SNARV: Succinct Non-Interactive Argument of Voting

Vincenzo Iovino, Alex Kampa, and Matan Prasma [⋆]

Aragon Association

Abstract. We present new verifiable voting schemes that are suitable for Token-based voting on Blockchains. Voting procedures are carried out with the purpose of deciding whether or not to transfer funds to some Blockchain address. Currently, this has been done without privacy, that is the voters make public their voting preferences. There are two main types of voting settings: the *multisig* and the *referendum* one. In the former, there is a threshold k and the transfer of funds is accepted if there are at least k votes in support of the transfer. In the latter, the transfer is accepted if there is support from a majority of voters.
Voting on-chain, e.g. on the Ethereum network, is costly and may consume a large quantity of GAS. For this reason, in OVOTE (arnacube *et al.* - 2022) it is proposed to perform the voting procedure off-chain and send to a smart contract only a SNARK proof that allows the smart contract to verify that a sufficient number of signatures have been submitted by valid voters.
We put forth a formal model of what we call a *Succinct Non-Interactive Argument of Voting* (SNARV) that encompasses both SNARK-based and non SNARK-based voting solutions. We present two non SNARK-based SNARVs instantiated over bilinear groups, that we call BatRaVot and SchnorrVot, in which the election result can be verified efficiently. All our protocols are provably secure in the random oracle model under the Computational Diffie-Hellman assumption and do not depend on any CRS or trusted setup.
In our protocols each voter needs to register his/her own public-key into the smart contract so that the voting rights might be directly linked to token ownership and there is no need for an admin to setup a census.
Keywords: e-voting, proof systems, Blockchain.

1 Introduction

E-voting on Blockchain. E-voting on Blockchains has become popular in recent years. Often, voting eligibility is determined by ownership of a token, and the election result triggers a certain action in a smart-contract.

A common use-case appears in Decentralised Autonomous Organisations (DAOs), that are members-owned communities without central leadership. Most DAOs run over the Ethereum network and are governed by simple smart contracts. A main DAO's functionality is to transfer funds to an account if a sufficient number of members support that.

There are two main types of E-voting on Blockchain: *multisig* and *referendum*. In the former, there is a threshold k and the transfer of funds is accepted if there are at least k votes in support of the transfer. In the latter, the transfer is accepted if there is support from the majority of the voters. Moreover, each voter can have a weight depending on their amount of owned tokens. For simplicity, hereafter we will suppose that each voter has the same weight and in Remark 1 we outline how to extend our model to encompass weights.

A naïve implementation of an e-voting protocol may perform all computations on-chain. However, certain Blockchains charge fees per computation unit that may result expensive. To address this problem, Arnaucube *et al.* [aEBK22] proposed solutions that minimise computation on-chain to save, for example, GAS consumption on Ethereum. To this end, the voting procedure is done off-chain (e.g., on another Blockchain) and from the signatures of the voters, a SNARK proof is computed along with the result of the election. The SNARK security guarantees that if the proof is accepted by the corresponding SNARK Verifier, the result is correct. In this way, only the result and the SNARK proof are sent to a smart contract on-chain, which then performs the verification of the result using the SNARK Verifier.

More precisely, the system works as follows. The public-keys (PKs) of the n eligible voters are arranged in a Merkle tree whose root C is called the *census*. Each voter signs their own preference using their own PK. Suppose that m_0, m_1 voters submit a NO and YES preference respectively. The SNARK proof proves knowledge of m_0 signatures for NO and m_1 signatures for YES, where each signature is verified with respect to one of the PKs in the Merkle tree.

Using general-purpose SNARKs like the one of Groth [Gro16], the proof can be verified in constant time, precisely using 3 pairings and t exponentiations where t is the number of elements of the public statement.

Our goals. We take a practical stance and attempt to minimise the verification time in the multisig case. In particular, we propose voting procedures whose tally can be verified more efficiently than the SNARK-based solution when the number of voters is moderately high. That is, from a theoretical point of view, the verification will take more than constant time but will be however faster than the one in SNARK-based solutions, and in particular OVOTE, for several practical scenarios. Our protocols are also suitable for traditional organisations in which a non-private voting must be verified on low performance devices (e.g. smartphones). In our proposed protocols, voting rights can be directly linked to token ownership and there is no administrator that needs to setup a census. Moreover, as we will see in Section 1.1.4, SNARK-based solutions that attempt to guarantee privacy and succinctness (i.e., short proofs) at the same time suffer from a problem when used for referenda. We aim at resolving this issue. To achieve our goals, we propose two different protocols in a formal model which we call *Succinct Non-interactive Argument of Voting* (SNARV), that encompasses both SNARK-based solutions and the protocols we will present.

[⋆] In this manuscript, we will sometimes refer to a full version of this work that will be made available later.

1.1 Outline of SNARVs

Our protocols are formalised in the following model of SNARV. The system is parameterized by an integer $n > 0$ representing a bound on the maximum number of voters. We suppose there is a *parameters generation* (PPT) algorithm Gen that, on input the security parameter, outputs the public parameters pp. For the security we will require the adversary to see the random coins used to produce the public parameters. This models the fact that we do not assume the public parameters to be generated in a trusted way, that is there is no secret with which the security can be broken.

We suppose that at most n voters $V_1, \ldots, V_n$ can register themselves in the system at any moment and for all $i \in [n]$ the PK Pk_i of V_i, if V_i registered in the system, is publicly known to everyone and V_i keeps the corresponding *secret-key* (SK) Sk_i in secret. The pair $(\mathsf{Pk}_i, \mathsf{Sk}_i)$ is produced by voter V_i by means of a *setup* algorithm Setup on input the public parameters pp.

Moreover, for simplicity we suppose that voters register in sequence, that is the first voter to register is V_1, then V_2, etc. Thus, at any time there exists an integer $0 \leq m \leq n$ such that the only voters registered in the system are $V_1, \ldots, V_m$. Our procedures are independent from the way users carry out the registration. Indeed, we assume the PKs to be publicly visible to everyone on the Ethereum network and the smart contract to have lookup access at them. What matters is that registering a PK in the smart contract that triggers the transfer of funds should be a trusted operation because it allows the owner of the corresponding SK to submit a vote. This is a minimal assumption: without registered identities, an attacker could perform a Sybil attack adding votes as it likes.

From the n possible PKs, an administrator may compute a *census* r by means of a *census algorithm* Census executed with input the list of all PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$. The census can be used for efficiency to replace the list of all PKs with a shorter string and the verification of the election process will be done with respect to the census and not with respect to the list of all PKs. The census has to be computed by a trusted *administrator* since the administator has the power to decide which user is eligible or not. We call a protocol *administrator-free* if the census algorithm returns an empty string, i.e. if the protocol has no administrator.

Each voting procedure (i.e. election) is identified by a public *election identifier* $\mathtt{id} \in \{0,1\}^\star$. In the *voting phase*, a generic voter $V_i, i \in [n]$ can submit to a Blockchain, possibly different from Ethereum, the *first message* $c_i \triangleq (i, v_i, \mathsf{Blt}_i)$, where Blt_i is a bit string called the *ballot* of voter i computed by means of a *voting* algorithm Vote that takes as input the election identifier $\mathtt{id}$, the voter's preference v_i and its SK Sk_i.

There is a *ballot verification* algorithm $\mathsf{BalVerify}$ that on input the PK Pk of a voter, an election identifier $\mathtt{id}$ and a voter's preference v and ballot Blt returns 1 or 0 depending on whether the ballot was genuinely computed by the voter in an election for identifier $\mathtt{id}$ or not. This procedure is used in the following way: a first message on the Blockchain has the form (i, v_i, Blt_i). Then, the procedure is invoked on input $(\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)$. The ballot verification allows to discard invalid ballots, including double votes. The votes corresponding to the valid ballots can be then publicly tallied. Let m_0, m_1 be the number of NO and YES eligible votes and let $S_0, S_1 \subseteq [n]$ be the set of indices of voters who submitted a NO or YES respectively. Thus, $m_0 = |S_0|$ and $m_1 = |S_1|$.

Now, a Prover Prov wants to convince a Verifier Ver with low resources (e.g., the smart contract that implements the transfer of funds) that the result is (m_0, m_1). The **prover** algorithm Prov, on input $S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_1}$, the election identifier $\mathtt{id}$, the census r and the claimed result (m_0, m_1), outputs a **proof** π of the fact that (m_0, m_1) is the tally of the election. There is a *verifier* algorithm that, with RAM access to all possible PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$, the election identifier $\mathtt{id}$, the census r, the claimed result (m_0, m_1) and a proof π outputs 1 or 0 to denote acceptance or rejection of the result. If the output is 1 the funds are transferred, and are not transferred otherwise.

In Section 2 we will provide formal security models for SNARVs that encompass both OVOTE, and our protocols BRV and SV. In the full version [Aut22] we show that OVOTE can be regarded as a SNARV.

1.1.1 Outline of BatRaVot

Our first protocol, *BatRaVot* which stands for **Batch Ratified Voting** and denoted BRV, is a SNARV that makes use of bilinear groups [BF01] and can be seen as a variant of the BLGS's aggregate signature scheme [BGLS03].

It can be instantiated over both symmetric and asymmetric bilinear groups. Here we present an outline of the symmetric version and defer to Section 3 for a complete description of the scheme.

We also use a hash function Hash that maps binary strings in $\{0,1\}^\star$ into $\mathbb{G}$ (the hash function takes as input the description of the group, but henceforth we will skip this detail). We now specify the algorithms

$$(\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{BalVerify}, \mathsf{Vote}, \mathsf{Prov}, \mathsf{Ver}).$$

The Gen algorithm, on input the security parameter 1^λ, outputs the public parameters pp that consist of a description of a bilinear group of order q where q is a prime of λ bits and a generator g of $\mathbb{G}$. This protocol is administrator-free, i.e., we can assume the census algorithm Census to output an empty string.

The setup algorithm Setup, on input the public parameters pp, outputs the pair $(\mathsf{Pk}, \mathsf{Sk}) \triangleq (g^s, s)$, where s is a randomly sampled in $\mathbb{Z}_q$. We assume that for all $i \in [n]$, the PK $\mathsf{Pk}_i \triangleq g^{s_i}$ of V_i is publicly known to everyone (if V_i registered in the system) and V_i keeps the corresponding *secret-key* (SK) $\mathsf{Sk}_i \triangleq s_i$ secret. Each election is identified by a public *election identifier* $\mathtt{id} \in \{0,1\}^\star$. The voting procedure Vote, on input a preference v_i and SK Sk_i of a generic voter $V_i, i \in [n]$, works as follows. The procedure sets $g_{v_i} = \mathsf{Hash}(\mathtt{id}, v_i)$ and outputs a ballot $\mathsf{Blt}_i \triangleq g_{v_i}^{\mathsf{Sk}_i} = \mathsf{Hash}(\mathtt{id}, v_i)^{s_i}$.

The verification ballot algorithm $\mathsf{BalVerify}$, on input a PK Pk of a voter, an election identifier $\mathtt{id}$, a vote v, and a ballot Blt outputs 1 if and only if $\mathbf{e}(\mathsf{Blt}, g) = \mathbf{e}(\mathsf{Pk}, g_v)$. Only a voter whose PK is Pk can compute a ballot Blt that is verified under Pk [BLS01]. Moreover, it is possible to check that a voter cast more than one valid ballot. For $v \in \{0,1\}$, let $S_v \equiv S_v(\mathtt{id})$ be the set of indices $i \in [n]$ such that voter V_i submitted a valid vote with preference v for a given election identifier $\mathtt{id}$.

The prover algorithm Prov works as follows. On input $S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_1}$, and the election identifier $\mathtt{id}$, computes the following. Prov sets

$$\gamma = \prod_{i \in S_0} \mathsf{Blt}_i \cdot \prod_{i \in S_1} \mathsf{Blt}_i$$

and $\pi \triangleq (\gamma, S_0, S_1)$ as its *proof*.

The verifier algorithm Ver, with RAM access to all possible PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$, the election identifier $\mathtt{id}$ and a proof π, computes the following. For all $v \in \{0,1\}$, Ver computes $g_v \triangleq \mathsf{Hash}(\mathtt{id}, v), H_v \triangleq \prod_{i \in S_v} \mathsf{Pk}_i$ and accepts the proof if and only if:

$$\mathbf{e}(\gamma, g) = \mathbf{e}(H_0, g_0) \cdot \mathbf{e}(H_1, g_1). \tag{1}$$

By properties of the bilinear map, Equation 1 holds if and only if γ is honestly computed and so the prover "knows" $m_0 \triangleq |S_0|, m_1 \triangleq |S_1|$ is the correct tally. We will prove that, under the Computational Diffie-Hellman (CDH) Assumption, except with negligible probability, Equation 1 holds only if the prover "knows" m_0, m_1 signed ballots.

