E-voting: state of the art

Vincenzo Iovino
Aragon ZK Research
Reviewed by Ivan Visconti
University of Salerno

6th April 2023

1 Introduction

E-voting is becoming popular and in recent years several researchers, companies and projects have proposed solutions which target and are optimized for various scenarios such as political elections, board of directors elections and Decentralized Autononomous Organizations (DAOs).

In this document we will provide a background on the most popular cryptographic techniques and the desirable properties for e-voting. Finally, we will present e-voting solutions that are specifically suitable for web3, highlighting the main issues that arise in that setting.

2 Tools used in e-voting in a nutshell

In this section, the most important cryptographic tools used in e-voting are discussed.

2.1 TLS/HTTPS

Most e-voting protocols require the voters to privately communicate with a server or to send private information to other parties. Standard TCP/IP communication does not guarantee secrecy and authenticity of information, but TLS and HTTPS with certification authorities (see also 2.4) installed in modern browsers are a standard tool to deal with that. In addition, TLS and HTTPS can be also used by servers that require client authentication.

2.2 Bulletin board

Most e-voting protocols assume the availability of a bulletin board where the ballot of each voter and other information can be posted in a way that no malicious party can delete or modify the content. The bulletin board represents

the source of truth of the election. One way to implement decentralized bulletin boards is using blockchains.

2.3 Cryptographic hash functions

Hash functions [1, 2] allow compression of an arbitrarily long input into a short string of fixed size called the *digest*. A hash function is secure when it is hard to find a preimage to a value in the range of the function (one-wayness) and when it is hard to find two inputs that map to the same output (collision-resistance). Under the term "cryptographic hash function", one considers heuristically that a specific collision-resistant hash function behaves like a random oracle, that is as an ideal random function. Popular hash functions include SHA256, KeccaK, Pedersen and Poseidon/Poseidon2 [3, 4, 5, 6]. It is useful to mention that Pedersen is collision-resistant and SNARK-friendly but is not a random oracle so one should be careful in choosing the correct hash function based on the security requirements.

2.4 Digital signatures

Digital signatures (DS) [1, 2] are the digital version of handwritten signatures. A valid digital signature gives a recipient a guarantee that the message was created by a known sender, and that the message was not altered in transit. It should be noted that this isn't foolproof and can be defeated by a replay attack. The sender is identified by a verification key, also called a public key, and the sender is able to sign a message using their own secret key. Digital signatures allow for achieving *eligibility* in e-voting schemes: voters sign their ballots so anyone can check that such ballots are cast by eligible voters, voters who have the right to cast a vote.

The set of PKs of the eligible voters should be published and represents the so called *census* of the election.

A DS scheme should satisfy *unforgeability* meaning that an attacker is not able to produce signatures for messages which the attacker did not observe before. Popular digital signature schemes include ECDSA, Schnorr, RSA and BLS. The BLS signature scheme enjoys some special aggregation properties that have been exploited by the LOE drand service [7] to compute a random beacon in a distributed way.

Digital signatures are used to define public-key certificates that in turn are used in combination with TLS/HTTPS protocols to establish private and authenticated channels.

2.5 Public-key encryption

Public-key encryption (PKE) [1, 2] is one of the most basic cryptographic primitives. PKE allows Alice to send an encrypted message to Bob, a *ciphertext* CT. Alice encrypts her message under the *public-key* (PK) Pk of Bob using a procedure Enc and Bob can decrypt the ciphertext using the corresponding *secret*

key (SK) Sk by means of a procedure $\mathsf{Decrypt}$. Both Pk and Sk are computed by means of a procedure Setup. Intuitively, only Bob should be able to decrypt and a malicious Eve who intercepts the ciphertext should not be able to leak information in the encrypted message, excluding trivial information such as the length of the encrypted message. The latter property is equivalent to saying that no attacker, given the PK, can distinguish with non-negligible probability whether a ciphertext encrypts one of two equal length messages of its choice.

2.5.1 Homomorphic public-key encryption

An example of a popular PKE scheme is ElGamal which is a homomorphic encryption scheme[1, 2]. In a homomorphic encryption scheme, anyone, given two ciphertexts $\mathsf{CT}_1, \mathsf{CT}_2$ encrypting respectively integers v_1, v_2 with randomness r_1, r_2, can produce a ciphertext CT encrypting $m_1 + m_2$ with randomness $r_1 + r_2$. The ElGamal encryption scheme [1, 2] satisfies this property when the message space is small. After applying many homomorphic operations, a ciphertext could be the encryption of an integer n and then the decryption procedure is efficient only if n is sufficiently small, $< 2^{20}$. Other protocols like Paillier's encryption scheme [8] do not suffer from this issue and allow encryption of large integers, with decryption being time independent from the bit length of the tally.

