







# ZK-SNARKs for Ballot Validity: A Feasibility Study

Nicolas Huber<sup>(✉)</sup>, Ralf Küsters<sup></sup>, Julian Liedtke<sup></sup>, and Daniel Rausch<sup></sup>

University of Stuttgart, Stuttgart, Germany  
firstname.secondname@sec.uni-stuttgart.de

**Abstract.** Electronic voting (e-voting) systems have become more prevalent in recent years, but security concerns have also increased, especially regarding the privacy and verifiability of votes. As an essential ingredient for constructing secure e-voting systems, designers often employ zero-knowledge proofs (ZKPs), allowing voters to prove their votes are valid without revealing them. Invalid votes can then be discarded to protect verifiability without compromising the privacy of valid votes. General purpose zero-knowledge proofs (GPZKPs) such as ZK-SNARKs can be used to prove arbitrary statements, including ballot validity. While a specialized ZKP that is constructed only for a specific election type/voting method, ballot format, and encryption/commitment scheme can be more efficient than a GPZKP, the flexibility offered by GPZKPs would allow for quickly constructing e-voting systems for new voting methods and new ballot formats. So far, however, the viability of GPZKPs for showing ballot validity for various ballot formats, in particular, whether and in how far they are practical for voters to compute, has only recently been investigated for ballots that are computed as Pedersen vector commitments in an ACM CCS 2022 paper by Huber et al. Here, we continue this line of research by performing a feasibility study of GPZKPs for the more common case of ballots encrypted via Exponential ElGamal encryption. Specifically, building on the work by Huber et al., we describe how the Groth16 ZK-SNARK can be instantiated to show ballot validity for arbitrary election types and ballot formats encrypted via Exponential ElGamal. As our main contribution, we implement, benchmark, and compare several such instances for a wide range of voting methods and ballot formats. Our benchmarks not only establish a basis for protocol designers to make an educated choice for or against such a GPZKP, but also show that GPZKPs are actually viable for showing ballot validity in voting systems using Exponential ElGamal.

## 1 Introduction

A prominent approach for constructing secure e-voting systems is the homomorphic aggregation of ballots. In such systems, a vote/ballot is a vector of numbers, with one number per possible choice in the election. Typically, a choice corresponds to a candidate that the voter can give one or several votes/points, so

in an election with  $n_{\text{cand}}$  candidates, a vote would be a vector of length  $n_{\text{cand}}$ . An additively homomorphic encryption or commitment scheme is then used to hide the vote. This scheme is typically applied component-wise, i.e., a vote vector of length  $n_{\text{cand}}$  results in an encrypted ballot<sup>1</sup> consisting of  $n_{\text{cand}}$  many ciphertexts/commitments. When using commitment schemes for hiding votes, voters have to send (shares of) an (encrypted) opening of their commitment. Currently, Exponential ElGamal (EEG) encryption is the most relevant option in practice [2, 11]. To tally the election, all encrypted ballots are first homomorphically aggregated (component-wise) to obtain a single aggregated encrypted ballot that hides individual votes. This aggregated ballot is decrypted to obtain the *aggregated tally* consisting of a list of the total votes/points for each candidate.

**Proofs for Ballot Validity.** For the above approach of aggregation-based e-voting to be reasonable, one needs to ensure that all encrypted ballots used for aggregation are well-formed, i.e., that they contain a valid vote. The standard approach is to have voters use zero-knowledge proofs (ZKPs) to prove *ballot validity* during ballot submission.

A ZKP for ballot validity proves that the vote contained in an encrypted ballot belongs to the set of votes permitted by the current election. We call this set a *choice space* in the following. For instance, consider the straightforward case of single-vote elections, where a voter can cast a single vote for one out of  $n_{\text{cand}}$  candidates. A corresponding choice space can be defined as follows, where  $v_i$  denotes the number of votes given to candidate  $i$  in a ballot:

$$C_{\text{single}} := \left\{ (v_1, \dots, v_{n_{\text{cand}}}) \mid v_i \in \{0, 1\}, \sum_{i=1}^{n_{\text{cand}}} v_i \in \{0, 1\} \right\}.$$

A voter is supposed to choose her ballot  $\mathbf{b}$  as a vector from this set, i.e.,  $\mathbf{b} \in C_{\text{single}}$ . The voter then computes an encrypted ballot  $\mathbf{c}$  from  $\mathbf{b}$  and submits  $\mathbf{c}$  alongside a ZKP which shows that  $\mathbf{c}$  was obtained by encrypting a ballot  $\mathbf{b} \in C_{\text{single}}$ . Ballots without valid ZKP are discarded by the voting system, ensuring that even malicious voters can contribute only one vote for one candidate.

