER ELLIPTIC CURVES (256 BIT SECRET KEY) VERSUS THE ORIGINAL PROPOSAL FROM [13] (3072 BIT SECRET KEY)

Signers	ECC PKs	ECC Sig	non-ECC PKs	non-ECC Sig
10	640	416	3840	4608
100	6400	3296	38400	39168
1000	64000	32096	384000	384768

Using cyclic groups over finite fields (i.e., which we call non-ECC case), both T and the public keys belong to the same group as the private keys, so each signature needs a space of $384 + 384(n+1) = 384(n+2)$ bytes, while storing the ring of public keys requires $384 \cdot n$ bytes.

Public key's ring size in the ECC case is roughly one fifth of the non-ECC case, but the most important saving in storage comes for signature sizes: the ECC signature length is less than one tenth of the non-ECC case. As we will see in Section IV-B, a signature is saved in the blockchain per each ballot, so the saving in space with the ECC approach is huge.

IV. VOTING FRAMEWORK

This section presents the design of the proposed system. We start by defining the different actors involved in its construction. Then, we will move to describe their interactions. In [19] we report the sequence diagrams that summarize the voting protocol.

A. Actors

The voting process is divided into phases, in which different actors play specific roles. Each actor is intended to be an independent entity for simplicity, but nothing prevents an actor from being a set of cooperating subjects, leveraging blockchain and smart contracts capabilities or, as an alternative, legal contracts or other constraints. We will mention related scenarios in the subsequent Section V.

a) Voter: A voter is the entity engaged in the voting process with the aim to express his/her vote. Voters own enough information for being correctly identified (Line 1 Alg. 1), as well as to store and manage secret data linked to their digital identity. It is their own responsibility to make correct computations following the scheme specification in order to generate and submit a valid ballot.

Algorithm 1 Code for Voter

1: **Init:** $IDData \leftarrow$ Voter Identification Data
2: **Init:** $(pk, sk) \leftarrow$ KeyGen()
3: **function** SIGNUP( )
4: IdentityManager.SIGNUP($IDData$, pk)
5: **function** VOTE(v)
6: $PKS \leftarrow$ IDStorage.Get()
7: $ballot \leftarrow$ Encrypt(v, ConfManager.PublicKey)
8: $\sigma \leftarrow$ Sign(sk, PKS, $ballot$) ▷ Computes the ring signature
9: BallotBox.VOTE(σ, $ballot$)
10: **function** GETRESULT( )
11: **return** BallotBox.$Result$

b) Organizer: The *Organizer* is the entity that configures and selects parameters characterizing a concrete voting event, i.e., vote's properties and voters' rights. It is responsible for setting up the overall voting session.

Algorithm 2 Code for Organizer

1: **Init: deploy** ID Storage, Ballot Box, Confidentiality Manage
2: **Init: configure** vote open, vote close
3: **upon** (ConfManager.$SecretKey \neq \emptyset$) do
4: $Ballots \leftarrow$ BallotBox.Get()
5: $Result \leftarrow$ Decrypt($Ballots$, ConfManager.$SecretKey$)
6: BallotBox.SETRESULT($Result$)
7: **end upon**

c) Identity Manager: Identification is taken care of by the so-called *Identity Manager*. Its main task is to identify actors involved in the voting process and to register them (Line 2 and 4 Alg. 3) in the platform according to the policies defined by the *Organizer*. The $label$ derived from the $IDData$ works as identifier for the voter and can be useful to match the list of registered voters to some external public one.

Algorithm 3 Code for Identity Manager

1: **function** SIGNUP($IDData$, pk)
2: **if** VerifyIdentity($IDData$) **then**
3: $label \leftarrow$ LabelFromIDData($IDData$)
4: **call** IDStorage.SIGNUP(pk, $label$)

d) Confidentiality Manager: When required, a *Confidentiality Manager* is responsible to guard and distribute information necessary to encrypt and decrypt votes. This is a sensitive component, and Section V-B goes in depth about possible solutions that guarantee confidentiality with and without a *Confidentiality Manager*.

Algorithm 4 Smart Contract for Confidentiality Manager

1: **Init:** $PublicKey \leftarrow DistrPKGeneration()$
2: **Init:** $SecretKey \leftarrow \emptyset$
3: **upon** (vote close) do ▷ event fired/enabled by the blockchain
4: $SecretKey \leftarrow DistrSKGeneration()$
5: **end upon**

B. Voting Protocol

To describe the voting process we introduce two more components:

- the *ID Storage* (see Alg. 5) is the bulletin board of eligible voters.
- the *Ballot Box* (see Alg. 6) is the storage of valid ballots.