2.6 Commitments

A commitment scheme [1, 2] is a protocol between a Sender and a Receiver. The Sender holds a message m and, in the first phase, it picks a randomness r and then "encodes" the message using r and sends the encoding (a commitment to m) to the Receiver. In the second phase, the Sender opens the commitment by sending to the Receiver m and the randomness r so that the Receiver can verify that the content of the commitment was the message m. A commitment scheme should satisfy two security properties:

- Hiding: Receiving a commitment to a message m should give no information to the Receiver about m.
- Binding: The Sender cannot "cheat" in the second phase and convince the Receiver that the content of the commitment was a message m', different from m.

Commitments schemes can be perfectly hiding when the receiver has unbounded time and perfectly binding when the sender has unbounded time.

Pedersen and Poseidon [5] commitments are SNARK-friendly. Moreover, the SHA256 hash function can be used to construct a commitment that is secure in the RO model.

2.7 Zero-Knowledge proofs

Consider a toy e-voting scheme in which a voter V encrypts her preference under the public key Pk_A of an authority A and signs her ciphertext using her secret key corresponding to her public key Pk_V The authority checks the digital signatures of each ciphertext for eligibility and then privately decrypts each ciphertext and computes the tally of the election.

This toy example suffers a major problem. The authority A can completely subvert the result of the election by announcing a different result. In other words, the result announced by A may not correspond to the true result. Furthermore, in a referendum, voters can submit invalid preferences different from $\{0, 1, \star\}$ with the purpose of giving more weight to one of the two options.

This problem is addressed using proofs. The idea is that both the voters and the authority A can add some proofs that demonstrate they made the computation honestly; the authority decrypts correctly and the voters encrypt a vote in the valid range. It is trivial to see that these proofs exist: the voters could reveal the randomness used to encrypt and A could reveal the SK. However, these trivial proofs would reveal the individual preferences of each voters and so this option does not solve the problem. Specifically, we desire zero-knowledge proofs [9], which can be verified without leaking additional information about the statement being proven.

In particular, we will need non-interactive zero-knowledge proofs, that is proofs which are publicly verifiable by everybody, not only by those who participated in the voting process. Efficient NIZK proofs [10] exist for several useful relations, for example those which have existing efficient sigma protocols, including DLog, DDH, and Pedersen commitments. Moreover, there exist efficient NIZKs that are also succinct and efficiently verifiable, like in SNARKs [11, 12, 13].

Several SNARKs are based on the Common Reference String (CRS) model, where a trusted party generates a string called the CRS. The CRS is used by both provers and verifiers to compute and verify proofs. The party who computes the CRS has to be trusted to delete the randomness used to generate the CRS; this randomness can be exploited to compute proofs for false statements. Alternatively, a ceremony has to be run among several parties to distribute the trust of the computation of the CRS so that at the protocol's end an attacker cannot prove a false statement without colluding with the majority of parties. Observe that usually the CRS is long and depends on the length of the circuits whose computation is to be a verifier, but in most SNARKs the verifier is only given a short key, called the verification key, so the verification of the proofs can be fast. Moreover, unlike [11, 12], in Plonk [13] the CRS is independent of the circuits to be computed, it is *universal*. There exists also succinct NIZK proofs that are based in the RO model and avoid the CRS model [14].

SNARKs in general may suffer from malleability attacks. In a malleability attack, an adversary, observing an accepted proof for statement x computed with a witness y, is able to produce another accepted proof of a related, but different, statement x', even without knowing y, or any other witness to x.

When non-malleability is a desired property, simulation-extractable SNARKs must be used [15].

2.8 Secret sharing

Secret sharing [16] is a protocol run by a set of participants to share a secret string s in a way that no sufficiently large subset of participants can recover the secret. Precisely, in a (t, n)-secret sharing ((t, n)-SS) protocol a set of n participants can interact together in a *sharing phase* to distribute a share to each participant $i \in [n]$. The set of shares induces a secret s and we say that the participants share s. The security guarantees that no subset of t or less participants can recover the secret, but any subset of $t + 1$ participants can recover the secret in a *reconstruction phase*.

Some secret sharing protocols may require a trusted dealer, a special participant that distributes the share of the secret s to different participants and then deletes the randomness. In Shamir's SS, the dealer distributes to participants the evaluation of a degree t polynomial with constant term equal to the secret s to be shared; any subset of $t + 1$ participants can then reconstruct the secret s by polynomial interpolation.