Note also that in the multisig case, the number of pairings is reduced to 1.

1.1.2 Outline of SchnorrVot To demostrate the generality of our model we show a second SNARV protocol that is inspired by the Schnorr's proof of knowledge protocol and called SchnorrVot. SchnorrVot, denoted SV, makes use of bilinear groups and can be instantiated over both symmetric and asymmetric bilinear groups. In this section we present an outline of the symmetric version and defer to Section 3 for a complete description the scheme. As in BRV, we use a hash function Hash that takes as input a description of the bilinear group and maps binary strings in $\{0,1\}^\star$ into $\mathbb{G}$. We now specify the algorithms ($\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{BalVerify}, \mathsf{Vote}, \mathsf{Prov}, \mathsf{Ver}$).

The Gen algorithm, on input the security parameter 1^λ, outputs the public parameters pp that consist of a description of a bilinear group of order q where q is a prime of λ bits and a generator g of $\mathbb{G}$. The setup algorithm Setup, on input the public parameters pp, outputs the pair $(\mathsf{Pk}, \mathsf{Sk}) \triangleq (g^s, s)$, where s is a randomly sampled element in $\mathbb{Z}_q$. As in BRV, we assume that for all $i \in [n]$ the PK $\mathsf{Pk}_i \triangleq g^{s_i}$ of voter V_i is publicly known to everyone (if V_i registered in the system) and V_i keeps the corresponding *secret-key* (SK) $\mathsf{Sk}_i \triangleq s_i$ secret. This protocol is administrator-free, i.e., we can assume the census algorithm Census outputs an empty string.

Each election is identified by a public identifier $\mathtt{id} \in \{0,1\}^\star$. The voting procedure Vote, on input an election identifier $\mathtt{id}$, a preference v_i and the SK Sk_i of a generic voter $V_i, i \in [n]$, works as follows. The procedure sets $g_v \equiv g_v(\mathtt{id}) \triangleq \mathsf{Hash}(\mathtt{id}, v)$. The procedure samples $r_i \leftarrow \mathbb{Z}_q$ and outputs the ballot $\mathsf{Blt}_i \equiv (x_i, y_i, z_i) \triangleq (g_{r_i}, g_v^{r_i}, r_i + \mathsf{Sk}_i \mod q)$. The verification ballot algorithm $\mathsf{BalVerify}$, on input a PK Pk of a voter, an election identifier $\mathtt{id}$, a vote v, and a ballot $\mathsf{Blt} \triangleq (x, y, z)$, computes $g_v \equiv g_v(\mathtt{id}) \triangleq \mathsf{Hash}(\mathtt{id}, v)$ and outputs 1 if and only if $g^z = x \cdot \mathsf{Pk}$ and $\mathbf{e}(x, g_v) = \mathbf{e}(y, g)$. We will prove that only a voter whose PK is Pk can compute a ballot that is verified under Pk. Moreover, it is possible to check that a voter submitted more than one valid ballot.

For all $v \in \{0,1\}$, let $S_v \equiv S_v(\mathtt{id})$ be the set of indices $i \in [n]$ such that voter V_i submitted a valid vote with preference v for a given election identifier $\mathtt{id}$. The prover algorithm Prov works as follows. On input

$$S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i \triangleq (x_i, y_i, z_i))_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i \triangleq (x_i, y_i, z_i))_{i \in S_1},$$

and an election identifier $\mathtt{id}$, Prov computes, for all $v \in \{0,1\}$,

$$X_v \triangleq \prod_{i \in S_v} x_i, \quad Y_v \triangleq \prod_{i \in S_v} y_i, \quad Z_v \triangleq \sum_{i \in S_v} z_i$$

and sets $\pi \triangleq (X_v, Y_v, Z_v, S_v)_{v \in \{0,1\}}$ as its *proof*.

The verifier algorithm Ver, with RAM access to all possible PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$ and on input the election identifier $\mathtt{id}$ and a proof $\pi = (X_v, Y_v, Z_v, S_v)_{v \in \{0,1\}}$ computes the following. Ver computes, for all $v \in \{0,1\}$, $g_v \triangleq \mathsf{Hash}(\mathtt{id}, v)$ and $H_v \triangleq \prod_{i \in S_v} \mathsf{Pk}_i$ and accepts the proof if and only if for all $v \in \{0,1\}$, the following equations hold:

$$\begin{gathered} g^{Z_v} = X_v \cdot H_v \\ \mathbf{e}(X_v, g_v) = \mathbf{e}(Y_v, g) \end{gathered} \tag{2}$$

By the properties of the bilinear map $\mathbf{e}$, the equations pass if the prover "knows" m_0, m_1 signed ballots for 0 or 1 respectively. We will prove that, under the Computational Diffie-Hellman (CDH) Assumption, except with negligible probability, Equations 2 hold only if the prover "knows" m_0, m_1 signed ballots for 0 or 1 respectively.

Observe that the verification can be performed using just 4 pairings (respectively 2) plus $m_0 + m_1 + 2$ (respectively $m_1 + 1$) group operations and 2 (respectively 1) exponentiations for the referendum (respectively multisig) mode.

Boardroom voting. Observe that in a setting where all voters are required to cast a vote (e.g., boardroom voting), in both previous protocols the two checks $H_v = \prod_{i \in S_v} \mathsf{Pk}_i, \quad v \in \{0,1\}$ can be removed and replaced by the only check $H_0 \cdot H_1 = H$, where H may be set at the beginning of the election as a census $H = \prod_{i\in[n]} \mathsf{Pk}_i$. In this setting, the verification is performed more efficiently.

1.1.3 Improving the verification cost for large number of voters

In both our SNARVs BRV and SV the verifier needs to take a product of PKs. In Section 3.2.3 we show how to replace the product of $m_0 + m_1$ PKs by $m_0 + m_1$ additions modulo q adding just a constant number of exponentiations and group operations.

We call the variants of BRV and SV with this optimisation BRV^{opt} and SV^{opt} respectively.

1.1.4 Our model and security of our protocols in brief

In Section 2 we will formally define a secure SNARV. In essence, a SNARV is secure if the following soundness property holds. For an efficient attacker it is difficult to produce a proof that convinces the verifier that the claimed result consists, for instance, of m_1 votes for YES if the adversary corrupted at most c voters and observed at most l_1 votes for YES with $c + l_1 < m_1$.

In our notion of soundness, we assume the PKs to be generated honestly, that is the adversary can corrupt voters only after they registered their PKs.

We also consider a stronger security definition that we call strong soundness in which the adversary can corrupt the voters before they generate the PKs (and so can generate ill-formed PKs) in an attempt to break the security. We say that a SNARV is strongly secure if it is strongly sound.

For strong soundness, we need to modify the protocols so that the PK include a proof of knowledge (PoK) of the corresponding SK. We assume that the PoK is verified by the smart contract at time of registration of the PK and this step is done only once for each subsequent election, so in our analysis we will not count the cost of verification of the PoKs. We call the variants of our protocols satisfying strong soundness in the RO model $\mathsf{BRV}^{\star}$ and $\mathsf{SV}^{\star}$, $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$.

We will also show how to instantiate the strong secure variants of BRV and SV of our protocols in the CRS model and we denote them by $\mathsf{BRV}^{\star,crs}$, $\mathsf{SV}^{\star,crs}$. We defer to Section 4 for the security proofs of our protocols in our model. In the full version we will show that OVOTE can be regarded as a strongly secure SNARV.

On the property of administration-freeness. It may seem that the property of being administration-free is not a real advantage because we already implicitly assume that the PKs are registered in the system in a trusted way. An administration-free SNARV could be used for instance in Blockchains where the PKs of the SNARV are the same as the PKs of the Blockchain. In that situation, one could allow to each Blockchain user to vote (likely according to some weight that is function of the user's account balance) using directly the SK of the user's account. Moreover, associating each voter to a cryptographic PK allows to link the voter to a real Ethereum token and so to link the voting rights in a given DAO the real DAO's users and the process can be completely done on-chain.

The referendum problem and bindingness. Consider OVOTE in referendum mode. Similar considerations will hold for other SNARK-based solutions that attempt to preserve privacy of voters with succinct proofs. Observe that two proofs π_1 and π_2 can be computed relative to valid signatures of voters who possibly sign a YES and a NO in an election for the same identifier. We may view it as if some voters participated in different elections over separate Blockchains, possibly with "bad" voters who cast opposite preferences in different elections, and each proof was produced with respect to the transcript in a given Blockchain. Moreover, each proof can be computed with respect to a smaller number of YES or NO by deleting a certain number of YES or NO.

In an election with 49 votes for YES and 50 for NO, it is possible to remove, for example, 2 signatures for NO and produce a proof π_1 that verifies with respect to the claimed result for 49 YES and 48 NO, thus triggering a transfer of funds. One may try to solve this issue by forbidding the smart contract to transfer the funds if another proof π_2 for e.g., 49 YES and 50 NO is sent to the smart contract. However, what if in addition another proof π_3 for 50 YES and 49 NO is sent to the smart contract? Because of voters' privacy, we cannot identify "bad" voters and decide which election result should be chosen. Moreover, if in such a case we say that a new election has to be organised, the protocol is exposed to a DoS attack.

Our protocols offer to circumvent this issue in that each proof comes along with the set of voters (S_0, S_1) who submitted a preference for No and YES, so the smart contract can, e.g., remove the "bad" voters from each election and end up with multiple proofs referring to disjoint sets. It can then consider them as a single proof for the union of such sets.

To take in account this problem that arises for referenda we need to consider a variant of the notion of strong soundness that binds the tally and the proof to the voters PKs. In this variant, the verfier gets as input the set of voters and so the proof can be associated to sets of voters and not just to the tally results. Moreover, it is hard to forge an accepted proof for two different pairs of sets. We say that a SNARV is *binding* if it satisfies this variant of strong soundness. Our stongly secure protocols can be (slightly) changed so that the sets are part of the proof and the verifier gets the set as input instead of m_0, m_1 and computes $m_0 = |S_0|, m_1 = |S_1|$. With this modification, our protocols can be proven to satisfy bindingness under the same assumptions. Without loss of generality, we will interchangeably refer to the strong sound and binding variants of our protocols with the same names.

1.1.5 Extensions

Delegatability. Observe that in a given referendum carried out with a generic SNARV, Alice can delegate her voting capability to Bob by just providing him with both ballots (for YES and NO).

Adding pseudonimity to SNARVs. We can use blind signatures to add pseudonimity to any SNARV in the following way. Each voter interacts with an authority to get a blind signature of his/her own SNARV PK and then submits to the smart contract such PK along with the signature of the authority. The smart contract registers the PK if and only if the signature is accepted.

1.1.6 Comparison and related work As discussed above, the major advantages of our protocols are the possibility of linking the voting rights directly to Ethereum token ownership; moreover, our protocols do not suffer from the referendum problem outlined in Section 1.1.4.

We now analyse and compare the computational costs of our protocols.

Our protocols directly improve on SNARK-based solutions like OVOTE [aEBK22] in multisig mode when the number of voters is relatively low (in the order of thousands). In particular, in OVOTE the verification, both for the multisig and referendum modes, requires 3 pairings plus t exponentiations where t is the number of field elements in the statement.

In BRV the verification requires 3 (respectively 2) pairings for the referendum (respectively multisig) mode plus m group operations, where $m \overset{\triangle}{=} m_0 + m_1$ is the number of submitted votes.

In SV the verification can be performed using 4 (respectively 2) pairings (i.e., computations of the bilinear map) plus $m+2$ (respectively $m+1$) group operations and 2 (respectively 1) exponentiations for referendum (respectively multisig) mode. Furthermore, in the case of boardroom voting, in which all voters are required to cast a vote, verification in SV achieves constant-time verification.

In BRV^{opt} and SV^{opt}, m group operations can be replaced by 2 (respectively 1) plus 4 pairings (respectively 2) plus 2 (respectively 1) group operations in the referendum (respectively multisig) mode.

The prover in our protocols performs just $O(m)$ group operations (plus a constant number of exponentiations in SV) in both modes whereas SNARK-based solution are orders of magnitude slower. Precisely, OVOTE performs $\Omega(n \cdot \log n)$ exponentiations.