**State of the Art.** A ZKP for ballot validity depends on the underlying choice space and the encryption/commitment scheme used to obtain  $\mathbf{c}$ . Therefore, ZKPs for ballot validity have usually been designed and proven secure only for specific combinations of choice spaces and (classes of) encryption/commitment schemes.

For example, Helios 2.0 [2] and Belenios [11, 18] support  $C_{\text{single}}$  with component-wise EEG encryption. That is,  $\mathbf{c}$  is a vector of EEG ciphertexts  $c_i$ , each encrypting one  $v_i$ . The ballot validity ZKPs in Helios and Belenios are based on disjunctive Chaum-Pedersen proofs [9, 13], which show that an EEG ciphertext encrypts a value from a specific set  $S$ . Concretely, for  $C_{\text{single}}$  one considers the set  $S = \{0, 1\}$ . Voters then compute a full proof for ballot validity by combining (i) one proof for each ciphertext  $c_i$  showing that the corresponding

<sup>1</sup> For simplicity of presentation, we will often only say “encrypted ballot” to refer to both cases, i.e., encryption or commitments.

plaintext  $v_i$  is from  $S$ , and (ii) one proof for the homomorphic sum of all  $n_{\text{cand}}$  ciphertexts  $c_i$  showing that the decryption lies in  $S$ . Generalizing single-vote, one can also use disjunctive Chaum-Pedersen proofs for showing ballot validity for multi-vote elections, where voters can assign up to  $n_{\text{max}}$  votes to candidates of their choice (up to a limit  $t$  for any individual candidate)[2, 11, 18]. However, for larger values of  $n_{\text{max}}$  (and  $t$ ) this quickly becomes too inefficient. In such cases, one can replace disjunctive Chaum-Pedersen proofs with range-proofs [31].

Designing efficient ZKPs for ballot validity becomes an increasingly difficult task for more complex voting methods and ballot formats. As an example, consider the class of Borda count election methods, where points are assigned to candidates based on a ranking chosen by the voter. Such a ranking creates dependencies between points assigned to different candidates which cannot be captured by the above approach but requires different ZKPs. The ZKPs for Borda Ballots proposed in [21] only work when ties between candidates are not allowed. The proofs in [27] work for Borda ballots that allow ties at the last place. Both constructions are based on arguments for the correctness of a shuffle. To the best of our knowledge, the only work that has considered ZKPs for Borda ballots with ties at arbitrary positions is the Kryvos system [25], which uses GPZKPs (see below).

Condorcet methods are another class of elections that use very complex choice spaces and thus require advanced validity proofs. These ranked voting methods aim to determine a Condorcet winner who would win against every other candidate in a direct comparison. In [12], two ZKPs for validity of Condorcet ballots have been described. Both ZKPs are for ballots that are encrypted using EEG, but they differ in the ballot formats that are used to encode a vote.

Altogether, while efficient ZKPs for proving ballot validity exist for many election types, they are generally designed only for a specific voting method, ballot format/choice space, and (class of) encryption or commitment scheme. Designing an e-voting system for new types of elections with new ballot formats, therefore, usually entails constructing and proving the security of suitable ZKPs. **Using GPZKPs for Ballot Validity.** A promising alternative which we investigate in this work are *general purpose zero-knowledge proofs (GPZKPs)*. GPZKPs can, in theory, show arbitrary statements, including ballot validity for any ballot and election. The main task left for a protocol designer using a GPZKP is to propose an optimized circuit for computing the statement that should be proven so that the resulting GPZKP instance is sufficiently efficient. Thus, GPZKPs have the potential to simplify the process of designing electronic election systems, enable faster prototyping if a new type of election with a different ballot format is implemented, and allow for supporting ballot formats that are so far out of reach of current *specialized ZKPs* which are constructed for showing a specific statement.