A problem with Shamir's SS and SS in general is that the dealer could distribute the shares in a malicious way, for example so that different sets of $t + 1$ participants reconstruct a different secret. To resolve this issue, Verifiable SS (VSS) [17] is required.

3 Popular approaches to e-Voting

In this Section we review the most popular approaches to construct e-voting protocols. We skip the techniques to mitigate coercion attacks in light of the fact that currently these approaches are not practically instantiated for web3 voting yet. For definitions about e-voting we defer to another internal document by Roger Baig [18].

3.1 Threshold and homomorphic encryption-based protocols

Recall the toy e-voting scheme defined in Section 2.7. The problem of that protocol is the single authority A can decrypt each individual ciphertext and violate the privacy of each voter. This issue can be mitigated using a (t, n)-SS (cf. Section 2.8). The SK of the encryption scheme will be shared by a set of n participants in a way that no subset t or less participants can break the privacy of the voters but any subset of $t + 1$ participants can compute the tally adding ZK proofs of correctness. This can be done using a trusted dealer or employing protocols for Distributed Key Generation (DKG) [19]. Observe that the authorities, i.e. the parties sharing the SK, should not be able to decrypt the

ciphertexts individually because ciphertexts are associated with digital signatures of the voters, so the authorities should not be able to publicly disclose the preference encrypted by each ciphertext. Theoretically, it is possible to provide a proof of correctness of the tally with respect to all posted ciphertexts, that is a ZK proof that the ciphertexts $\mathsf{CT}_1, ..., \mathsf{CT}_k$ publicly posted in some bulletin board by eligible voters are such that they decrypt to plaintexts $v_1, ..., v_k$, which encode valid voting options, and $T = f(v_1, \ldots, v_k)$, where f is the tally function. However, this proof can be expensive, hence a popular methodology [20] to gain efficiency while preserving privacy is to assume that the voters can cast only binary preferences representing YES or NO.

Now, let's use an e-voting protocol which employs a homomorphic encryption scheme. The ciphertexts $\mathsf{CT}_1, ..., \mathsf{CT}_k$ posted on the bulletin board by eligible voters can be homomorphically tallied to compute a single ciphertext CT that contains the sum $T = \sum_{i \in [k]} m_i$ of each voter's preference, and then a single proof of correct decryption that CT decrypts to T is provided by the authorities. This can be done without requiring the participants to reconstruct the SK, and can be implemented using variants of the ElGamal encryption scheme.

Observe that for this approach to work we must also assume that each voter submits a proof that his ciphertext encrypts 0 or 1.

3.2 Shuffle-based protocols

The problem with the approach of Section 3.1 is that it can only be implemented efficiently when the voting options are tightly limited, for instance a binary decision of YES or NO. When the voting options are more complex it consequently becomes more complex to take advantage of homomorphic encryption. Moreover, homomorphic tallying cannot recover the individual input plaintexts, which is required by election laws in some countries and in the case of write-in votes. In that case, another popular approach based on verifiable shuffles or mix-nets [21, 22] can be described.

Let's assume that the ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ respectively encrypt the preferences $v_1, \ldots, v_n$ of the voters $V_1, \ldots, V_N$ and are published on the Public Bulletin Board (PBB) along with the public-key/signature pairs $(\mathsf{Pk}_1, \sigma_1), \ldots, (\mathsf{Pk}_n, \sigma_n))$. We also assume that the ciphertexts are generated using a threshold PKE scheme that is also *re-randomizable* per ElGamal or any other threshold PKE scheme that is re-randomizable. The public-keys and the signatures can be used to verify that such ballots come from eligible voters, then the following protocol can be applied.

We assume that there are d *mixers* $M_1, \ldots, M_d$. The mixers may also be the same authorities which perform threshold decryption. Initially, the list L_1 is filled with the ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ of the n voters encrypting respective votes $v_1, \ldots, v_n$. For $i = 1, \ldots, d$, the mixer M_i (that is also the i-th authority) performs the following operations:

- The mixer M_i re-randomizes each ciphertext $\mathsf{CT}_j, j \in [n]$ using a new random value $r_{i,j}$ to compute a new ciphertext CT'_j encrypting the same

plaintext v_j and sets the list $L'_i = (\mathsf{CT}'_1, \ldots, \mathsf{CT}'_n)$.