We wish to compare both our solutions to the trivial solution in which the voters send off-chain a digital signature of their vote (and the election identifier) and then a prover sends all valid signatures to the smart contract that in turn verifies each signature individually to tally the result. For this comparison, we suppose the voters in the trivial solution use a Schnorr's signature. We also compare the costs for the voters among our solutions, OVOTE and the trivial solution, supposing that both in OVOTE and in the trivial solution the ballots correspond to Schnorr's signatures. We summarise the comparison of the costs in Table 1, where we also provide the costs for the variants of our protocols and OVOTE in the asymmetric bilinear group setting. Variants of our protocols over asymmetric bilinear groups will appear in a full version of this paper.

Our protocols do not use any common reference string (i.e., trusted parameters) and hence they do not require a trusted ceremony as in OVOTE. Moreover, our protocols do not require an administrator, i.e., are administrator-free as specified in Section 1.1. Both BRV and SV and their variants are proven secure in our model of secure and strongly secure SNARVs of Section 2 under the CDH assumption over bilinear groups (cf. Theorems 5 and 8). On the other hand, any SNARK-based solution, and OVOTE in particular, is based on Groth's SNARK that is only proven secure in the generic group model and, by known limitations [GW11], cannot be proven secure under falsifiable assumptions.

Additionally, BRV SV and their variants, slightly modified so that to not include the voters' sets as part of the proof, satisfy bindingness under the CDH assumption. For the strong security and bindingness of BRV^{opt} and SV^{opt} we additionally need a knowledge of exponent assumption, in particular the KEA1 of [BP04].

We summarise the comparison of properties in Table 1 for OVOTE and some of our protocols.

The notion of SNARV borrows obvious similarities from the setting of aggregate signatures and multisignatures [BLS01,BDN18].

Our first protocol, BRV, implicitly uses BLS' aggregate signatures and multisignatures [BLS01,BGLS03,BDN18] in a way that is optimised for our voting procedures. The protocol SV can be seen as a variant of batch Schnorr' signatures [PS96] and its aggregated version to compress many signatures in a short cryptographic object, that can be verified fast.

The need of adding PoKs to the PKs in both our protocols is to avoid attacks similar to rogue key attacks in the context of multisignatures [MOR01] of which we also share similarities in the setting.

Solution	No CRS	Administration-free	Off-chain	Assumption	Is safe in referendum mode
OVOTE [aEBK22]	×	×	✓	Generic group model	×
BRV, $\mathsf{BRV}^{\star}$, SV, $\mathsf{SV}^{\star}$(this work)	✓	✓	✓	CDH	✓
$\mathsf{BRV}^{\star,opt}$, $\mathsf{SV}^{\star,opt}$ (this work)	✓	✓	✓	CDH, KEA1	✓

Table 1. Comparison of the properties of our protocols with OVOTE.

Organisation. In Section 2 we will present our building blocks and security models for SNARVs and in Section 3 we will provide constructions for BRV and SV two strongly secure SNARVs along with their variants (that will appear in a future full version).

Solution	Verification cost (VC) for referendum	VC for multisig	Prover cost (both modes)	Voter cost (both modes)
OVOTE [aEBK22] (symmetric)	$3{\cdot}\mathbb{P}+d{\cdot}\mathbb{E}$	$3{\cdot}\mathbb{P}+d{\cdot}\mathbb{E}$	$\Omega(n{\cdot}\log n{\cdot}\mathbb{E})$	$1{\cdot}\mathbb{E}$
OVOTE [aEBK22] (asymmetric)	$3{\cdot}\mathbb{P}+d{\cdot}\mathbb{E}_1$	$3{\cdot}\mathbb{P}+d{\cdot}\mathbb{E}_1$	$\Omega(n{\cdot}\log n{\cdot}(\mathbb{E}_1+\mathbb{E}_2))$	$1{\cdot}\mathbb{E}_1$
BRV (symmetric)	$3{\cdot}\mathbb{P}+m{\cdot}\mathbb{G}$	$2{\cdot}\mathbb{P}+m{\cdot}\mathbb{G}$	$m{\cdot}\mathbb{G}$	$1{\cdot}\mathbb{E}$
BRV (asymmetric)	$3{\cdot}\mathbb{P}+m{\cdot}\mathbb{G}_1$	$2{\cdot}\mathbb{P}+m{\cdot}\mathbb{G}_1$	$m{\cdot}\mathbb{G}_2$	$1{\cdot}\mathbb{E}_2$
SV (symmetric)	$4{\cdot}\mathbb{P}+2{\cdot}\mathbb{E}+(m+2){\cdot}\mathbb{G}$	$2{\cdot}\mathbb{P}+1{\cdot}\mathbb{E}+(m+1){\cdot}\mathbb{G}$	$O(m){\cdot}\mathbb{G}+O(1){\cdot}\mathbb{E}$	$2{\cdot}\mathbb{E}$
SV (asymmetric)	$4{\cdot}\mathbb{P}+2{\cdot}\mathbb{E}_1+(m+2){\cdot}\mathbb{G}_1$	$2{\cdot}\mathbb{P}+1{\cdot}\mathbb{E}_1+(m+1){\cdot}\mathbb{G}_1$	$O(m){\cdot}(\mathbb{G}_1+\mathbb{G}_2)+O(1){\cdot}\mathbb{E}_1$	$1{\cdot}(\mathbb{E}_1+\mathbb{E}_2)$
BRV^{opt} (symmetric)	$7{\cdot}\mathbb{P}+2{\cdot}\mathbb{E}+2{\cdot}\mathbb{G}+m{\cdot}\mathbb{Z}$	$4{\cdot}\mathbb{P}+1{\cdot}\mathbb{E}+1{\cdot}\mathbb{G}+m{\cdot}\mathbb{Z}$	$m{\cdot}\mathbb{G}$	$1{\cdot}\mathbb{E}$
SV^{opt} (symmetric)	$8{\cdot}\mathbb{P}+4{\cdot}\mathbb{E}+4{\cdot}\mathbb{G}+m{\cdot}\mathbb{Z}$	$6{\cdot}\mathbb{P}+2{\cdot}\mathbb{E}+2{\cdot}\mathbb{G}+m{\cdot}\mathbb{Z}$	$O(m){\cdot}\mathbb{G}+O(1){\cdot}\mathbb{E}$	$2{\cdot}\mathbb{E}$
Trivial solution	$2m{\cdot}\mathbb{E}+m{\cdot}\mathbb{G}$	$2m{\cdot}\mathbb{E}+m{\cdot}\mathbb{G}$	0	$1{\cdot}\mathbb{E}$

Table 2. Comparison of computational costs of our protocols with known solutions. In the table, d stands for the number of field elements of which a statement in OVOTE is composed of. The verification costs in referendum mode for OVOTE are highlighted in red to remark that there is no clear way to securely use OVOTE, as any SNARK-based solution, in the referendum mode (see Section 1.1.4). This depends on the implementation. $\mathbb{G}, \mathbb{E}, \mathbb{P}, \mathbb{Z}$ stand respectively for the number of group operations, exponentiations, pairings and modular additions, and for the asymmetric case the subscript 1 (respectively 2) refers to operations in $\mathbb{G}_1$ (respectively $\mathbb{G}_1$). The parameter n represents the number of voters registered in the system and m represents the number of voters who submit a valid vote. In the trivial solution the voters produce Schnorr's signatures of their votes, the prover just sends all valid signatures to the smart contract that verifies each signature individually in the obvious way. We also suppose the voters in OVOTE to use Schnorr's signatures. The cryptographic operations for the trivial solution refers to a generic and possibly non-pairing-friendly secure group of comparable security. For strong soundness, the voter also needs to add a Schnorr's proof of knowledge of the SK (not included in the analysis) just once at setup time. For simplicity we discuss the costs of just the symmetric variants of BRV^{opt} and SV^{opt}.

2 Definitions

Throughout this paper, PPT stands for probabilistic polynomial-time algorithm and nuPPT for non-uniform PPT algorithm. We assume PPT algorithms to use only λ bits of randomness on inputs of length λ; a cryptographic pseudo-random generator can be used to expand the randomness from λ to any polynomial number of bits in λ. If $\mathcal{A}$ is a probabilistic algorithm and x, r are two strings of equal length we denote by $\mathcal{A}(x; r)$ the execution of $\mathcal{A}$ on input x and random coins r. We denote by $y \leftarrow \mathcal{A}(x)$ the random process of choosing a random string $r \leftarrow \{0,1\}^{|x|}$ and computing $y = \mathcal{A}(x; r)$; in this case we say that y is **sampled** from $\mathcal{A}(x)$. For simplicity, we will sometimes make use of *stateful algorithms*. A stateful algorithm is one that during the execution in an experiment keeps a state. For instance if $y = \mathcal{A}(x; r)$ and then $\mathcal{A}$ is executed again on possibly different inputs, $\mathcal{A}$ remembers its previous state, that is the previous input x and the previous random string r. This is equivalent to assume that in each execution $\mathcal{A}$ also takes as input the previous *state* (that is empty in the first execution) and also outputs the update state.

2.1 Building blocks

2.1.1 Bilinear groups

Definition 1. [BF01,CHKM10] A **symmetric bilinear group** consists of a pair of groups, $\mathbb{G}, \mathbb{G}_T$, called the **base** and **target** group respectively, both of some prime order q endowed with a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ satisfying the following properties.

1. Bilinearity: for all $a, b \in \mathbb{Z}_q, g \in \mathbb{G} : e(g^a, g^b) = e(g, g)^{ab}$.
2. Non-degeneracy: $\exists\ g \in \mathbb{G} : e(g, g) \neq 1_{\mathbb{G}_T}$.

A symmetric bilinear group is also called a **Type I** bilinear group in [CHKM10].

Definition 2. A PPT algorithm BiGen is called a **generator of a family of (symmetric) bilinear groups** if the following properties hold.

- Efficient generation of the bilinear group description. BiGen, on input the security parameter λ, outputs a description of the base group $\mathbb{G}_\lambda$ and of the target group $\mathbb{G}_{T,\lambda}$ along with a prime number q_λ of λ bits and a generator g_λ of $\mathbb{G}_\lambda$. Moreover, the algorithm outputs the description of a map $\mathbf{e}_\lambda : \mathbb{G}_\lambda \times \mathbb{G}_\lambda \longrightarrow \mathbb{G}_{T,\lambda}$ satisfying bilinearity and non-degeneracy.
- Efficient operations. Group operations for groups output by BiGen are efficient relative to the security parameter λ.
- Efficient decidability. Membership in any group $\mathbb{G}, \mathbb{G}_T$ output by BiGen can be tested efficiently.
- Efficient samplability. Elements can be sampled efficiently from any group output by BiGen.

Assumption 1. [Computational Diffie-Hellman Assumption over a symmetric bilinear group generator BiGen]
We assume the reader to be familiar with the Computational Diffie-Hellman Assumption formulated over a symmetric bilinear group generator and we defer to the future full version for formal definitions of the assumption and related CDHExp experiment that can be also found in [KL07].

2.1.2 Digital Signatures Digital signatures are the digital version of handwritten signatures. A valid digital signature gives a recipient guarantee that the message was created by a known sender, and that the message was not altered in transit. The sender is identified by a verification key (also called public key) and the sender is able to sign a message using his own secret key. We have the following formalisation.

Definition 3. A **digital signature** (DS) scheme $\mathsf{DS} \triangleq (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verfy})$ is a tuple of 3 PPT algorithms satisfying the standard correctness and unforgeability requirements. In the full version we will present details and the definition of the standard unforgeability experiment $\mathsf{UnforgExp}$ for DS that can be also found in [KL07].

2.2 SNARV

Definition 4. A **succinct non-interactive argument of voting** (SNARV) for parameter n (representing maximum number of voters in the system) is a tuple of PPT algorithms $\mathsf{SNARV} \triangleq (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ with the following syntax.