Algorithm 5 Smart Contract for ID Storage.

1: **Init:** $PublicKeys \leftarrow \emptyset$
2: **function** SIGNUP(pk, label) ▷ Only by the Identity Manager
3: **if** $\neg$ BallotBox.$active$ **then**
4: $PublicKeys \leftarrow PublicKeys || < pk, label >$

With the above ingredients, the framework phases are described as follows:

Algorithm 6 Smart Contract for Ballot Box. p is the total number of possible choices for a ballot.

1: **Init:** $choices \leftarrow \{v_0, ..., v_p\}$
2: **Init:** $active \leftarrow false$
3: **Init:** $Ballots \leftarrow \emptyset$; $Tags \leftarrow \emptyset$; $Result \leftarrow \emptyset$
4: **function** VOTE(σ, b)
5: **if** $\sigma.T \notin Tags \wedge active$ **then** ▷ The scheme Link function
6: **if** Verify(BallotBox.$PublicKeys, b, \sigma$) **then**
7: $Ballots \leftarrow Ballots \cup b$
8: **upon** (vote open) do ▷ event fired/enabled by the blockchain
9: $active \leftarrow true$
10: **end upon**
11: **upon** (vote close) do ▷ event fired/enabled by the blockchain
12: $active \leftarrow false$
13: **end upon**
14: **function** SETRESULT(R) ▷ Only by Organizer
15: $Result \leftarrow R$

1) Setup: The *Organizer* deploys and configures on the blockchain the three smart contracts that hold the public information for the voting session: *ID Storage*, *Ballot Box*, *Confidentiality Manager* (Line 1 of Alg. 2). The *Organizer* configures, in this phase, the firing for the *vote opening* and *vote closing* events. Moreover, the *Organizer* declares the tolerance to failures for the *Identity Manager* and the *Confidentiality Manager* if they adopt any threshold mechanism (K out of N discussed in Sections V-A and V-B).

2) Registration: Voters are required to sign up (Line 3 Alg. 1) to the system by providing (a) information that allows to prove their identity and (b) the public key they are going to use for the vote (Line 4 Alg. 1). This is a simplification of a typical certificate issuing procedure, since the outcome is not a signed certificate but a registration on the ID Storage contract. Only voters who sign up strictly before the vote opening will be allowed to vote.

3) Vote Opening: The *Ballot Box* is set to active (Line 9 of Alg. 6) so it begins to accept ballots and the *ID Storage* stops accepting new public keys. The vote opening/closing event reported in Algorithms 2, 5 and 6 is pointed as an external condition, since it is enforced by leveraging the blockchain and smart contract property of referring to the constant-time block issuance process. In this way, operations are constrained in respect to time. Anyway, if this property is not available in the chosen system deployment, the *Organizer* is in charge of firing those events by writing them in the blockchain.

4) Voting: Eligible voters interact with the platform in order to get the information required for voting. In order to submit a valid ballot, voters: (1) fetch identities needed for the linkable ring signature from the *ID Storage* (Line 1 Alg. 1), (2) use the format/encoding defined in the *Ballot Box* to create a ballot, encrypt it by fetching information from the *Confidentiality Manager* (Line 7 Alg. 1), (3) compute the linkable ring signature (Line 8 Alg. 1) and finally (4) anonymously send their vote to the system (Line 9 Alg. 1).

5) Vote Closing: The same considerations made for the Vote Opening hold for the Closing event.

6) Tally: Once the Voting Phase has been closed, the *Confidentiality Manager* is in charge of releasing the Secret Key paired with the Public Key published for ballot encryption (Line 4 Alg. 4). Once done, anyone, thanks to the information publicly available on the blockchain, is able to decrypt ballots and compute the final result.

V. PROTOCOL DETAILS AND DESIGN CHOICES

This section gives details on how our framework achieves *Identification*, *Confidentiality*, *Anonymity* and *Scalability* in different application scenarios.