- Choose a random permutation γ_i from $[n]$ to $[n]$ and compute the list $L''_i = (\mathsf{CT}'_{\gamma_1(1)}, \ldots, \mathsf{CT}'_{\gamma_1(n)})$. Publish L''_i on the PBB. Set L_{i+1} to L''_i. The new list L_{i+1} will be the input to the next mixer.

In the end, the authorities can decrypt each ciphertext in the list L_d to get the n individual preferences in a permuted way and from this the tally can be computed.

Observe that even if an attacker knew that $\mathsf{CT}_{1,j}, j \in [n]$ comes from a certain IP address X, the attacker will not know which is the ciphertext in the list L_d that encrypts v_j, unless the attacker gets the randomness values $r_{i,j}, i = 1, \ldots, d$ used to re-randomize the original ciphertext and the permutations $\gamma_1, \ldots, \gamma_d$ from all the mixers $M_1, \ldots, M_d$. As a byproduct, if the attacker is not able to collude with all of the mixers, the attacker will not be able to de-anonymize the voters. Existing mixers in the context of web3 are provided by TornadoCash and Coinjoin which are based on a different logic than the approach demonstrated above, see [23] for details on how coin mixers work and their applications to e-voting.

3.3 Approaches that achieve everlasting privacy

In some situations it may be required that a voter's preference can never be disclosed. This property is called *everlasting privacy*.

This can be achieved using perfectly hiding commitments that are homomorphic instead of public-key encryption following the approach of [24]. The generic i-th voter can compute a commitment C_i to her preference v_i with randomness r_i and post it on the bulletin board. The voter acts as a dealer in a (t, k)-VSS protocol with authorities $1, \ldots, k$ with secrets r_i. At the end of the protocol authority, $j \in [k]$ gets share $r_{i,j}$. For each voter, $t + 1$ authorities can homomorphically tally the commitments $C_1, \ldots, C_k$ with respective randomness $r_1, \ldots, r_k$ to compute a commitment C to $V = \sum_{i \in [k]} v_i$ with randomness $R = \sum_{i \in [k]} r_i$.

These authorities then perform a VSS-reconstruction phase in the following way:

For each $j \in [n]$ they compute $R_j = \sum_{i \in [k]} r_{i,j}$, and then, by the properties of the VSS protocol they use, it holds that the values R_j's represent shares of the secret R.

Hence the authorities can perform a VSS-reconstruction phase to recover R, verify that C is a commitment to V with randomness R, and they can publicly post (R, V) on the bulletin board so that anyone can verify that C was a commitment to preference V and this implies that the sum of the voters' options is actually V. The verification that C is a commitment to V with randomness R can be completed by brute force when the space of voting options is small, as it is done for homomorphic encryption.

The privacy will be unconditional under the assumption that after the election the authorities delete their memory. The commitment itself is perfectly hiding so leaks no information about the preference.

Beyond VSS, for verifiability we additionally request the voters to add digital signatures of their commitments.

3.4 Protocols based on membership proofs

Another popular approach, especially in the web3 domain, is the following.

One assumes that the census of the eligible voters is represented by a vector commitment [25] T whose values contain the public-keys of the voters for a digital signature scheme. A traditional choice for the vector commitment is Merkle trees but nowadays new and more efficient vector commitment schemes are available and can be used for this task, for example Verkle trees. In an election identified by a string id, a voter with public-key Pk and corresponding secret key Sk, submits to the bulletin board the voting option v in the clear, a value $h = H(\mathsf{Sk}, \mathsf{id})$, where H is a cryptographic hash function such as SHA256, and a ZK proof of the following fact: there exists a signature σ of v with respect to a public-key Pk that is contained in the vector commitment T.

The value h is called *nullifier* and it is used to prevent a voter casting their vote twice. If this occurred there would be two equal nullifiers h_1, h_2 appearing on the bulletin board and the corresponding votes will be discarded. Since the votes appear on the bulletin board in the clear, the tally can be easily computed. Here, we suppose that the voters should send their data to the bulletin board using TOR or a similar tool to hide their identity, otherwise the administrator or the nodes running the bulletin board can easily link a voting option to a specific IP address.

This design has been already validated, in a different context, by the ZCash system [26].

3.4.1 Membership proofs based on ring and group signatures

The content of this section is based on a previously published technical report by Rebekah Mercer [27].