- $\mathsf{Gen}(1^\lambda)$: the **parameters generation** algorithm that, on input the security parameter, outputs the public parameters pp.
- $\mathsf{Setup}(\mathsf{pp})$: the **setup** algorithm is a run by a voter who, on input the public parameters pp, outputs the pair of public- and secret-keys $(\mathsf{Pk}, \mathsf{Sk})$. All voters *can register* their PK in the system at any moment and we assume that each of the n possible PKs can be identified by an index $i \in [n]$.
- $\mathsf{Census}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$: the **census** algorithm, on input the list of n possible PKs, outputs the *census* r.
- $\mathsf{Vote}(\mathtt{id}, \mathsf{v}, \mathsf{Sk})$: the **voting** algorithm, on input the **election identifier** $\mathtt{id}$, the **voter's preference** v and the voter's SK Sk, outputs the **ballot** Blt. The triple (i, v, Blt), where i is such that $\mathsf{Sk} = \mathsf{Sk}_i$ is the SK corresponding to the PK Pk_i, is called the **first message** of the i-th voter.
- $\mathsf{BalVerify}(\mathsf{Pk}, \mathtt{id}, v, \mathsf{Blt})$: the **ballot verification** algorithm, on input the PK Pk of a voter, the voter's preference v and the voter's ballot Blt, outputs 0 or 1 meaning rejection or acceptance of the fact that Blt was generated by the voter identified by PK Pk with vote v in election $\mathtt{id}$).
- $\mathsf{Prov}(S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_1}, \mathtt{id}, r, m_0, m_1)$: the **prover** algorithm, on input two disjoint sets $S_0, S_1 \subseteq [n]$ of voters' indexes who casted a NO or YES preference resp., 2 list of triples $(\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_1}$ such that $\mathsf{Pk}_i, v_i, \mathsf{Blt}_i$ are the PK, the preference and the ballot of the i-th voter respectively, if $i \in S_v$ then $v_i = v$, an election identifier $\mathtt{id}$, a census r and a claimed result (m_0, m_1) such that $|S_0| = m_0$ and $|S_1| = m_1$, outputs a **proof** π of the fact that (m_0, m_1) is the correct **tally**.
- $\mathsf{Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, m_0, m_1, \pi)$: the **verifier** algorithm, with RAM access to the list of all n possible PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$, an election identifier $\mathtt{id}$, a census r relative to the same list of PKs, a claimed result (m_0, m_1) consisting of two integers and a proof π, outputs 0 or 1 meaning acceptance or rejection of the claimed result.

A SNARV for parameter n is **secure** if it satisfies the following properties.

- Completeness. Suppose $\lambda > 0, \mathtt{id} \in \{0,1\}^\star$ and $\vec{x} \in \{0, 1, \perp\}^n$. Let $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$ and for $i \in [n]$, let
$$(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{Setup}(\mathsf{pp}), \quad r \leftarrow \mathsf{Census}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n).$$
For all $i \in [n]$ such that $x_i \neq \perp$ let $\mathsf{Blt}_i \leftarrow \mathsf{Vote}(\mathtt{id}, \mathsf{v_i}, \mathsf{Sk_i})$ and for all $v \in \{0,1\}$, let $S_v = \{i | x_i = v\}$ and $m_v = |S_v|$. Then, it holds that for all $i \in [n]$ such that $x_i \neq \perp$: $\mathsf{BalVerify}(\mathsf{Pk}_i, \mathtt{id}, v_i, \mathsf{Blt}_i) = 1$ and if $\pi \leftarrow \mathsf{Prov}(S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i \in S_1}, \mathtt{id}, r, m_0, m_1)$ then $\mathsf{Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, m_0, m_1, \pi) = 1$.
- Unforgeability. In the rest of this work, for any SNARV $(\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ we let Setup' be the algorithm that, on input 1^λ, computes $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$ and outputs $(\mathsf{Pk}, \mathsf{Sk})$. SNARV is **unforgeable** if the tuple $(\mathsf{SNARV.Setup}', \mathsf{SNARV.Vote}, \mathsf{SNARV.BalVerify})$, when seen as a DS scheme, satisfies the unforgeability property of a DS scheme (note that the pair $(\mathtt{id}, v)$ in a SNARV is interpreted as the message M to be signed in the definition of a DS scheme).
- Soundness. We first present the property in the framework of cryptographic games and then discuss the intuition behind it. For any integer $n > 0$, any list of n elements $(\mathsf{Sk}_1, \ldots, \mathsf{Sk}_n)$ and set $S \subseteq [n]$, let $\mathsf{VoteOracle}_{(\mathsf{Sk}_1, \ldots, \mathsf{Sk}_n), S}(\cdot, \cdot, \cdot)$ be the algorithm that, on input $(\mathtt{id}, v, i)$, outputs $\mathsf{Vote}(\mathtt{id}, \mathsf{v}, \mathsf{Sk_i})$ if $\mathtt{id} \in \{0,1\}^\star, v \in \{0,1\}, i \in S$ and $\perp$ otherwise. Consider the following experiment $\mathsf{SndExp}^{\mathcal{A}, \mathsf{SNARV}, \lambda, \mathsf{n}}$ parameterised by an integer $n > 0$, a security parameter λ, a SNARV for parameter n and a stateful adversary $\mathcal{A}$.

$\mathsf{SndExp}^{\mathcal{A}, \mathsf{SNARV}, \lambda, \mathsf{n}}$:

- **Parameters Generation Phase.** $s \leftarrow \{0,1\}^\lambda, \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda; s)$.
- **Setup Phase.** $\forall i \in [n]$ $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$ and $r \leftarrow \mathsf{Census}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$.
- **Corruption Phase.** $C \leftarrow \mathcal{A}(s, \mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$. If $C \not\subseteq [n]$ the experiment outputs 0 ($\mathcal{A}$ loses).
- **Voting Phase.** $(\mathtt{id}, r, m_0, m_1, \pi) \leftarrow \mathcal{A}^{\mathsf{VoteOracle}_{(SK_1, \ldots, \mathsf{Sk}_n), [n]-C}(\cdot, \cdot, \cdot)}(\{\mathsf{Sk}_i\}_{i \in C})$.
- **Winning Condition.** For $v \in \{0,1\}$, let $S_v^q \subseteq [n]$ be the set of indexes such that $\mathcal{A}$ asked the query $(\mathtt{id}, v, i)$ to $\mathsf{VoteOracle}_{(\mathsf{Sk}_1, \ldots, \mathsf{Sk}_n), [n]-C}$.
The output of the experiment is 1 (i.e., $\mathcal{A}$ wins) if and only if the following condition holds:
$\mathsf{Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, m_0, m_1, \pi) = 1 \ \wedge \ m_0 + m_1 > |S_0^q \cup S_1^q \cup C|$.

A SNARV satisfies **soundness** if for all nuPPT stateful algorithms $\mathcal{A} \triangleq \{\mathcal{A}_\lambda\}_\lambda$, the probability that $\mathsf{SndExp}^{\mathcal{A}_\lambda,\mathsf{SNARV},\lambda,\mathsf{n}}$ outputs 1 is a negligible function of λ.
The intuition behind the definition is the following. The adversary can corrupt a subset of users C meaning it gets their SKs and thus can impersonate them and produce ballots for any preference relative to any identifier via Vote. The adversary can also use the $\mathsf{VoteOracle}$ to get ballots of any voter not in C, any preference and any identifier. If the adversary asks to the oracle at least one query of the form $(\mathtt{id}^\star, v^\star, i), v \in \{0,1\}, i \in [n]$, then we say that the adversary simulated an election for identifier $\mathtt{id}^\star$.
After having simulated a certain number of elections, the adversary outputs $\mathtt{id}$, a claimed result (m_0, m_1) and a proof π. For $v \in \{0,1\}$, S_v^q corresponds to the set of voters for which the adversary queried $\mathsf{VoteOracle}$ with $(\mathtt{id}, v, i)$ and hence saw a ballot for preference v, identifier $\mathtt{id}$ and SK Sk_i. m_0, m_1 represent the claimed number of NO and YES preferences, submitted by distinct voters relative to identifier $\mathtt{id}$. We assume $C \cap (S_0^q \cup S_1^q) = \emptyset$ since $\mathsf{VoteOracle}$ does not output valid ballots for corrupted users.
The adversary wins if the proof can convince the verifier that for either $v = 0$ or $v = 1$, m_v is strictly greater than the number of ballots relative to $(\mathtt{id}, v)$ that the adversary saw, i.e if $m_v > |S_v^q \cup C|$. The addition of $|C|$ models the fact that the adversary can use the corrupted users to produce ballots for preference v and identifier $\mathtt{id}$ to its like.
Observe that, by the completeness property, the adversary can indeed always compute an accepted proof for claimed result $(|S_0^q \cup C|, |S_1^q \cup C|)$. So, a successful attack is only one in which m_v is strictly greater than $|S_v^q \cup C|$, for either $v = 0$ or $v = 1$.

A SNARV may have the following additional properties.

– Administration-freeness. A SNARV $\mathsf{SNARV} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ is **administrator-free** if Census outputs the empty string ϵ on any input.
– Strong soundness. In soundness, we only consider adversaries that can corrupt the voters after the voters honestly generate the PKs. In strong soundness, we allow the adversary to corrupt the voters' PKs before they are published, i.e., the adversary can possibly generate ill-formed PKs in an attempt to get an advantage for breaking the security. For this reason, we also consider a stronger soundness guarantee that withstands adversaries that can corrupt voters before they generate their PKs.
A strongly sound SNARV SNARV includes an additional **public-key verification** algorithm $\mathsf{PkVerify}$ that takes as input public parameters pp, and a PK Pk and outputs 0 or 1 for rejection or acceptance respectively. For completeness we require in addition that for any public parameters $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$ and a PK $\mathsf{Pk} \leftarrow \mathsf{Setup}(\mathsf{pp})$, $\mathsf{PkVerify}(\mathsf{pp}, \mathsf{Pk}) = 1$.
Consider the following experiment $\mathsf{SSndExp}^{\mathcal{A},\mathsf{SNARV},\lambda,\mathsf{n}}$ parameterised by an integer $n > 0$, a security parameter λ, a SNARV for parameter n and a stateful adversary $\mathcal{A}$.

$\mathsf{SSndExp}^{\mathcal{A},\mathsf{SNARV},\lambda,\mathsf{n}}$:

- **Parameters Generation Phase.** $s \leftarrow \{0,1\}^\lambda, \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda; s)$.
- **Corruption Phase.** $C \leftarrow \mathcal{A}(s)$. If $C \nsubseteq [n]$ the experiment outputs 0 ($\mathcal{A}$ loses).
- **Setup Phase.** $\forall i \in [n] - C$ $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$. $\forall i \in C$ set $\mathsf{Sk}_i \triangleq \bot$.
- **Voting Phase.** $(\mathtt{id}, m_0, m_1, \pi, \{\mathsf{Pk}_i\}_{i \in C}) \leftarrow \mathcal{A}^{\mathsf{VoteOracle}_{\{\mathsf{Sk}_1,\ldots,\mathsf{Sk}_n\},[n]-C}(\cdot,\cdot,\cdot)}(\{\mathsf{Pk}_i\}_{i\in[n]-C})$.
$r \leftarrow \mathsf{Census}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$.
- **Winning Condition.** For any $v \in \{0,1\}$, let $S_v^q \subseteq [n]$ be such $i \in S_v^q$ if and only if $\mathcal{A}$ asked the query $(\mathtt{id}, v, i)$ to $\mathsf{VoteOracle}_{(\mathsf{Sk}_1,\ldots,\mathsf{Sk}_n),[n]-C}$.
The output of the experiment is 1 (i.e., $\mathcal{A}$ wins) if and only if the following condition holds:
$$\begin{aligned}\mathsf{SNARV.Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, m_0, m_1, \pi) = 1 \;\wedge\; m_0 + m_1 > |S_0^q \cup S_1^q \cup C| \\ \wedge\; \forall i \in [n]\ \mathsf{PkVerify}(\mathsf{pp}, \mathsf{Pk}_i) = 1 \;\wedge\; \forall i \neq j \in [n],\ \mathsf{Pk}_i \neq \mathsf{Pk}_j.\end{aligned}$$

A SNARV SNARV satisfies strong soundness if for all nuPPT stateful algorithms $\mathcal{A} \triangleq \{\mathcal{A}_\lambda\}_\lambda$, the probability that $\mathsf{SSndExp}^{\mathcal{A}_\lambda,\mathsf{SNARV},\lambda,\mathsf{n}}$ outputs 1 is a negligible function of λ.
A SNARV SNARV is **strongly secure** if it is secure and strongly sound.
– Bindingness. Consider an experiment $\mathsf{BindExp}^{\mathcal{A}_\lambda,\mathsf{SNARV},\lambda,\mathsf{n}}$ that is identical to the one of $\mathsf{SSndExp}^{\mathcal{A}_\lambda,\mathsf{SNARV},\lambda,\mathsf{n}}$ except that the adversary outputs two sets S_0, S_1 of voters instead of the values m_0, m_1 and in the winning conditions $m_0 = |S_0|, m_1 = |S_1|$ and the verifier gets as input S_0, S_1 instead of m_0, m_1.
A SNARV SNARV satisfies **binding strong soundness** if for all nuPPT stateful algorithms $\mathcal{A} \triangleq \{\mathcal{A}_\lambda\}_\lambda$, the probability that $\mathsf{BindExp}^{\mathcal{A}_\lambda,\mathsf{SNARV},\lambda,\mathsf{n}}$ outputs 1 is a negligible function of λ. We say that a SNARV SNARV satisfies **bindingness** if no nuPPT adversary can output with non-negligible probability a sets of PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$, an identifier $\mathtt{id}$, a proof $\pi \in \{0,1\}^\star$ and two different pairs (S_0, S_1) and (S'_0, S'_1) of disjoint subsets of $[n]$ (i.e., S_0 is disjoint from S_1 and S'_0 is disjoint from S'_1) such that the following condition holds:
$$\begin{aligned}\mathsf{SNARV.Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, S_0, S_1, \pi) = 1 = \mathsf{SNARV.Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, r, S'_0, S'_1, \pi) \;\wedge \\ \forall i \in [n]\ \mathsf{PkVerify}(\mathsf{pp}, \mathsf{Pk}_i) = 1 \;\wedge\; \forall i \neq j \in [n],\ \mathsf{Pk}_i \neq \mathsf{Pk}_j.\end{aligned}$$

We say that a SNARV SNARV is **binding secure** (or, with a slight abuse, just binding) if it satisfies completeness, unforgeability, binding strong soundness and bindingness.