A. Identification

Identification is a vital step in the vote startup. There are many known and well assessed procedures employed nowadays for mapping real identities to digital ones, like adopting multiple factors of authentication and/or relying on certificate authorities. We deliberately wrap any of these procedures in the $IDData$ variable (Line 1 Alg. 1). The *Identity Manager* is thus responsible for mapping the chosen identification procedures to our system. Since achieving resilience to Byzantine failures (unwanted or malicious failures and deliberate misbehavior) is one of our major goals, it is recommended to distribute the identification responsibility among a set of entities. That is, multiple entities should contribute to the *Identity Manager*'s operation and can leverage the smart contract capabilities in order to cooperate (e.g., use a K out of N threshold mechanism).

Moreover, to let failures be as independent as possible, the *Identity Manager* should implement different voters' identification procedures (that is, in our construction, implement different VerifyIdentity($IDData$) functions in Line 2 of Alg. 3).

B. Confidentiality

Our system guarantees secrecy to voters and avoids unwanted early result disclosures. To this aim, there are several concrete solutions that may be implemented. Indeed, a *Ballot box* built on the blockchain can be compared to a transparent box with unfolded paper inside and anyone being able to take note of added votes. The vote result is known step by step during the Voting Phase and, in most use cases, this is not a desired possibility because real time results may influence the voters who haven't voted yet. That is why a mean to obfuscate votes contents during the Voting Phase is required.

1) Ballot encoding and encryption: Each participating entity can acquire information about choices encoding from the *Ballot Box* (Line 1 Alg. 6). Then, we rely on the fact that the Encrypt function in Line 7 of Alg. 1 is always able to provide a different ciphertext at each execution even if encrypting the same plaintext. If encryption were deterministic, this is trivially achieved by embedding a random salt at each encryption (non-deterministic schemes, like RSA-OAEP [3], will do the job without modifications).

2) Confidentiality Manager: The *Confidentiality Manager* is the component in charge of providing the public key used to encrypt ballots during the Voting Phase. From the trustworthiness perspective, the *Confidentiality Manager* has the possibility to completely invalidate a voting session because the tally directly depends on the secret key disclosure. Hence, for an attacker, it would be sufficient to compromise the *Confidentiality Manager* to void the result of the voting session. For this reason, the responsibility of generating a key pair at the correct time is shared among a set of cooperating entities. Properly designed for use cases like ours, the Ethereum Distributed Key Generation (ETHDKG) [20] allows different entities to generate and share a key pair with a threshold mechanism of K out of N for private key disclosure. It exploits Shamir's Secret Sharing [21] and Verifiable Secret Sharing [7], and relies on the blockchain as an auditable communication mean. In particular the whole ETHDKG protocol is executed before vote opening, resulting in the master public key for ballot encryption (Line 1 Alg. 4). As soon as the vote is closed, the individual secret key shares are revealed in order to execute the Tally Phase (Line 4 Alg. 4). If K is the number of shares required to reconstruct the secret key and N the number of *Confidentiality Manager* members, the tolerance to malicious behavior is $min(K-1, N-K)$; in fact the system correctness can be undermined in two cases: (1) the secret is disclosed before the Tally Phase, so K entities disclosed earlier; or (2) the secret is never disclosed because $N - K + 1$ entities do not correctly participate in the secret reconstruction. Since both cases must be avoided, tolerance to failing entities is the minimum between the two.

C. Anonymity

Ring signatures grant anonymity, providing at the same time the required authenticity guarantees. This solves the problem of recognizing a ballot as valid without disclosing the underlying voter identity, thanks to the fact that a ring signature obfuscates the link between a signature and the related signer.

On the blockchain interaction side, in Line 9 of Alg. 1 the voter directly calls a method of a smart contract that causes a state change in the blockchain, so he/she needs a valid identity to do that. The word "valid" can have two different meanings: in a *permissioned* blockchain it means that the identity is authorized to submit transactions, while in a *permissionless* one it means that the identity has enough resources to create an acceptable transaction. Neither condition pairs well with an anonymous and scalable ballot submission. The former would mean to identify the actual user behind the identity, so voiding the linkable ring signature property. The latter would imply the voter to procure by himself/herself the needed resources, which is usually not scalable on a large non-technical voter base. Moreover, the resource (cryptocurrency) procurement should preserve anonymity. While we listed three practically feasible solutions to this problem [19], we report here the most scalable one that we also implemented.