Membership proofs can be obtained not only via SNARKs but also via ring signatures like in DualDory [28]. DualDory is an elliptic curve discrete log-based scheme with pairings-based pre-processing. This scheme was introduced in mid-2022, giving an advantage that it is implementable and deployable today, with no future research needed. We would simply need to implement the protocol in Rust, and the verification section in Solidity, to determine how concretely expensive it is. DualDory has O(n) voter computation time, signatures of size O(log(n)), the verification has a O(n) pre-processing stage (one per census), and then an additional O(log(n)) verification computation is required per vote.

Group signatures, one form of which is known as BBS signatures, were introduced by Boneh, Boyen, and Shasham in 2004 [29]. In group signatures the census is implicitly defined, there is no explicit list of members authorized by a

single or set of authorities but there is group administrator with the capability of authorizing people to sign on behalf the group.

BBS in particular is a pairings-based scheme, with constant size signatures, and proving and verification complexity. This means that the signature size, proving and verification computations do not grow with the number of voters, allowing indefinite scaling and therefore supporting censuses the size of the global population. This scheme is currently going through a standardisation process, as it is being used in companies that are trying to provide privacy-preserving proof-of-identity solutions. Due to the existence of these prior implementations, it should be straightforward to implement. The issue with this solution is that it is currently linkable only to an authority. Future research could determine whether this authority can be easily distributed or replaced with publicly verifiable equations.

3.5 Protocols based on blind signatures

In a blind signature protocol two parties P_1, P_2 interact together in the following way. At the beginning of the interaction, P_1 has a private message m and a public-key Pk for a digital signature scheme, and P_2 has the corresponding secret-key Sk. At the end of the interaction P_1 gets a signature of m that is verified under Pk in a way that P_2 does not leak any information about m and P_1 does not leak any information about Sk.

This protocol can be used by a voter to request a digital signature of his voting option that later on can be posted on the bulletin board. In this way, anyone can check the eligibility of the voter but the authority cannot link any pair of voting option and signature to a specific voter. The authority must be decentralized, as in most e-voting protocols, to prevent the authority easily generating signatures of voting options. Examples of blind signatures are the RSA's and Schnorr's schemes [30]; the latter scheme has been subject to recent attacks [31].

4 E-voting in the web3 domain

In the web3 domain, e-voting is usually done not for political reasons, but to take decisions regarding financial decisions on the so called *Decentralized Autonomous Organizations* (DAO,in short). There are some peculiarities and features that are specific to the web3 domain.

- Scalability: In Ethereum any computation costs gas and this has to be minimized.
- Crypto primitives optimized for web3: Ethereum offers a limited number of cryptographic operations. Technically it is possible to implement any computer program in the Ethereum Virtual Machine (EVM) but it is preferable to make use of cryptographic tools that are optimized for

Ethereum and specifically the *precompiles*. For example, only some specific bilinear groups are available on Ethereum as precompiles such as the bn128 curve.

- Token voting: In the context of web3, elections may be based on, e.g., NFT ownership. Here the census should be implicitly linked to the NFT ownership data stored on Ethereum.

- Hiding voting power: In voting based on NFT ownership, a desirable property is that it should not be possible to infer whether a given voter owns a large amount of tokens, hence "whales" should not be identifiable. In practice it may be known that Bob is the major token holder so knowing that a particular ballot was cast by someone with a very high amount of tokens implies that the relevant ballot was likely cast by Bob.

- Fairness: E-voting in the web3 domain is usually performed using proofs of membership as described in Section 3.4. If e-voting protocols based on simple membership proofs, voter preferences sent to the smart contract are publicly visible. This poses the problem of *fairness*: the voters who have not yet cast a ballot can be influenced by the partial tally that is publicly visible on the blockchain.

- Equity of costs: The gas fees are paid by whoever submits a transaction and this has consequences. In traditional e-voting systems, computing a tally at the end of the election carries a negligible cost. In Ethereum, however, one has to prefer solutions in which the cost of tallying the result is fairly divided among the voters. If, for instance, a system were such that the last voter needs to compute the tally on-chain in an expensive way, his transaction would cost more gas than for previous voters and this would raise an equity issue.

- Storage: The cost of storage in Ethereum is very high. Preference must therefore be given to solutions in which only a short digest is published *on-chain*,i.e. on Ethereum, and the full data is stored *off-chain*.

  The cost of storage also poses some equity problems. In a system where each voter needs to add an element to a set, if this cost grows linearly with the number of elements in the set and the number of voters is very high ($> 10k$), the last voters would pay more than the first voters.

5 E-voting Projects

Recently, the Nouns DAO announced a competition [32] to select the 3 best researched solutions to private voting for their DAO. In this section we highlight some e-voting projects both for web3 and traditional e-voting, giving a focus on some of the most interesting solutions proposed for the Nouns' call.