– Soundness, strong soundness and bindingness in the CRS model. Moreover, we consider a weaker notion of secure, strongly secure, and binding secure for SNARV. A SNARV is secure (respectively strongly secure and binding) in the Common Reference String (CRS) model if the soundness (respectively strong soundness and bindingness) property is changed as follows. In the soundness (respectively strong soundness and binding) experiment, the adversary does not get as input the randomness used to compute the public parameters.

Remark 1. [On weighted voting] Our model can be extended to deal with weighted voting in the following way. A SNARV is also parameterised by a weight function W that takes as input an election identifier $\mathtt{id}$, an index $i \in [n]$ and outputs an integer $w_i = W(\mathtt{id}, i)$ that reflects the weight of voter i at time of the election $\mathtt{id}$, e.g, the token balance of voter i at that time. Then, the winning conditions in the soundness, strong soundness, and bindingness experiments are changed in the obvious way: for all $v \in \{0,1\}$, the quantity $|S_v|$ is replaced by $\sum_{i\in S_v} W(\mathtt{id}, i)$. Note that weighted voting in the multisig case is conceptually closer to the idea of petition: a petition passes if there is a sufficient number of weighted votes in favour of it.

It is trivial to see that a SNARV that satisfies strong soundness also satisfies soundness.

Lemma 1. Let $n > 0$ be an integer and let $\mathsf{SNARV} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ be a SNARV. If SNARV satisfies strong soundness then SNARV also satisfies soundness.

A SNARV that satisfies soundness also satisfies unforgeability. The proof for the following lemma will be provided in the full version.

Lemma 2. Let $n > 0$ be an integer and let $\mathsf{SNARV} \triangleq (\mathsf{SNARV.Gen}, \mathsf{SNARV.Setup}, \mathsf{SNARV.Census}, \mathsf{SNARV.Vote}, \mathsf{SNARV.BalVerify},$ be a SNARV. If SNARV satisfies soundness then it also satisfies unforgeability.

3 Our SNARVs

In this section we present our SNARVs BatRaVot (BRV) and SchnorrVot (SV) over symmetric bilinear groups. The asymmetric analogues can made with obvious changes and are thus deferred to a future full version.

3.1 BatRaVot and its variants

Let BiGen be a symmetric bilinear group generator and Hash be an hash function that maps binary strings in $\{0,1\}^\star$ into $\mathbb{G}$. The hash function Hash actually takes as input the description of the group, but we will omit this detail to ease notation. As costumary, Hash will be modelled as a random oracle in the security reduction.

Let $n > 0$ be an integer. We construct an administrator-free SNARV for parameter n

$$\mathsf{BRV} \triangleq (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$$

as follows.

Construction 3. [Our SNARV BRV over symmetric bilinear groups]

– $\mathsf{Gen}(1^\lambda)$: the parameters generation algorithm that, on input the security parameter, computes $(\mathbb{G}, \mathbb{G}_T, q, g) \leftarrow \mathsf{BiGen}(1^\lambda)$, sets $\mathsf{pp} \triangleq (\mathbb{G}, \mathbb{G}_T, q, g)$ and outputs the public parameters pp.
– $\mathsf{Setup}(\mathsf{pp})$: the setup algorithm is a run by a voter who, on input the public parameters $\mathsf{pp} \triangleq (\mathbb{G}, \mathbb{G}_T, q, g)$, samples $s \leftarrow \mathbb{Z}_q$ and outputs the pair of public- and secret-keys ($\mathsf{Pk} \triangleq g^s, \mathsf{Sk} \triangleq s$).
– $\mathsf{Census}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$: the census algorithm, on input a list of n PKs, outputs an empty string (the SNARV is administrator-free).
– $\mathsf{Vote}(\mathtt{id}, \mathsf{v}, \mathsf{Sk})$: the voting algorithm, on input the election identifier $\mathtt{id}$, the voter's preference v and the voter's SK $\mathsf{Sk} \triangleq s$, computes the ballot $\mathsf{Blt} \triangleq \mathsf{Hash}(\mathtt{id}, v)^s$.
– $\mathsf{BalVerify}(\mathsf{Pk}, (\mathtt{id}, v), \mathsf{Blt})$: the ballot verification algorithm, on input a PK Pk of a voter, the voter's preference v and the voter's ballot Blt, computes $g_v = \mathsf{Hash}(\mathtt{id}, v)$, and outputs 1 if and only if$\mathbf{e}(\mathsf{Blt}, g) = \mathbf{e}(\mathsf{Pk}, g_v)$.
– $\mathsf{Prov}(S_0, S_1, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i\in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i\in S_1}, \mathtt{id})$: the prover algorithm, on input the sets $S_0, S_1 \subseteq [n]$ of voters who submitted a NO or YES preference respectively such that $S_0 \cap S_1 = \emptyset$, two list of triples $(\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i\in S_0}, (\mathsf{Pk}_i, v_i, \mathsf{Blt}_i)_{i\in S_1}$ such that for all $v \in \{0,1\}, i \in S_v$, Pk_i is the PK of the i-th voter, $v_i = v$ is the preference of the i-th voter, Blt_i is the ballot of the i-th voter and the election identifier $\mathtt{id}$, computes the following.
  Prov sets $\gamma = \prod_{i\in S_0} \mathsf{Blt}_i \cdot \prod_{i\in S_1} \mathsf{Blt}_i$ and outputs $\pi \triangleq (\gamma, S_0, S_1)$ as its proof of the fact that $(m_0 \triangleq |S_0|, m_1 \triangleq |S_1|)$ is the correct *tally*.
– $\mathsf{Ver}(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n, \mathtt{id}, \pi)$: the verifier algorithm, with RAM access to the the list of all PKs $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$, the election identifier $\mathtt{id}$ and a proof $\pi \triangleq (\gamma, S_0, S_1)$, computes the following. Check that pp is a valid bilinear group instance, all PKs in the list $(\mathsf{Pk}_1, \ldots, \mathsf{Pk}_n)$ and γ are valid group elements in $\mathbb{G}$. If the check fails, output 0 otherwise continue as follows. For all $v \in \{0,1\}$, Ver computes $g_v \triangleq \mathsf{Hash}(\mathtt{id}, v), H_v \triangleq \prod_{i\in S_v} \mathsf{Pk}_i$. Ver outputs 1 if and only if:

$$\mathbf{e}(\gamma, g) = \mathbf{e}(H_0, g_0) \cdot \mathbf{e}(H_1, g_1).$$

3.1.1 $\mathsf{BRV}^\star$ and $\mathsf{BRV}^{\star,crs}$

To achieve strong soundness (cf. Definition 4), we need to modify BRV in the following way. We consider two versions, one in the random oracle model that we call $\mathsf{BRV}^\star$ and one in the CRS model that we call $\mathsf{BRV}^{\star,crs}$.

$$\mathsf{BRV}^\star = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$$

is identical to BRV except that in the following.

- $\mathsf{BRV}^\star.\mathsf{Setup}$. The setup algorithm is a run by a voter who, on input the public parameters $\mathsf{pp} \triangleq (\mathbb{G}, \mathbb{G}_T, q, g)$, samples $s \leftarrow \mathbb{Z}_q$ and outputs the pair of public- and secret-keys $(\mathsf{Pk} \triangleq (\mathsf{Pk}^{(1)} \triangleq g^s, \mathsf{Pk}^{(2)} \triangleq \pi), \mathsf{Sk} \triangleq s)$, where π is a Schnorr's ZK proof of knowledge of the discrete log of $\mathsf{Pk}^{(1)}$ in basis g.
- The $\mathsf{BRV}^\star.\mathsf{PkVerify}$ algorithm. $\mathsf{BRV}^\star$ also comes with a public-key verification algorithm $\mathsf{BRV}^\star.\mathsf{PkVerify}$ algorithm that on input the public parameters pp and a PK $\mathsf{Pk} = (\mathsf{Pk}^{(1)}, \pi)$ outputs a bit 1 if and only if the Schnorr's verifier accepts the proof π with respect to the statement $\mathsf{Pk}^{(1)}$.

The SNARV

$$\mathsf{BRV}^{\star,crs} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{PkVerify})$$

is identical to BRV except in the following. We assume the existence of a simulation-sound extractable NIZK $\mathsf{NIZK} \triangleq (K, P, V)$ (cf. [GO14]) for the NP relation R of all triples (G, A, a) such that G is the description of a symmetric bilinear group $(\mathbb{G}, \mathbb{G}_T, q, g)$, $A \in \mathbb{G}, a \in \mathbb{Z}_q$ and $A = g^a$.

- Modification to $\mathsf{BRV}^{\star,crs}.\mathsf{Gen}$. $\mathsf{BRV}^{\star,crs}.\mathsf{Gen}(1^\lambda)$, on input the security parameter, computes $(\mathbb{G}, \mathbb{G}_T, q, g_1, g_2) \leftarrow \mathsf{BiGen}(1^\lambda)$, a CRS $\sigma \leftarrow K(1^\lambda)$ and sets $\mathsf{pp} \triangleq (G \triangleq (\mathbb{G}, \mathbb{G}_T, q, g), \sigma)$ and outputs the public parameters pp.
- Modification to $\mathsf{BRV}^{\star,crs}.\mathsf{Setup}$. The setup algorithm is a run by a voter who, on input the public parameters $\mathsf{pp} \triangleq (\mathbb{G}, \mathbb{G}_T, q, g)$, samples $s \leftarrow \mathbb{Z}_q$ and outputs the pair of public- and secret-keys $(\mathsf{Pk} \triangleq (\mathsf{Pk}^{(1)} \triangleq g^s, \mathsf{Pk}^{(2)} \triangleq \pi), \mathsf{Sk} \triangleq s)$, where π is a NIZK proof for R computed as $P(\sigma, (G, \mathsf{Pk}^{(1)}, s))$.
- The $\mathsf{BRV}^\star.\mathsf{PkVerify}$ algorithm. $\mathsf{BRV}^\star$ also comes with a public-key verification algorithm $\mathsf{BRV}^\star.\mathsf{PkVerify}$ algorithm that, on input the public parameters $\mathsf{pp} \triangleq (G, \sigma)$ and a PK $\mathsf{Pk} = (\mathsf{Pk}^{(1)}, \pi)$, outputs the bit 1 if and only if $V(\sigma, (G, \mathsf{Pk}^{(1)}), \pi) = 1$.

3.2 SchnorrVot and its variants

An outline of SV has been described in Section 1. We defer the details to the full version [Aut22].

3.2.1 $\mathsf{SV}^\star$ and $\mathsf{SV}^{\star,crs}$

To achieve strong soundness (cf. Definition 4), we need to modify SV in a way that is identical to what done for BRV (cf. constructions of $\mathsf{BRV}^\star$ and $\mathsf{BRV}^{\star,crs}$). We call the modified scheme in the RO model that uses Schnorr's PoKs $\mathsf{SV}^\star$ and the one in the CRS model $\mathsf{SV}^{\star,crs}$.

3.2.2 Binding secure variants of our protocols

Our strongly secure protocols can be easily adapted to be binding (cf. Definition 4 in the following way. The modified prover does not not include the sets S_0, S_1 as part of the proof. The modified verifier gets as input S_0, S_1, instead of m_0, m_1, and runs the original verifier on $m_0 = |S_0|, m_1 = |S_1|$.