While GPZKPs such as ZK-SNARKs (zero-knowledge succinct non-interactive arguments of knowledge, called just SNARKs in the following) have recently gained traction in several areas such as blockchains [24], they have so far mostly gone unnoticed in the area of e-voting. In [15], techniques based on inner product

arguments (which are commonly used for constructing GPZKPs [8]) are used for proving that a vector of ciphertexts encrypts bits. This can be used for proving validity of, e.g., single-vote ballots and can drastically outperform the Chaum-Pedersen-based approach we described above. The first (and so far the only) work that considered GPZKPs for more complex relations in encrypted ballots is the Kryvos system [25]. While not their primary focus, as a side result the authors of [25] were able to show that and how the state-of-the-art Groth16 SNARK [22] can be instantiated to obtain practical ballot validity proofs for a wide variety of common election types as long as encrypted ballots are computed by using Pedersen Vector Commitments (PVCs). Among others, and as mentioned above, using this GPZKP, they obtained the first (practical) ZKP for showing validity of Borda ballots that allow ties at arbitrary positions. However, the focus of the Kryvos system is the design of a publicly tally-hiding system rather than the design of ballot validity proofs. Hence, the authors did not further investigate the viability of GPZKPs for ballot validity beyond the uncommon case of PVCs.

It remains unclear if GPZKPs for ballot validity are practical beyond these specific settings, notably for complex ballots in the standard case of (component-wise) EEG. We note that specialized ZKPs, which have been constructed for and are tailored towards a specific election system, voting method, ballot format/choice space, and encryption/commitment scheme, can, of course, be more optimized and hence more efficient than GPZKPs. The advantage of GPZKPs lies in their generality, which, if shown to be practical in at least some settings, would open up a simple and generic approach to building new e-voting systems.

**Contributions.** In this work we perform a feasibility study that investigates viability and limits of GPZKPs for ballot validity for many ballot formats in commonly used EEG-based e-voting systems. On a technical level, we build on the techniques for instantiating the Groth16 SNARK established in Kryvos [25] and explain how they can, in principle, be used for proving ballot validity when ballots are encrypted component-wise via EEG. As part of this, we also provide a detailed description of their techniques for proving ballot validity, which had only been briefly sketched in [25] with most information left to their implementation.

As the main contribution of our feasibility study, we have implemented several circuits and benchmarked and compared the corresponding Groth16 SNARK instances for showing ballot validity for EEG encryption for a wide range of voting methods and corresponding choice spaces.<sup>2</sup> This includes not only major existing ones: Single- and Multi-Vote, Borda Count, and Condorcet methods. To investigate the potential and limits of GPZKPs for developing and supporting new voting methods and systems, we also consider two new variants of Multi-Vote. These variants introduce non-trivial conditions on ballot formats and mainly serve demonstration purposes. We are not aware that they are currently used in real elections.

To summarize our findings, our benchmarks show that all of these instances are actually practical, both for simple and complex voting methods and choice spaces. Performance depends mainly on the number of candidates. Interestingly,

---

<sup>2</sup> All of our implementations are available at [26].

however, the performance of these Groth16 instances is otherwise essentially independent of the complexity of the underlying choice space. That is, introducing and proving additional conditions on the format of ballots, even multiple highly complex ones, barely changes overall performance.

Altogether *our work establishes for the first time that current GPZKPs are a viable option even for complex ballot formats for commonly used Exponential ElGamal-based e-voting systems*, which opens up new options for supporting different voting methods. Our benchmarks further provide a basis for protocol designers to make an educated choice for or against a Groth16-based ballot validity ZKPs.