- Proposal of Aragon ZK Research (AZKR) and Aztec (AZ) for the Nouns DAO [32]: AZKR, in partnership with AZ, proposes an anonymous voting scheme based on membership proofs. There are several main challenges that are adopted and proposed by AZKR for the web3 setting. First of all, the census should be implicitly given by NFT ownership and should not be computed by a trusted party. In order to guarantee that, the AZKR+AZ proposal will employ a SNARK proof of Ethereum storage. The smart contract will take a snapshot of the Ethereum storage at a determined point, for example when the voting period starts. This snapshot is represented in the form of the root of a Patricia Merkle tree of the Ethereum storage for the Nouns contract. Consequently, each voter should perform proofs of token ownership with respect to that root.

  A challenge that arises here is the fact that the voter does not have direct access to his own secret key SK of his wallet. For instance, in Metamask the user can request signatures of any message under his secret key but cannot retrieve the bits of the secret key. This makes it difficult to compute the nullifier $H(SK, id)$, where id is an identifier for the election derived by the proposal and the current chain. To solve this issue, AZKR+AZ proposes a ZK registry that will be used not just by Nouns members but also for any other possible application. In the ZK registry, a user Alice can set a new public key PK_Z of which Alice knows the corresponding secret key SK_Z and can associate this value to her own Ethereum account address. In this way, the nullifier is set to $H(SK_Z, id, NFT_{id})$, where NFT_{id} is the id of the NFT that the voter is using to cast a vote. Then the ZK proof additionally proves that SK_Z is associated in the ZK registry to an Ethereum address that owns the token associated with id NFT_{id}. Note that with this approach the user will use her Metamask account only once to register the ZK key and then she can vote with only her secret key SK_Z.

  A problematic use of anonymous voting is a lack of guarantee of fairness: the partial tally at each time slot can be disclosed since the voting preferences are published. To address this issue, AZKR+AZ proposes to rely on the League of Entropy (LOE) drand service [7] in a way which is inspired by the recent tlock encryption system [33].

  LOE publishes a public key PK_L whose secret key SK_L is shared among the LOE members. The public key is for a bilinear map group and at each interval of 30 seconds the members perform a joint BLS signature in a way that the signature of time t can be computed if the majority of members participated in the signature process for time t. This signature has the form $H(t)^{SK_L}$. Then each voter can do the following. The generic i-th voter chooses a random value r_i and computes $A_i = g^{r_i}$ and key $K_i = e(H(t), PK_L)^{r_i}$. The key K_i can be hashed to reduce its size but for simplicity this will not be considered. Then the voter i with voting preference v_i, and NFT NFT_{id} publishes A_i, the nullifier $H(SK_Z, id, NFT_{id})$, the value $B_i = H(K_i, v_i)$ and the ZK membership proof that secret key

SK_Z is associated in the ZK registry with an Ethereum account that holds an NFT with identifier NFT_{id}, and that there exist values K_i, v_i such that $B_i = H(K_i, v_i)$ and v_i is a valid voting preference. Note that K_i is used in the circuit only to prove the preimage of v_i and that A_i is not used at all by the voter in the ZK circuit. Consequently, A_i could be stored elsewhere outside Ethereum. Each time a new ballot of voter k is published and the corresponding ZK proof is accepted, the contract hashes the newly received B_k with the partial hash of all previous B_i's for $i = 1, \ldots, k-1$, so the contract results in a value B that is a vector commitment to all values B_i's of voters who submitted valid proofs.

To tally the result after time t, anyone, including voters who participated in the election, the proposal creator, or any external party, can use the values A_i and the signature $H(t)^{SK_t}$ released by LOE to compute $e(H(t)^{SK_L}, A_i) = K_i$ and by brute force can compute the value v_i such that $H(K_i, v_i) = B_i$. This party can then send proof to the contract that B is the hash of values B_i whose preimages are v_i and that the v_i's tally to a given value. The contract will announce the result only if a tally with corresponding accepted proof of correctness is received.

Notice that with this method the SNARK circuit of the voter does not need to perform the pairing internally, therefore avoiding expensive non-SNARK-friendly operations. The SNARK circuit of the party who computes the proof of correctness of the tally instead needs to perform the pairing for ballots that are incorrect. It is possible that despite the voter's ZK proof being accepted, B_i is dishonestly computed and so the prover needs to show the dishonesty. Moreover, observe that for this proof the ZK property is not essential since the K_i's could be sent to the contract publicly.