1) Anonymization Proxy: The anonymization proxy is a component that acts as a proxy between voters and the blockchain. In practice it is responsible of taking valid votes, packing them in transactions, and submitting them to the blockchain. Multiple authorized anonymization proxies can be provided to avoid a single point of failure and trust. Voters can autonomously choose which (one or more) they prefer and they can send the ballot without being identified. If they require advanced anonymity guarantees, they can leverage any anonymization technique for ordinary web interactions (e.g. Onion routing). After the submission, they can easily check that their operations have been effectively executed on the blockchain, thanks to its transparency. On the *Organizer*'s side this has advantages, since there is no need to ensure write access to the blockchain to each voter, but only to the authorized proxies, who can also pack ballots in less transactions and optimize execution costs.

D. Scalability

The voting system should easily scale in the presence of a large base of actors. In particular, the number of voters is expected to be orders of magnitude higher than the number of all the other participating entities.

Since scalability currently is a key issue in blockchain-based systems, we take special care dealing with it and furthermore compare our design with the proposals [15, 14]; the former is based on blind signatures and the latter, like ours, on linkable ring signatures. We assume to have M voters and to impose a K_i on N_i threshold mechanisms for voters identification and K_e on N_e secret sharing threshold for ballot encryption.

In our design, in the Registration Phase, voters must successfully identify themselves with at least K_i different *identity managers*; so, a voter has at least a communication round with each one of them. Unlike [14], this process can be reused indefinitely for any subsequent voting session, even if not initially scheduled. Any subsequent vote can directly start its protocol considering this phase as already completed. This is really efficient for recurring votes with a great number of voters in common. In [14], identification and vote submission are made in the same phase, so for each voter, for each ballot submission and in each voting session, K_i identifications and blind signature computations need to be run.

About vote submission, in our proposal, voters read the ring of public keys and the encryption key from the blockchain, and they submit their ballot along with the ring signature; this translates into 1 read and 1 write in the blockchain per voter. Providing confidentiality through the ETHDKG [20] protocol requires 2 rounds of interactions by the N_e *Confidentiality Managers*, in which each of them has to write to the blockchain in both rounds (*Encryption*) in order to publish the *master public key*. In the Tally Phase it is enough that K_e *Confidentiality managers* disclose their secrets to decrypt ballots. In [15] the distributed key generation for ballot encryption is extended to all the voters. This let the system to hardly scale on a large base of voters since the distributed

secret generation requires multiple communication rounds and writes onto the blockchain.

VI. Properties and additional features

We now discuss the relevant properties achieved by our design. Our full version [19] includes and extend those reported in [14]. For each one of them we provide an informal definition and discuss why we are able to fulfill it.

1) Required properties: Required properties are those that must be granted for any use case of the e-voting system, as they encompass the minimal correctness and security prerequisites needed for any application scenario:

a) Verifiability: The *ID Storage* (Alg. 5) and the *Ballot Box* are implemented by smart contracts, leveraging the blockchain capabilities. That is, each method invocation is permanently stored in the system. All ballots remain saved and publicly visible along with their attached linkable ring signature. So, a voter, who knows which his/her own ballot is, can verify the correctness of both the ballot submission and the tally (Individual Verifiability). From the global system point of view (Universal Verifiability), every entity can read the *ID Storage* and verify the linkable ring signature, checking that all saved ballots are legitimate. Finally, the saved result (Line 6 Alg. 2) can be verified through the secret key published by the *Confidentiality Manager*. The *Organizer* saves back the result to the *Ballot Box* only as a convenience, since the secret key for decryption is already public at that moment.

b) Legitimacy: Only those who have the right to vote can take part in the voting process, and only for as long as they're granted to participate. The satisfaction of this property is directly tied to the *Identity Manager* correctness. That's why a K out of N mechanism is recommended. Each information in the *ID Storage* is provided by the (or K out of N) *Identity Manager* through signed transaction submitted to the blockchain. The label, saved along with the public key (Line 4, Alg. 5), acts as a public identifier for each voter in the list, so anyone who knows the list of allowed voters, can check it against the registered one (e.g., in our deployment in Section V, we set the label as the SHA256 of the voter's email). Beyond this, the linkable ring signature and the trusted execution environment of the blockchain ensure that only the identities in the *ID Storage* can create and submit valid ballots. In fact, signature verification is implemented in the *Ballot Box* smart contract, so double votes are detected through the linkability and non-frameability properties.

c) Neutrality: The voting outcome cannot be influenced neither by the voting process itself nor by the tools used to vote. In the Confidentiality Section V-B, we explain how our design allows for the vote result to be hidden during a voting session, in order to avoid influencing voters with already accepted ballots.

d) Auditability: The whole voting procedure must be auditable during and after the vote. In our case, this feature is provided by the blockchain, which gives the abstraction of an append-only register that allows auditing all methods' invocations on the components implemented through smart contracts (*ID Storage*, *Ballot Box*, *Confidentiality Manager*).