## 2 Preliminaries: GPZKPs, SNARKs, Groth16

A general purpose zero-knowledge proof (GPZKP) system takes as input an arbitrary indicator function  $f_R : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}$  for some binary relation  $R$  such that  $f_R(x, w) = 1$  iff  $(x, w) \in R$  for a public *statement*  $x$  and a secret *witness*  $w$ . It then allows for computing a zero-knowledge proof (ZKP) which shows the existence/knowledge of  $w$  such that  $f_R(x, w) = 1$ . In the following, we only consider proofs of knowledge as typically needed in e-voting [30].

To be practical for showing ballot validity, good prover efficiency and small proof sizes are crucial: Impatient voters have to be able to compute and then transmit the GPZKP using their own personal devices within reasonable time and possibly having only little bandwidth available. While election verification is less time critical, verification speed should at least be moderately fast and, again, proof sizes should be small since proofs from all voters need to be downloaded.

Of the various GPZKP systems [3, 6, 10, 17, 20, 22, 29], SNARKs fit these requirements best. Following [25], we use the highly efficient state-of-the-art Groth16 SNARK [22] that offers constant small proof size of less than 1 kilobyte with (almost) constant verification time of about a few milliseconds on a standard PC<sup>3</sup> - independently of the function  $f_R$ . It further achieves fast polynomial proving time and thus scales well even for highly complex functions  $f_R$ . The Groth16 SNARK is therefore an ideal candidate for showing ballot validity.

A bit simplified, Groth16 consists of three algorithms: **Setup**, **Prove**, and **Verify**. The **Setup**( $f_R$ ) algorithm generates two common reference strings,  $\text{CRS}_{\text{EK}}$  (*evaluation key CRS*) and  $\text{CRS}_{\text{VK}}$  (*verification key CRS*) that depend on  $f_R$ .  $\text{CRS}_{\text{VK}}$  is a much smaller substring of  $\text{CRS}_{\text{EK}}$ . This creates an instance of Groth16 that is specific to the function  $f_R$ .<sup>4</sup> The  $\text{CRS}_{\text{EK}}$  can be used by anyone to create a proof  $\pi \stackrel{\$}{\leftarrow} \text{Prove}(\text{CRS}_{\text{EK}}, x, w)$  for  $f_R(x, w) = 1$ . One can use  $\text{Verify}(\text{CRS}_{\text{VK}}, x, \pi)$  to verify the proof, which requires only the smaller  $\text{CRS}_{\text{VK}}$ . Groth16 SNARKs are based on pairing groups of elliptic curves; a proof consists of 3 group elements.

<sup>3</sup> All of our benchmarks were obtained on an ESPRIMO Q957 (64-bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM).

<sup>4</sup> Some other SNARK constructions, such as [17] have a universal setup ceremony, i.e., the CRS only needs to be generated once and can then be updated for different indicator functions. This comes at the cost of increasing proof size and proving times.

We use the common curve BN254, which is defined over a base field of size  $\sim 2^{254}$  and provides  $\sim 100$  bits of security. Concretely, and following [25, 28], we use the libsnark implementation [34] of Groth16 for obtaining our benchmarks. Other implementations [4, 19] support curves for higher security levels, such as BLS12-381 or BLS24-317 for 128 – 160 bits of security.

Groth16 uses the language of quadratic arithmetic programs (QAPs) to specify the indicator function  $f_R$  and hence the underlying relation  $R$ . Typically, in order to obtain a QAP,  $f_R(x, w)$  is first expressed as an arithmetic circuit where each input/output/internal wire is represented either by a variable or a constant. The public input  $x$  is a list of values assigned to some wire variables (not necessarily only input wires). A valid witness  $w$  then consists of values assigned to all remaining wire variables such that all of these values, together with constants, describe a correct computation of the circuit.<sup>5</sup> This circuit is then converted to a set of so-called constraints that can in turn be compiled into a QAP instance, which we will not discuss further in this paper. A constraint over  $n$  variables  $a_1, \dots, a_n$  is an equation  $\sum_{i=1}^n a_i u_i \cdot \sum_{i=1}^n a_i v_i = \sum_{i=1}^n a_i w_i$ , where  $u_i, v_i$  and  $w_i$  are constants defining the constraint. For describing instantiations of concrete indicator functions  $f_R$  one can thus use both arithmetic circuits and constraints mostly interchangeably. We will usually describe an instantiation as a circuit yielding a certain number of constraints.