AZKR+AZ is currently studying more efficient ways to achieve their ZK proofs and improve on the previous system. To prevent a specific Ethereum address being associated with a voting preference, the voter's ballot should be sent by a different Ethereum account. Alternatively, AZKR+AZ propose a Relay service that receives transactions, groups them together, and delays their publication on Ethereum at random time intervals.

- DeFrost proposal for the Nouns DAO [32]: DeFrost uses the threshold and homomorphic cryptography combined with DKG to guarantee privacy and verifiability in the following way. A set of Nouns members run a DKG protocol to share the secret key SK corresponding to a public key PK that is published on-chain.

  Each voter can then send the ElGamal encryption of its voting power directly from his own Ethereum account adding a proof that his ethereum address holds the amount of NFTs encrypted in the ciphertext. The smart contract can check that the Ethereum account from which the transaction originates did not vote before, without the need of using any nullifier, therefore the system satisfies privacy but not anonymity.

The encryptions are homomorphically tallied so that there is a single ciphertext CT encrypting the tally of the election. The DKG parties can then decrypt the tally in CT in a verifiable way.

- Axiom proposal to the Nouns DAO [32]: The Axiom proposal is not a fully complete voting solution for the Nouns DAO but is a technique that can be exploited by any other solution based on proofs of membership. To explain the motivation, consider the above solution by AZKR. The contract is taking a snapshot at a determined point and then each voter can provide SNARK proofs with respect to that snapshot. The issue to be solved is that the snapshot encodes the Ethereum storage, so the SNARK circuits will be very complex. Axiom proposes to take a snapshot S_1 of the current Ethereum storage and then constructs a second snapshot S_2 which represents a Poseidon Merkle tree, a Merkle tree based on the Poseidon hash function, whose leaves encode the same information as S_1. Next, a prover proves that S_2 is well-formed with respect to S_1. This requires a complex proof. However, from this point it is guaranteed that the current NFT ownership is encoded in S_2. The tree S_2 is SNARK friendly so the voter proofs can be much more efficient. The previous approach can be improved by having a tree where leaves are hashed using Poseidon and internal nodes are instead hashed using a simpler and more efficient hash function.

- Helios [34]: Helios is a notable internet e-voting system that is based on different cryptographic primitives. Previous versions of Helios offered only verifiable shuffle-based tallying whereas newer variants also offer homomorphic-based tallying. Helios has different modes of operations ranging from a single trusted authority to a set of trusted authorities. It is currently used by the International Association for Cryptology Research (IACR) to perform internal elections of their board of directors.

- Verificatum [22]: Verificatum is an e-voting system and library based on verifiable shuffles. It supports the most efficient proofs of correct shuffles but unfortunately is not optimized for web3 applications. Verificatum was used in the 2017 and 2019 Estonian electronic elections.

- OpenVote Network [35]: This system differs from all other approaches mentioned in this survey. Specifically, this protocol requires additional rounds of interaction among the voters as follows: Each voter is required to send a first message to the blockchain. When all first messages are received, the protocol continues with a second round in which all voters are again required to send a message. This is a boardroom voting protocol where all voters are required to participate. If even a single voter does not participate, the others cannot compute the tally. The advantage is that it achieves maximum voter privacy meaning that no collusion of voters or external attackers can break the privacy of any voter. Additionally, no TOR network is required for privacy, as the system is fully

private without requiring any trusted assumption or anonymous channel. Such anonymous channels were implicitly required in approaches based on membership proofs.

An issue with this approach is that the last voter can compute the tally and if he does not like the result, he can decide to not finish the protocol, making it impossible for the other voters to compute any tally. This issue can be mitigated in the web3 setting using escrow mechanisms to incentivize voters to finish the protocol.

- Vocdoni [36]: Vocdoni is a versatile and modular blockchain-based protocol which enables various voting schemes and configurations. It offers support for vote privacy through the following methods:
  - Elections utilizing Zero-Knowledge membership proofs: The Census consists of a vector commitment containing the addresses (20-byte hashed public keys) of eligible participants.
  - Elections employing blind signatures over the secp256k1 elliptic curve, facilitated by a Credential Service Provider that allows multiple identification types such as electronic ID, OAuth, and 2FA.

  To ensure votes remain secret until the end of the election, an onion-based encryption scheme is used. Votes are encrypted with keys generated by the blockchain validators.

  In order to minimize the impact of coercion and vote-buying, a SNARK-based shuffle protocol is proposed (though not yet implemented) for mixing votes within subgroups before registering them on the public ledger.