2) Information Security: Standard security properties, in general referred as the CIA triad [12]: confidentiality, integrity, and availability. We will refer to *data confidentiality*, *data integrity*, *system integrity*, and *availability* as defined in [11]. *Data confidentiality*, *system and data Integrity* are directly guaranteed by relying on the smart contract deployed in the blockchain as main storage for the system data. The *Availability* leverage the distributed nature of the blockchain, too, but in [19] we analyze an issue that can arise when allowing ballot replacement as coercion countermeasure.

3) Additional Features: All the vote data is stored inside the *ID Storage* and *Ballot Box* smart contracts. Each blockchain node holds a full copy of it with integrity proven by the chain-of-block data structure. In [19] we also discuss about the voters' coercion resistance, for which we can allow, as a mitigation, multiple ballot submission by the same voter.

VII. Implementation

We validate the feasibility of our proposal with a working implementation. Ring signatures and the blockchain are points of paramount importance, so our proof of concept implementation mainly focuses on them.

A. Smart Contract

We leverage the Ethereum ecosystem of software. The *ID Storage* and the *Ballot Box* are implemented in a single smart contract that stores the voting session state, voters information, and ballot data. The smart contract also implements linkable ring signature verification, so only ballots with valid signatures are actually stored. The *Organizer*, *Confidentiality Managers* and *Identity Managers* appear with their Ethereum addresses, they can hence be either individuals or other smart contracts like in the case of [20].

B. Signature Algorithm

The signature scheme described in Section III-A has been implemented in Solidity within the Ballot Box leveraging the BN256 curve since the Ethereum Virtual Machine has opcodes for the sum and multiplication on it. [6].

Moreover, in our schema, the public keys hash gets computed cumulatively at at each new added public key. This particularly suits the blockchain environment since the computation is spread across multiple transactions.

C. Proof of concept

We implemented the whole voting process by providing scripts to accomplish each phase we described for each involved actor. Confidentiality is provided through the ETHDKG [20] smart contract. The codebase [18] is in Python, except for the smart contracts that are written in Solidity, the programming language for smart contracts on the Ethereum blockchain.

The scripts can target any Web3 (the Ethereum standard RPC interface) compliant node. We also provide a ready-to-go testnet through a Docker container.

D. Real deployment

Our voting framework has been successfully deployed and used in the elections of university officers. The three distinct elections involved 428 total voters with a participation of 408 in the registration phase, and 398 in the voting phase. Although we are going to produce and publish a full report of this first use of the system, we think it is worth citing here the main characteristics of the events:

- The overall architecture was made of a blockchain, a blockchain explorer, a web server acting as *Identity Manager*, a web server acting as *anonymization proxy* for vote submission, an administration portal and a web application for voters to interact with the system.
- The signature algorithm has been implemented in plain Javascript to compute the signature in a common browser.
- Voters generated a *Voting Card* locally in their browser that included their personal keypair. The public key was sent to the *Identity Manager* that was integrated through SAML 2.0 with the university identity provider.
- We relied on a permissioned blockchain based on Hyperledger Besu that is fully Ethereum compliant deployed across different independent administrative entities.
- During the voting phase, voters selected their chosen candidate, uploaded the *Voting Card* to the browser that computed the linkable ring signature and encrypted the ballot choice. These were both sent to the *anonymization proxy* that redirected the signature and ballot to the blockchain. The transaction hash and a link to the blockchain explorer was given back to the voter as receipt.
- The Ethereum DKG [20] was not integrated, yet, but the administration portal generated an RSA key pair and published the public key to the *Ballot Box*. Once the voting phase was over, the members of the electoral committee requested ballots decryption on the administration portal. Decryption was computed as soon as all the members of the committee asked the decryption with their personal accounts.

VIII. Conclusion

The presented framework leverages the blockchain and linkable ring signature capabilities to create a comprehensive solution for e-voting in a wide range of scenarios. We show how our design can address the requirements of a voting system and provide a collection of additional features. In particular our proposal fits voting scenarios where coercion resistance and receipt freeness are not a key concern. Indeed, our system cannot be considered receipt-free because of its individual verifiability property. Our plan is to integrate the Ethereum DKG to guarantee confidentiality and neutrality to the platform with a decentralized approach.