For $\mathsf{SV}^\star$, $\mathsf{SV}^{\star,crs}$ we need to do an additional modification, in particular we need to have a deterministic (as opposed to randomised) vote algorithm. This is done by setting the value r in the algorithm to be computed as $\mathsf{Hash}(\mathtt{id}, v)$ instead of being a random element of $\mathbb{Z}_q$. With a slight abuse of notation, we will use the same names $\mathsf{BRV}^\star$, $\mathsf{BRV}^{\star,crs}$, $\mathsf{SV}^\star$, $\mathsf{SV}^{\star,crs}$ to refer to the binding and strong secure variants of our protocols.

3.2.3 $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$

In both our SNARVs BRV and SV the verifier needs to take a product of PKs. For instance, consider SV (similar consideration also holds for BRV). For all $v \in \{0,1\}$, the verifier computes $H_v \triangleq \prod_{i \in S_v} \mathsf{Pk}_i$. We now show how the SNARV can be changed so that the verifier just computes 2 (respectively 4) more pairings plus 1 (respectively 2) more exponentiation plus $|S_0| + |S_1|$ additions modulo q rather than $|S_0| + |S_1|$ group operations for multisig (respectively referendum) mode. The setup algorithm Setup, on input the public parameters pp, outputs the pair $(\mathsf{Pk} \triangleq (\mathsf{Pk}^{(1)} \triangleq g^s, \mathsf{Pk}^{(2)} \triangleq \bar{g}^t, \mathsf{Pk}^{(3)} \triangleq s + t \mod q), \mathsf{Sk} \triangleq s)$, where $\bar{g} \triangleq \mathsf{Hash}(g)$, and s, t are randomly sampled in $\mathbb{Z}_q$.

The prover algorithm Prov is unchanged except that it also outputs as part of the proof the following. For $v \in \{0,1\}$, prover computes and outputs $H_v \triangleq \prod_{i \in S_v} \mathsf{Pk}_i^{(1)}, \bar{H}_v \triangleq \prod_{i \in S_v} \mathsf{Pk}_i^{(2)}$. Verifier Ver is unchanged except for the following. The algorithm Ver computes $\bar{g} \triangleq \mathsf{Hash}(g)$ and for all $v \in \{0,1\}$, computes $T_v \triangleq g^{\sum_{i \in S_v} \mathsf{Pk}_i^{(3)}}$. Ver first verifies that for all $v \in \{0,1\}$, $\mathbf{e}(T_v/H_v, \bar{g}) = \mathbf{e}(\bar{H}_v, g)$. We call these additional checks the *Z checks*. The Z checks guarantee H_v is correctly computed and thus the other checks can be executed identically as in SV. We call the variants of BRV (respectively SV) with this optimisation BRV^{opt} (respectively SV^{opt}). To get strong soundness and bindingness we need to add a Chaum-Pedersen proof of knowledge of well-formedness of the DH tuple $\left(g, \bar{g}, \mathsf{Pk}_i^{(1)}, \bar{g}^{\mathsf{Pk}_i^{(1)}}/\mathsf{Pk}_i^{(2)}\right)$, and we assume that at registration phase the PK is checked for well-formedness. We denote by $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$ the strong soundness variants resulting from this optimisation and with a slight abuse, we will use these names also to refer to the binding variants that do not include the sets as part of the proof.

4 Security analysis of our protocols

Our security reductions will focus on the soundness experiment SndExp of 4 for BRV as the soundness proof of SV uses essentially identical argument. It will be useful to make the following definition.

Definition 5. *let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \geq 0}$ be an adversary to either $\mathsf{SndExp}^{\mathsf{BRV,n}}$ or $\mathsf{SndExp}^{\mathsf{SV,n}}$ whose output is of the form $(\boldsymbol{id}, \pi = (\alpha, S_0, S_1))$ for some binary string α and disjoint sets $S_0, S_1 \subseteq [n]$. For $i \in S_0 \cup S_1$ denote by $v(i) \in \{0,1\}$ the unique bit such that $i \in S_{v(i)}$. For $v \in \{0,1\}$, let $S_v^q \subseteq [n]$ be the set of all indicies i such that $\mathcal{A}$ asked the query $(\boldsymbol{id}, v(i), i)$ to $\mathsf{VoteOracle}$ and $F \overset{\triangle}{=} (S_0 \cup S_1) \setminus (S_0^q \cup S_1^q \cup C)$ (regarded as random variables).*

To clarify our soundness proof for BRV, it is convenient to have the following definition and proposition.

Definition 6. *When $\mathcal{A} \overset{\triangle}{=} \{\mathcal{A}_\lambda\}_\lambda$ is an adversary in $\mathsf{SndExp}^{\mathsf{BRV,n}}$ we set $\mathsf{Blt}_i \overset{\triangle}{=} \mathsf{Hash}(\boldsymbol{id}, v(i))^{\mathsf{Sk}_i}$ and for any $j \in S_0 \cup S_1$, we define $\gamma_j \overset{\triangle}{=} \gamma / (\prod_{i \in S_{v(j)}, i \neq j} \mathsf{Blt}_i \cdot \prod_{i \in S_{1-v(j)}} \mathsf{Blt}_i)$.*

Proposition 4.

$$\Pr\left[F \neq \emptyset \wedge \forall j \in S_0 \cup S_1 : \gamma_j = \mathsf{Blt}_j \,\middle|\, \mathsf{SndExp}^{\mathcal{A}_\lambda, \mathsf{BRV}, \lambda, \mathsf{n}} = 1 \right] = 1.$$

Proof. Assuming that $\mathcal{A}$ satisfies that winning condition of SndExp we clearly have $F \neq \emptyset$ and a direct computation using pairing properties of $e : \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ shows that $\gamma = \prod_{i \in S_0} \mathsf{Blt}_i \cdot \prod_{i \in S_1} \mathsf{Blt}_i$ (alternatively one can directly show that the RHS satisfies the winning condition hence must be equal to γ). Since $\gamma = \prod_{j \in S_0 \cup S_1} \gamma_j$, the winning condition of SndExp implies that $\forall j \in S_0 \cup S_1 : \gamma_j = \mathsf{Blt}_j$ as required.

4.1 Security analysis of BatRaVot and its variants

Theorem 5. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \overset{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2).

Let $n > 0$ be an integer and let $\mathsf{BRV} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ be the SNARV for parameter n of Construction 3. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) then BRV is a secure administration-free SNARV for parameter n in the RO model.

Proof.

- Completeness. It is easy to see that BRV satisfies completeness.
- Unforgeability. The unforgeability comes from the fact that the tuple $(\mathsf{BRV.Setup}', \mathsf{BRV.Vote}, \mathsf{BRV.BalVerify})$ is the BLS DS scheme [BLS01] over symmetric bilinear groups (cf. Definition 4 for the definition of $\mathsf{BRV.Setup}'$) or recalling Lemma 2.
- Soundness. For an adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda > 0}$ we denote $\mathsf{SndExp}^{\mathcal{A}_\lambda} \equiv \mathsf{SndExp}^{\mathcal{A}_\lambda, \mathsf{BRV}, \lambda, \mathsf{n}}$ the random variable representing the result of the soundness experiment. Let $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda > 0}$ be a nuPPT adversary such that $\Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1\right] = t(\lambda)$ is a non-negligible function of the security parameter λ. We will assume that for every λ, $\mathcal{A}_\lambda$ always completes the experiment, does not repeat queries to either Hash or $\mathsf{VoteOracle}$ and its outputs S_0, S_1 satisfy $S_0 \cap S_1 = \emptyset$. This assumption is no loss in generality since from any adversary $\mathcal{A} \overset{\triangle}{=} \{\mathcal{A}_\lambda\}_\lambda$ that does not satisfy it, we can construct an adversary $\mathcal{A}' = \{\mathcal{A}'_\lambda\}_\lambda$ that satisfies this assumption with $\Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1\right] = \Pr\left[\mathsf{SndExp}^{\mathcal{A}'_\lambda} = 1\right]$.
  Let $Q \equiv Q(\lambda)$ be a bounding polynomial for the run-time of $\mathcal{A}_\lambda$ and $\mathsf{pp} = (\mathbb{G}, g, q, e) \equiv (\mathbb{G}_\lambda, \mathbb{G}_{T,\lambda}, g_\lambda, q_\lambda, e_\lambda) \leftarrow \mathsf{BiGen}(1^\lambda)$ the public parameters of the experiment. We will assume that for each λ, the RO queries to Hash in SndExp are answered in the following way. At the beginning of the experiment, Experimenter chooses a vector $\vec{x} = (x_i)_{i \in [Q]}$ of random elements from $\mathbb{Z}_q$. For any $i \in [Q]$, let $m_i = (\mathtt{id}_i, v_i)$ be the i-th RO query $\mathcal{A}_\lambda$ asks to the RO Hash if such query exists, otherwise let $m_i \overset{\triangle}{=} \perp$. The i-th query $\mathcal{A}_\lambda$ makes to Hash is answered by $\mathsf{Hash}(m_i) \overset{\triangle}{=} g^{x_i}$. We construct a nuPPT adversary $\mathcal{B} \overset{\triangle}{=} \{\mathcal{B}_\lambda\}_{\lambda > 0}$ such that for any λ, $\mathcal{B}_\lambda$ is an adversary to the experiment $\mathsf{CDHExp}^{\mathcal{B}_\lambda, \mathsf{BiGen}, \lambda}$ that wins in a non-negligible probability. $\mathcal{B}_\lambda$ works as follows.
  - **Parameters generation.** $\mathcal{B}_\lambda$ receives as input (cf. Definition 2) a symmetric bilinear group instance $\mathsf{pp} = (\mathbb{G}, \mathbb{G}_T, g, q) \equiv (\mathbb{G}_\lambda, \mathbb{G}_{T,\lambda}, g_\lambda, q_\lambda) \leftarrow \mathsf{BiGen}(1^\lambda)$ along with the randomness $s \in \{0,1\}^\lambda$ used to generate it, and a pair of group elements $h, u \in \mathbb{G}$ such that $u = g^a$ for some (secret) $a \leftarrow \mathbb{Z}_q$, and does the following.
  - **Setup.** $\mathcal{B}_\lambda$ chooses two random integers $j_B \leftarrow [n], k_B \leftarrow [Q]$. For all $i \neq j_B \in [n]$, $\mathcal{B}_\lambda$ computes $(\mathsf{Pk}_i, \mathsf{Sk}_i) \leftarrow \mathsf{BRV.Setup}(\mathsf{pp})$. $\mathcal{B}_\lambda$ sets $\mathsf{Pk}_{j_B} = u$. This sets implicitly $\mathsf{Sk}_{j_B} = a$. $\mathcal{B}_\lambda$ also sets two variables $\mathtt{id}_B \overset{\triangle}{=} \perp, v_B \overset{\triangle}{=} \perp$, and for any $i \in [Q]$, $\mathcal{B}_\lambda$ chooses $x_i^B \leftarrow \mathbb{Z}_q$.
  - **Simulation of RO queries to Hash.** $\mathcal{B}_\lambda$ will simulate the RO queries that $\mathcal{A}_\lambda$ makes to Hash as follows. Suppose $m_i = (\mathtt{id}_i, v_i)$ is the i-th RO query $\mathcal{A}_\lambda$ asks to the RO Hash (so that $m_i \neq \perp$). $\mathcal{B}_\lambda$ simulates to $\mathcal{A}_\lambda$ the response to m_i as follows. If $i \neq k_B$ then $\mathcal{B}_\lambda$ outputs $g^{x_i^B}$ as an answer for $\mathsf{Hash}(m_i)$. If $i = k_B$, $\mathcal{B}_\lambda$ sets $\mathtt{id}_B \overset{\triangle}{=} \mathtt{id}_{k_B}$ $v_B \overset{\triangle}{=} v_{k_B}$, and outputs h as its answer to $\mathsf{Hash}(m_{k_B})$.