The time required to create a proof and the size of  $\text{CRS}_{\text{EK}}$  of a Groth16 SNARK instance depend linearly on the number of inputs and the number of constraints, i.e., the size of the circuit. Concretely,  $\text{CRS}_{\text{EK}}$  consists of  $3\nu + \mu + 6$  group elements, where  $\nu$  denotes the number of constraints and  $\mu$  denotes the number of inputs. As the number of inputs only has a minor effect on these benchmarks, typically only the number of constraints is considered. To get an idea, here are some figures using the libsnark instantiation over BN254 for a standard PC (cf. Footnote 3): For 100,000, 500,000, and 1,250,000 constraints, the size of the  $\text{CRS}_{\text{EK}}$  is about 162 MB, 810 MB, and 2 GB, respectively. Note that these  $\text{CRS}_{\text{EK}}$  sizes are uncompressed sizes and can usually be reduced by a factor of at least 2 via standard compression methods. Proofs can be computed in about 4.46, 22.3, and 55.75 seconds, respectively. As mentioned above, proof size and verification time are small and independent of the circuit while  $\text{CRS}_{\text{VK}}$  is a small subset of  $\text{CRS}_{\text{EK}}$  which only contains  $\ell + 4$  group elements, where  $\ell$  is the number of wires assigned to the public input (e.g., for 3,000 such wires - far more than we will need -  $\text{CRS}_{\text{VK}}$  is smaller than 500 KB). In this paper we therefore mainly focus on determining and optimizing prover runtime and size of  $\text{CRS}_{\text{EK}}$ .

**CRS Generation and Soundness.** We note that soundness of our ZKPs breaks down if the CRSs are not generated honestly. One can mitigate this issue by computing the CRSs in a distributed fashion before an election; see, e.g., [1, 5, 7]. We note that it is of course desirable to minimize trust assumptions for verifiability. However, in practice one often still has some trust assumptions,

<sup>5</sup> Usually, a valid  $w$  is described only in terms of input wire variables as this already fully defines the remaining witness values for internal and output wire variables.

e.g., trusted bulletin boards, authentication/registration servers, or a trusted PKI. Alternatively, there are other GPZKPs, such as [3, 8], which do not require a trusted CRS generation and are in principle compatible with our constructions, as they use a similar underlying language as Groth16, while being less efficient in terms of computation and proof size.

### 3 Proving Ballot Validity Using Groth16

To construct ballot validity proofs using Groth16, we follow the approach from Kryvos [25] for PVC-based encrypted ballots. In this section, we give a complete overview of their approach, explain how the same techniques can be used for EEG-based ballots, and provide the first benchmarks for several subcomponents. Our benchmarks for complete ballot validity proofs are then given in Sect. 4.

Recall that voters choose their plain ballot  $\mathbf{b}$  as a length- $N$ -vector from some choice space  $C$  and then use an (additively) homomorphic encryption or commitment scheme  $\text{Enc}(\cdot)$  to obtain an encrypted ballot  $c \leftarrow \text{Enc}(\mathbf{b})$ . To show ballot validity via a GPZKP such as Groth16, a voter uses the following indicator function  $f_R(x, w)$ : the public statement  $x$  contains the encrypted ballot  $c$ . The witness  $w$  contains a plain ballot  $\mathbf{b}$  and randomness  $\mathbf{r}_w$  such that  $f_R(x, w) = 1$  iff  $\text{Enc}(\mathbf{b}, \mathbf{r}_w) = c$  and  $\mathbf{b} \in C$ .