6 Acknowledgements

We acknowledge Ivan Visconti, Alex Kampa and Andrew Barnes for reviewing, and improving this document and Rebekah Mercer for previous work on which part of this paper is based on.

References

[1] Oded Goldreich. *Foundations of Cryptography, Volume 1*. Cambridge University Press, 2001.

[2] Oded Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge University Press, 2004.

[3] Arno Mittelbach and Marc Fischlin. The theory of hash functions and random oracles: An approach to modern cryptography. *The Theory of Hash Functions and Random Oracles*, 2021.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak sha-3 submission. Submission to NIST (Round 3), 2011.

[5] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, August 2021.

[6] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. Cryptology ePrint Archive, Paper 2023/323, 2023. `https://eprint.iacr.org/2023/323`.

[7] Drand Team. Distributed randomness beacon. `https://drand.love/`.

[8] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[9] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np statements in zero-knowledge and a methodology of cryptographic protocol design (extended abstract). In *Advances in Cryptology — CRYPTO' 86*, pages 171–185. Springer Berlin Heidelberg, 1987.

[10] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, 1994.

[11] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645. Springer Berlin Heidelberg, 2013.

[12] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. `https://eprint.iacr.org/2016/260`.

[13] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. `https://eprint.iacr.org/2019/953`.

[14] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. `https://eprint.iacr.org/2018/046`.

[15] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. Cryptology ePrint Archive, Paper 2020/811, 2020. `https://eprint.iacr.org/2020/811`.

[16] Amos Beimel. Secret-sharing schemes: A survey. In *Coding and Cryptology*, pages 11–46. Springer Berlin Heidelberg, 2011.

[17] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, 1985.

[18] Roger Baig. E-voting Glossary. Aragon ZK Research, internal document, 2023.

[19] Chelsea Komlo, Ian Goldberg, and Douglas Stebila. A formal treatment of distributed key generation, and new constructions. Cryptology ePrint Archive, Paper 2023/292, 2023. `https://eprint.iacr.org/2023/292`.

[20] 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.

[21] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[22] Douglas Wikstrom. Verificatum. `https://www.verificatum.org/`, 2022.

[23] Stefano Bistarelli, Bruno Lazo La Torre Montalvo, Ivan Mercanti, and Francesco Santini. An e-voting system based on tornado cash. In *Emerging Technologies for Authorization and Authentication: 5th International Workshop, ETAA 2022, Copenhagen, Denmark, September 30, 2022, Revised Selected Papers*, page 120–135, 2023.

[24] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 103–118. Springer Berlin Heidelberg, 1997.

[25] Dario Catalano and Dario Fiore. Vector commitments and their applications. Cryptology ePrint Archive, Paper 2011/495, 2011. `https://eprint.iacr.org/2011/495`.

[26] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash Protocol Specification. `https://zips.z.cash/protocol/protocol.pdf`, 2022.

[27] Rebekah Mercer. Decentralised Private Voting. `https://github.com/aragonzkresearch/blog/blob/main/pdf/private-voting.pdf`, 2023.

[28] Jonathan Bootle, Kaoutar Elkhiyaoui, Julia Hesse, and Yacov Manevich. Dualdory: Logarithmic-verifier linkable ring signatures through preprocessing. In *Computer Security – ESORICS 2022*, pages 427–446. Springer Nature Switzerland, 2022.

[29] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology – CRYPTO 2004*, pages 41–55. Springer Berlin Heidelberg, 2004.

[30] Nabiha Asghar. A Survey on Blind Digital Signatures. `https://nabihach.github.io/co685.pdf`, 2011.

[31] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ros. Cryptology ePrint Archive, Paper 2020/945, 2020. `https://eprint.iacr.org/2020/945`.

[32] Nouns DAO. Private Voting Research Sprint. `https://prop.house/nouns/private-voting-research-sprint`, 2023.

[33] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. tlock: Practical timelock encryption from threshold bls. Cryptology ePrint Archive, Paper 2023/189, 2023. `https://eprint.iacr.org/2023/189`.

[34] Ben Adida. Helios Voting. `https://vote.heliosvoting.org/`.

[35] Mohamed Seifelnasr, Hisham S. Galal, and Amr M. Youssef. Scalable open-vote network on ethereum. Cryptology ePrint Archive, Paper 2020/033, 2020. `https://eprint.iacr.org/2020/033`.

[36] Vocdoni Team. Vocdoni: The Voice of Digital Voting. `https://https://vocdoni.io//`, 2023.