- **Execution of $\mathcal{A}$.** Since $\mathcal{A}_\lambda$ is a PPT algorithm, it can be turned into a deterministic algorithm by inputting a randomness $r \in \{0,1\}^Q$. We let $\mathcal{B}_\lambda$ sample $r \leftarrow \{0,1\}^Q$ and input it to $\mathcal{A}_\lambda$. $\mathcal{B}_\lambda$ runs $C \leftarrow A_\lambda(s, \mathsf{Pk}_1, \ldots, \mathsf{Pk}_{j_B} \overset{\triangle}{=} u, \ldots, \mathsf{Pk}_n; r)$ answering the RO queries as specified above. If $j_B \in C$, $\mathcal{B}_\lambda$ aborts its computation (and hence loses CDHExp). Otherwise, $\mathcal{B}_\lambda$ runs $\mathcal{A}_\lambda$ on input $\{\mathsf{Sk}_i\}_{i \in C}$ simulating RO queries to Hash as specified above and simulates queries to $\mathsf{VoteOracle}$ as follows.
- **Simulation of $\mathsf{VoteOracle}$ queries.** W.l.o.g., we will assume that a $\mathsf{VoteOracle}$ query is of always of the form $(\mathtt{id}_k, v_k, i)$ ie was preceded by an RO query to Hash, $m_k = (\mathtt{id}_k, v_k)$ for some k (this k is uniquely determined since we assume there are no repeated quarries to Hash). Let $(\mathtt{id}_k, v_k, i)$ be a query $\mathcal{A}_\lambda$ made to $\mathsf{VoteOracle}$ (thus, $i \notin C$ by assumption as otherwise $\mathsf{VoteOracle}$ aborts).
  If $i = j_B$, $\mathtt{id}_k = \mathtt{id}_B$ and $v_k = v_B$, $\mathcal{B}_\lambda$ aborts. If $i = j_B$ and either $\mathtt{id}_k \neq \mathtt{id}_B$ or $v_k \neq v_B$, then $\mathcal{B}_\lambda$ outputs $u^{x_k^B} = \mathsf{Hash}(\mathtt{id}_k, v_k)^{\mathsf{Sk}_{j_B}}$. (Observe that this output is distributed as $\mathsf{BRV.Vote}(\mathsf{id_k}, \mathsf{v_k}, \mathsf{Sk_{j_B}})$.)
  Otherwise, $i \neq j_B$ and $\mathcal{B}_\lambda$ outputs $\mathsf{Hash}(m_k)^{\mathsf{Sk}_i} = \mathsf{Vote}(\mathsf{id_k}, \mathsf{v_k}, \mathsf{Sk_i})$, exactly as $\mathsf{VoteOracle}$ would do.
- **Output.** $\mathcal{A}_\lambda$ outputs $(\mathtt{id}, \pi = (\gamma, S_0, S_1))$. If, at this stage, $v_B = \perp$, $\mathcal{B}_\lambda$ aborts the computation. $\mathcal{B}_\lambda$ computes, for every $i \neq j_B \in S_0 \cup S_1$, the group element $\mathsf{Blt}_i \overset{\triangle}{=} \mathsf{Hash}(\mathtt{id}, v(i))^{\mathsf{Sk}_i}$ (with $\mathsf{Hash}(\mathtt{id}, v(i))$ as simulated in the description above). This is computable by $\mathcal{B}_\lambda$ since it has all Sk_i for $i \neq j_B$. Finally, $\mathcal{B}_\lambda$ returns $w \overset{\triangle}{=} \gamma_{j_B}$ as its guess for h^a in CDHExp (see Definition 6).

Before going to the probability analysis let us fix some notations. We denote by $\mathsf{SndExp}^{\mathcal{A}_\lambda}_{\mathsf{CDHExp}^{\mathcal{B}_\lambda}}$ the random variable of the result of the soundness experiment when the environment of $\mathcal{A}_\lambda$ is simulated by $\mathcal{B}_\lambda$ within CDHExp. Similarly, we denote corresponding random variables in the soundness experiment as simulated to $\mathcal{A}_\lambda$ by $\mathcal{B}_\lambda$, by $\mathtt{id}_B, \pi_B, F_B$ and m_i^B. These random variables are defined to have the value $\perp$ when $\mathcal{B}_\lambda$ aborts.

We define a random variable $F^\star \overset{\triangle}{=} \{i \in F \big| \exists k \in [Q] : \ m_k = (\mathtt{id}, v(i))\} \subseteq F$, relative to the (plain) soundness experiment and the corresponding random variable relative to the simulated soundness experiment $F_B^\star \overset{\triangle}{=} \{i \in F_B \big| \exists k \in [Q] : \ m_k^B = (\mathtt{id}, v(i))\} \subseteq F_B$. We will now show that $\Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star = \emptyset\right]$ is a negligible function in λ. Let P be the event that the pairing equation $e(\gamma, g) = e(H_0, g_0)e(H_1, g_1)$ is satisfied. We denote by $v_\star = v(\min F) \in \{0,1\}$ (so that $\min F \in S_{v_\star}$), if such element exist and set $v_\star = \perp$ otherwise. Note first that for any $\gamma', g'_{1-v_\star}, H'_0, H'_1 \in \mathbb{G}$,

$$\Pr\left[P | g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 \wedge \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] = \\ \Pr\left[g_{v_\star} = g'_{v_\star} | g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 \wedge \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] = \frac{1}{|\mathbb{G}|}, \tag{3}$$

where $g'_{v_\star} \in \mathbb{G}$ is the unique element such that $\mathbf{e}(\gamma', g) = \mathbf{e}(H'_{v_\star}, g'_{v_\star}) \cdot \mathbf{e}(H'_{1-v_\star}, g'_{1-v_\star})$ is satisfied. The last equality holds since the event $g_{v_\star} = g'_{v_\star}$ is independent from $F \neq \emptyset \wedge F^\star = \emptyset$ since by definition $\mathcal{A}_\lambda$ did not make the query $(\mathtt{id}, v_\star)$ to Hash and thus $g_{v_\star}$ is independent from the view of $\mathcal{A}_\lambda$.

Thus, by Proposition 4 and equation 3 we get

$$\begin{aligned}
&\Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 | \gamma = \gamma' \wedge \mathsf{F} \neq \emptyset \wedge \mathsf{F}^\star = \emptyset\right] = \Pr\left[P | \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \\
&= \sum_{g'_{1-v_\star}, H'_0, H'_1 \in \mathbb{G}} \Pr\left[P \wedge g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 | \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \\
&= \sum_{g'_{1-v_\star}, H'_0, H'_1 \in \mathbb{G}} \Pr\left[P | g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 \wedge \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \cdot \\
&\qquad \Pr\left[g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 | \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \\
&= \sum_{g'_{1-v_\star}, H'_0, H'_1 \in \mathbb{G}} \frac{1}{|\mathbb{G}|} \cdot \Pr\left[g_{1-v_\star} = g'_{1-v_\star} \wedge H_0 = H'_0 \wedge H_1 = H'_1 | \gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] = \frac{1}{|\mathbb{G}|}.
\end{aligned} \tag{4}$$

It now follows that

$$\begin{aligned}
\Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star = \emptyset\right] &= \sum_{\gamma' \in \mathbb{G}} \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star = \emptyset | \gamma = \gamma'\right] \Pr\left[\gamma = \gamma'\right] \\
&= \sum_{\gamma' \in \mathbb{G}} \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 | \gamma = \gamma' \wedge \mathsf{F} \neq \emptyset \wedge \mathsf{F}^\star = \emptyset\right] \Pr\left[\gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \\
&= \sum_{\gamma' \in \mathbb{G}} \frac{1}{|G|} \cdot \Pr\left[\gamma = \gamma' \wedge F \neq \emptyset \wedge F^\star = \emptyset\right] \leq \frac{1}{|\mathbb{G}|}
\end{aligned} \tag{5}$$

is a negligible function in λ. Since

$$t(\lambda) = \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1\right] = \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star = \emptyset\right] + \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star \neq \emptyset\right]$$

is non-negligible, we deduce that:

$$t'(\lambda) \triangleq \Pr\left[\mathsf{SndExp}^{\mathcal{A}_\lambda} = 1 \wedge \mathsf{F}^\star \neq \emptyset\right] \tag{6}$$

is a non-negligible function in λ.
Recall that we may view $\mathcal{A}_\lambda$ as a deterministic (stateful) algorithm that inputs for its first phase s, $\mathcal{P} \triangleq (\mathsf{Pk}_1, ..., \mathsf{Pk}_n)$ and also a randomness $r \in \{0,1\}^Q$. In particular, we may view $\mathcal{A}_\lambda$ as a function with input $(r, s, \mathcal{P})$. We also view the RO Hash as a function, given Hash has chosen (before execution) a set of group elements $\mathcal{G} \triangleq (g^{x_k})_{k\in[Q]}$ for its responses on queries m_k made by $\mathcal{A}_\lambda$. Lastly, $\mathsf{VoteOracle}$ can be viewed as a function on $\mathcal{S} \triangleq (\mathsf{Sk}_i)_{i=1}^n$, $\mathcal{G}$ and C ie as a function in $(r, s, \mathcal{P}, \mathcal{G}, \mathcal{S})$. We may thus view the random variables $F^\star, F_B^\star$ and π, π_B as functions with input $(r, s, \mathcal{P}, \mathcal{G}, \mathcal{S})$. For a fixed choice of elements $r, s, \mathcal{P}, \mathcal{G}$, we will call the tuple $\mathcal{C} \triangleq (r, s, \mathcal{P}, \mathcal{G})$ a **configuration** and denote the set of all such configurations by Conf. There exists a function $\mathsf{Sk} : \mathsf{Conf} \longrightarrow (\mathbb{Z}_q)^n$, given by $\mathcal{C} \mapsto \mathsf{Sk}_\mathcal{C}$ where $\mathsf{Sk}_\mathcal{C} \triangleq (\mathsf{Sk}_i)_{i=1}^n$ are all secret keys corresponding to elements in $\mathcal{P}$ (this function is generally not efficiently computable).
With this in mind, we may rephrase Equation 6 as

$$\frac{\#\{\mathcal{C} \in \mathsf{Conf} \,\Big|\, \mathsf{SndExp}^{\mathcal{A}_\lambda}(\mathcal{C}, \mathsf{Sk}_\mathcal{C}) = 1 \wedge \mathsf{F}^\star(\mathcal{C}, \mathsf{Sk}_\mathcal{C}) \neq \emptyset\}}{\#\mathsf{Conf}} = t'(\lambda).$$