IX. Acknowledgements

We are grateful to Javier Herranz for fruitful discussions and pointers to relevant literature on ring signature schemes. This work is partially supported by the NATO Science for Peace and Security program (grant G5448), by the Spanish Ministerio de Ciencia e Innovación (MICINN) grant PID2019-109379RB-I00, and by the Regional Government of Madrid (CM) grant EdgeData-CM (P2018/TCS4499) cofounded by FSE & FEDER. This paper is part of the R&D&I project PID2019-109805RB-I00 funded by MCIN/AEI/10.13039/501100011033 (ECID).

References

[1] B. Adida. "Helios: Web-based Open-Audit Voting". In: *Proceedings of the 17th USENIX Security Symposium*. 2008, pp. 335–348.

[2] M. Backes et al. "Ring Signatures: Logarithmic-Size, No Setup - from Standard Assumptions". In: *Advances in Cryptology - EUROCRYPT 2019*. Vol. 11478. LNCS. Springer, 2019, pp. 281–311.

[3] M. Bellare and P. Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 1993, pp. 62–73. ISBN: 0897916298.

[4] Christian Cachin et al. "Anonymity preserving Byzantine vector consensus". In: *European Symposium on Research in Computer Security*. Springer. 2020, pp. 133–152.

[5] N. Chondros et al. "Distributed, end-to-end verifiable, and privacy-preserving internet voting systems". In: *Comput. Secur.* 83 (2019), pp. 268–299. DOI: 10.1016/j.cose.2019.03.001. URL: https://doi.org/10.1016/j.cose.2019.03.001.

[6] *EIP-1108: Reduce alt_bn128 precompile gas costs*. https://eips.ethereum.org/EIPS/eip-1108. Accessed: 2021-11-05.

[7] P. Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *28th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1987, pp. 427–437.

[8] Antonio Fernández Anta et al., eds. *Principles of Blockchain Systems*. Morgan & Claypool Publishers, 2021.

[9] Eiichiro Fujisaki and Koutarou Suzuki. "Traceable ring signature". In: *International Workshop on Public Key Cryptography*. Springer. 2007, pp. 181–200.

[10] S. Goldwasser, S. Micali, and C. Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM J. Comput.* 18.1 (1989), pp. 186–208.

[11] *Internet Security Glossary, Version 2*. https://tools.ietf.org/pdf/rfc4949.pdf. Accessed: 2021-04-13.

[12] *ISO/IEC 27000:2018*. https://www.iso.org/standard/73906.html. Accessed: 2021-11-05.

[13] J. K. Liu, V. K. Wei, and D. S. Wong. "Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups (Extended Abstract)". In: *Information Security and Privacy: 9th Australasian Conference, ACISP 2004*. Vol. 3108. LNCS. Springer, 2004, pp. 325–335.

[14] Y. Liu and Q. Wang. "An E-voting Protocol Based on Blockchain". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1043.

[15] J. Lyu et al. "A Secure Decentralized Trustless E-Voting System Based on Smart Contract". In: *Proceedings of TrustCom/BigDataSE 2019*. IEEE, 2019, pp. 570–577.

[16] T. P. Pedersen. "A Threshold Cryptosystem without a Trusted Party (Extended Abstract)". In: *Advances in Cryptology - EUROCRYPT '91*. Vol. 547. LNCS. Springer, 1991, pp. 522–526.

[17] R. L. Rivest, A. Shamir, and Y. Tauman. "How to Leak a Secret". In: *Advances in Cryptology - ASIACRYPT 2001*. Vol. 2248. LNCS. Springer, 2001, pp. 552–565.

[18] A. Russo. *Proof of concept Repository*. https://github.com/russanto/chirotonia_poc. Accessed: 2021-08-01.

[19] Antonio Russo et al. *Chirotonia: A Scalable and Secure e-Voting Framework based on Blockchains and Linkable Ring Signatures*. 2021. arXiv: 2111.02257 [cs.CR].

[20] P. Schindler et al. "ETHDKG: Distributed Key Generation with Ethereum Smart Contracts". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 985.

[21] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[22] N. Szabo. "Formalizing and Securing Relationships on Public Networks". In: *First Monday* 2.9 (1997).