We will refer to the set $\mathsf{Conf}_w \triangleq \{\mathcal{C} \in \mathsf{Conf} \,\Big|\, \mathsf{SndExp}^{\mathcal{A}_\lambda}(\mathcal{C}, \mathsf{Sk}_\mathcal{C}) = 1 \wedge \mathsf{F}^\star(\mathcal{C}, \mathsf{Sk}_\mathcal{C}) \neq \emptyset\}$ as the set of **winning configurations**. Note that if $\mathcal{C} \in \mathsf{Conf}_w$ is a winning configuration, there exists a **trace**, ie a tuple $(j, k) \equiv (j_\mathcal{C}, k_\mathcal{C})$ such that $j \in F^\star$, $m_k = (\mathtt{id}, v_B)$ and $j \in S_{v_B}$. Thus there is a function $\mathsf{tr} : \mathsf{Conf}_w \longrightarrow [n] \times [Q]$ defined by $\mathsf{tr}\mathcal{C} \triangleq (j_\mathcal{C}, k_\mathcal{C})$. For convenience, we extend tr to be a function $\mathsf{tr} : \mathsf{Conf} \longrightarrow [n] \times [Q] \cup \{\perp\}$ by setting $\mathsf{tr}\mathcal{C} = \perp$ for every $\mathcal{C} \in \mathsf{Conf} \setminus \mathsf{Conf}_w$.
Let us turn attention to $\mathcal{B}_\lambda$. Recall that before executing $\mathcal{A}_\lambda$, $\mathcal{B}_\lambda$ samples randomness $r \leftarrow \{0,1\}^Q$, $s \leftarrow \mathsf{BiGen}(\lambda)$, $(\mathsf{Pk}_i, \mathsf{Sk}_i)_{i=1}^n \leftarrow \mathsf{Setup}$, a set $(x_k^B)_{k\in[Q]}$ of elements in $\mathbb{Z}_q$ and gets two group elements $u, h \in \mathbb{G}$. We define random variables (relative to CDHExp) by $\mathcal{P}_B \triangleq (\mathsf{Pk}_1, ..., \mathsf{Pk}_{j_B} \triangleq u, ..., \mathsf{Pk}_n)$, $\mathcal{G}_B \triangleq (g^{x_1^B}, ..., h, ..., g^{x_Q^B})$ (where u appears in entry j_B and h appears in entry k_B), and $\mathcal{C}_B \triangleq (r, s, \mathcal{P}_B, \mathcal{G}_B)$.
By definition, the last random variable $\mathcal{C}_B$ takes values in Conf and is well-defined even when the execution of $\mathcal{A}_\lambda$ is not completed (ie when $\mathcal{B}_\lambda$ aborts). Note that if $\mathcal{B}_\lambda$ does not abort, it simulates the environment of $\mathcal{A}_\lambda$ indistinguishably from the environment of the plain soundness experiment. This is because all private- and public- keys and all answers of $\mathcal{B}_\lambda$ to queries $\mathcal{A}_\lambda$ makes to Hash or $\mathsf{VoteOracle}$, are randomly sampled elements from a uniform distribution in either $\mathbb{Z}_q$ or $\mathbb{G}$ that either $\mathcal{B}_\lambda$ sampled itself, or gotten as input from CDHExp. More precisely, if $\mathsf{CDHExp}^{\mathcal{B}_\lambda} \neq \perp$, then $\pi(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B}) = \pi_B(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B})$ and $F^\star(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B}) = F_B^\star(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B})$. In fact, even when $\mathcal{B}_\lambda$ aborts, it simulates the environment of $\mathcal{A}_\lambda$ indistinguishably from the environment of the plain soundness experiment until the abortion.
We define another random variable by $\mathsf{tr}\mathcal{C}_B \triangleq \mathsf{tr} \circ \mathcal{C}_B$. Since all elements in $\mathcal{C}_B$ are randomly sampled from a uniform distribution (either by $\mathcal{B}_\lambda$ or by CDHExp), $\Pr[\mathsf{tr}\mathcal{C}_B \neq \perp] = \Pr[\mathsf{tr}\mathcal{C} \neq \perp]$, where $\mathcal{C} \leftarrow \mathsf{Conf}$ stands for the random variable of the uniform distribution on Conf. Equation 6 then implies that $\Pr[\mathsf{tr}\mathcal{C}_B \neq \perp] = t'(\lambda)$. We now define an event in CDHExp by $E_B \triangleq \{\mathsf{tr}\mathcal{C}_B \neq \perp\}$. We also define a (sub-)event $E'_B \subseteq E_B$ as $E'_B \triangleq \{\mathsf{tr}\mathcal{C}_B \neq \perp \wedge \mathsf{tr}\mathcal{C}_B = (j_B, k_B)\}$. Observe that if we are in event E'_B ie $\mathcal{B}_\lambda$ executes $\mathcal{A}_\lambda$ with a winning configuration $\mathcal{C}_B$ and $\mathsf{tr}\mathcal{C}_B = (j_B, k_B)$, then $\mathcal{B}_\lambda$ does not abort. This is because in that case, $j_B \notin C$ since $j_B \in F^\star(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B}) = F_B^\star(\mathcal{C}_B, \mathsf{Sk}_{\mathcal{C}_B})$, and $\mathcal{A}_\lambda$ did not ask the query $(\mathtt{id}, v_B, j_B)$ as this would have caused $\mathsf{VoteOracle}$ to abort in the plain soundness experiment. Moreover, in the event E'_B, $\mathsf{SndExp}^{\mathcal{A}_\lambda}_{\mathsf{CDHExp}^{\mathcal{B}_\lambda}} = 1$ since $\mathcal{B}_\lambda$ executed $\mathcal{A}_\lambda$ with a winning configuration and simulated the environment of $\mathcal{A}_\lambda$ indistinguishably from the environment of the plain soundness experiment. It follows that in the event E'_B, $\mathcal{B}_\lambda$ wins since its output is γ_{j_B} that by Proposition 4 must satisfy $\gamma_{j_B} = \mathsf{Blt}_{j_B} = \mathsf{Hash}(m_{k_B})^{\mathsf{Sk}_{j_B}} = \mathsf{Hash}(\mathtt{id}, v_B)^{\mathsf{Sk}_{j_B}} = h^a$.
With this in mind, let us analyse the probability that $\mathcal{B}_\lambda$ wins CDHExp.

$$\Pr\left[\mathsf{CDHExp}^{\mathcal{B}_\lambda} = 1\right] \geq \Pr[E'_B] = \Pr[E_B \wedge E'_B] = \Pr[E_B]\Pr[E'_B | E_B] \geq t'(\lambda)\frac{1}{nQ}.$$

Thus, $\mathcal{B}_\lambda$ wins CDHExp in a non-negligible probability, in contradiction to the CDH assumption. We conclude that no nuPPT adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_\lambda$ can win the soundness experiment in a non-negligible probability so that BRV satisfies soundness.

Theorem 6. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \triangleq (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2). Let $n > 0$ be an integer and let

$$\mathsf{BRV}^{\star,crs} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{PkVerify})$$

be the SNARV for parameter n of Section 3.1.1 based on a NIZK $\mathsf{NIZK} \triangleq (K, V, P)$. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) and NIZK is a perfectly complete, computationally ZK and simulation-sound extractable (formal definitions can be found in [GO14,Gro06] and will be provided in the future full version) then $\mathsf{BRV}^{\star,crs}$ is a strongly secure administration-free SNARV for parameter n in the CRS model.

Proof (Omitted). The idea is that the PoK allows the simulator to extract all SKs needed for the simulation and thus the strong soundness security is reduced to (standard) soundness as in Theorem 5. Details will be provided in the future full version.

Theorem 7. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \stackrel{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2). Let $n > 0$ be an integer and let $\mathsf{BRV}^\star = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{PkVerify})$ be the SNARV for parameter n of Section 3.1.1. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) then $\mathsf{BRV}^\star$ is a strongly secure administration-free SNARV for parameter n in the RO model.

Proof (Sketch). The proof is similar to the one for Theorem 6 except that, instead of working in the CRS model with a general NIZK in the CRS model that is perfectly complete, computationally ZK and simulation extractable, we assume that the Schnorr's PoK satisfies the corresponding properties in the programmable RO model. We defer to [FKMV12] for definitions of NIZK in the RO model satisfying those properties and for the fact that the Schnorr's PoK satisfies such properties.

4.2 Security analysis of SchnorrVot and its variants

Theorem 8. Let Hash, be an hash function that takes as input a bilinear group description $G \stackrel{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ and maps binary strings in $\{0,1\}^\star$ into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2).

Let $n > 0$ be an integer and let $\mathsf{SV} = (\mathsf{Gen}, \mathsf{Setup}, \mathsf{Census}, \mathsf{Vote}, \mathsf{BalVerify}, \mathsf{Prove}, \mathsf{Ver})$ be the SNARV for parameter n of Section 1. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) then SV is a secure administration-free SNARV for parameter n in the RO model.

Proof (Omitted). The proof is very similar to that of Theorem 5 and will be provided in the future full version.

We state the following two theorems without proofs since they are identical to the versions for $\mathsf{BRV}^{\star,crs}$ and $\mathsf{BRV}^\star$ respectively.

Theorem 9. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \stackrel{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2). Let $n > 0$ be an integer and let $\mathsf{SV}^{\star,crs}$ be the SNARV for parameter n of Section 3.2.1 based on a NIZK $\mathsf{NIZK} \stackrel{\triangle}{=} (K, V, P)$. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) and NIZK is perfectly complete, computationally ZK and simulation extractable then $\mathsf{SV}^{\star,crs}$ is a strongly secure administration-free SNARV for parameter n in the CRS model.

Theorem 10. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \stackrel{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2). Let $n > 0$ be an integer and let $\mathsf{SV}^\star$ be the SNARV for parameter n of Section 3.2.1. If the CDH Assumption holds for the symmetric bilinear group generator BiGen (cf. Assumption 1) then $\mathsf{SV}^\star$ is a strongly secure administration-free SNARV for parameter n in the RO model.

4.3 Bindingness of our protocols

Recall that the binding variants of our protocols are identical to the strongly secure protocols except that the sets S_0, S_1 are not part of the proof and the verifier is changed accordingly. Moreover, with a slight abuse of notation we denote them by the same names, in particular $\mathsf{BRV}^\star$, $\mathsf{BRV}^{\star,crs}$, $\mathsf{SV}^\star$, $\mathsf{SV}^{\star,crs}$. Proposition 4 implies that a given RO, set of PKs and proof π and a pair of sets S_0, S_1, induce a unique set of valid ballots. Since the ballots in $\mathsf{BRV}^\star$ and $\mathsf{BRV}^{\star,crs}$ are deterministic, with all but negligible probability over the random choices of the RO, no adversary is able to find an accepted proof for two different pairs of sets. The binding strong security property is also seen to follow similarly to the strong soundness property. Regarding the binding variants of $\mathsf{SV}^\star$ and $\mathsf{SV}^{\star,crs}$ we recall that we changed the vote algorithm so as to be deterministic and in particular we set $r = \mathsf{Hash}(\mathtt{id}, v)$. In this case, a proposition similar to Proposition 4 also follows and as before we have that a given RO, set of PKs and proof π and a pair of sets S_0, S_1, induce a unique set of valid ballots. As consequence, with all but negligible probability over the random choices of the RO, no adversary is able to find an accepted proof for two different pairs of sets. The binding strong security property also follows similarly to the strong soundness property.

4.4 Security analysis of the improved versions of our protocols

In Section 3.2.3 we described an improvement to asymptotically improve both BRV, SV and the strong secure variants $\mathsf{BRV}^\star$ and $\mathsf{SV}^\star$. We denoted the protocols resulting from this improvement by BRV^{opt} SV^{opt} and by $\mathsf{BRV}^{\star,opt}$, $\mathsf{SV}^{\star,opt}$ for the strong secure variants . Moreover, the improvement also applies to the binding variants of our protocols. We now state the strong soundness and bindingness of $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$(the soundness of BRV^{opt} and SV^{opt} follows as a corollary).

Theorem 11. Let Hash be an hash function that takes as first parameter a symmetric bilinear group instance $G \stackrel{\triangle}{=} (\mathbb{G}, \mathbb{G}_T, g, q, e)$ for some parameter $\lambda > 0$ and a binary string in $\{0,1\}^\star$ and maps into $\mathbb{G}$. Let BiGen be a symmetric bilinear generator (cf. Definition 2). Let $n > 0$ be an integer and let $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$be the SNARVs for parameter n of Section 3.2.3. If the KEA1 [BP04] and the CDH Assumptions hold for the symmetric bilinear group generator BiGen (cf. [BP04] and Assumption 1) then $\mathsf{BRV}^{\star,opt}$ and $\mathsf{SV}^{\star,opt}$are strongly secure administration-free SNARV for parameter n in the RO model. Moreover, their variants in which the sets are not part of the proof are binding in the RO model.

Proof (Omitted). The idea of the proof is that if the adversary can compute a proof for H_v that is ill-formed then, by the KEA1 Assumption, it is possible to extract a randomness such that from this randomness a well-formed H'_v and a proof π' such that π' is a proof that is accepted for H'_v and then the security follows as in the analysis of strong soundness. We defer to the future full version for the details.

5 Conclusions and future work

In this work we studied the following e-voting problem. A voter is registered in a system that may be a traditional organisation or a permissionless Blockchain, and is possibly associated with a weight. At any time after registration, the voter can be called to express a preference in favour or against a certain proposal. A verifier needs to tally the preferences of all voters, possibly taking into account the weights, in order to announce the result and trigger some action.

When efficiency is not crucial, like in a traditional election on a server, this problem may be solved using standard digital signature schemes.

Instead, in a decentralised system like the Ethereum network it is crucial to tally the result with minimal computation and space since verifying a large amount of signatures can be costly. We put forth a formal model that encompasses both previous private voting solutions and provided novel non-private voting solutions that are optimised for main e-voting scenarios on Blockchain.

We leave as future work the implementation of our protocols for the Ethereum Virtual Machine (e.g., using the precompiles for the bn254 curve) and a detailed analysis of their Gas costs.

References

aEBK22. arnaucube, Pau Escrich, Roger Baig, and Alex Kampa. Ovote (v0.5) : Off-chain voting with on-chain trustless execution, 2022. https://github.com/aragonzkresearch/research/blob/main/drafts/ovote.pdf.

Aut22. Anonymous Authors. Snarv: Succinct non-interactive argument of voting. *Temporary web link.*, 2022.

BDN18. Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology - ASIACRYPT 2018 - Proceedings, Part II.* Springer, 2018.

BF01. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, August 2001.

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432. Springer, May 2003.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, December 2001.

BP04. Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 273–289. Springer, August 2004.

CHKM10. Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Designs, Codes and Cryptography*, 55(2-3):141–167, 2010.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79. Springer, December 2012.

GO14. Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. *J. Cryptol.*, 27(3), 2014.

Gro06. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, December 2006.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, June 2011.

KL07. Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007.

MOR01. Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 01: 8th Conference on Computer and Communications Security*, pages 245–254. ACM Press, November 2001.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, May 1996.