We construct a corresponding arithmetic circuit  $\mathcal{C}$  for ballot validity from two separate sub-circuits as shown in Fig. 1. The *encryption* subcircuit  $\mathcal{C}_{\text{Enc}}$  re-computes the encrypted ballot from the plain ballot  $\mathbf{b}$  and randomness  $\mathbf{r}_w$  contained in the witness  $w$  and from the public encryption key contained in a public input  $\text{aux}_{\text{Enc}}$ . The public encrypted ballot  $c$  is assigned to the output wires of  $\mathcal{C}_{\text{Enc}}$ , which implies that  $\text{Enc}(\mathbf{b}, \mathbf{r}_w) = c$  holds in a valid proof for this circuit. The *voting* subcircuit  $\mathcal{C}_{\text{Voting}}$  takes as input the plain ballot  $\mathbf{b}$  from the input witness  $w$  and then outputs a bit indicating whether  $\mathbf{b} \in C$ . The constant 1 is assigned to the output wire of  $\mathcal{C}_{\text{Voting}}$ , which implies that  $\mathbf{b} \in C$  holds for valid proofs. Both subcircuits might take additional auxiliary public and witness values as input which can be used to improve efficiency or to generalize circuits.

This modular design of  $\mathcal{C}$  simplifies circuit design and optimization while enabling the re-use of components shared by circuits for different voting methods, most notably  $\mathcal{C}_{\text{Enc}}$ , which does not depend on  $C$  (except for the length of the vote vector). In the following subsections, we will explain how we construct both subcircuits while keeping the number of constraints small. We note that the overall number of constraints and, hence, the overall performance of  $\mathcal{C}$  is essentially the sum of  $\mathcal{C}_{\text{Enc}}$  and  $\mathcal{C}_{\text{Voting}}$ . To compare their relative impact we therefore also provide benchmarks for all subcomponents.

#### 3.1 Constructing and Optimizing $\mathcal{C}_{\text{Enc}}$

Due to the complexity of encryption/commitment schemes, designing an efficient  $\mathcal{C}_{\text{Enc}}$  with a small and hence practical number of constraints is a highly non-trivial task that makes or breaks the practicality of the overall ballot validity proof. The

authors of Kryvos [25] spent much effort on designing a highly optimized  $\mathfrak{C}_{\text{Enc}}^{\text{PVC}}$  for PVCs which we will first recall and then show how it can be transformed into a circuit  $\mathfrak{C}_{\text{Enc}}^{\text{EEG}}$  for EEG. This transformation is mostly straightforward on a technical level as both primitives use the same operations. The main question we investigate rather is the resulting performance and practicality, which is unclear for  $\mathfrak{C}_{\text{Enc}}^{\text{EEG}}$  due to the reasons detailed at the end of the next paragraph.

**Existing Building Blocks for PVCs from [25].** Let  $\mathcal{G}$  be a (multiplicative) group of prime order  $q$  and let  $h, g_1, \dots, g_N$  be generators of  $\mathcal{G}$  such that no relation between these generators is known. A PVC on a plaintext vector  $\mathbf{v} = (v_1, \dots, v_N) \in \mathbb{Z}_q^N$  is defined as  $c =, (\mathbf{v}, r) = g_1^{v_1} \cdot \dots \cdot g_N^{v_N} \cdot h^r \in \mathcal{G}$  for (uniform) randomness  $r \xleftarrow{\$} \mathbb{Z}_q$ . The case  $N = 1$  gives a standard Pedersen commitment.

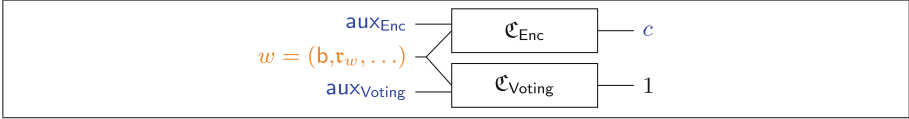
A major factor for the size and hence performance of  $\mathfrak{C}_{\text{Enc}}^{\text{PVC}}$  is exponentiation. Building on results from [28], Kryvos uses an instantiation of the common Montgomery elliptic curve Curve25591 over the scalar field of BN254 (the curve used for Groth16 by libsark [34], see Sect. 2), which allows for an efficient implementation of exponentiation via the Montgomery ladder algorithm.<sup>6</sup> More precisely, as described in [28], we set  $\mathcal{G}$  to be the large prime-order subgroup of this curve, which has size  $q \approx 2^{251}$ . A group element is a curve point that can be represented in affine or equivalently projective coordinates consisting of two resp. three coordinates in  $\mathbb{Z}_q$ . In  $\mathfrak{C}_{\text{Enc}}^{\text{PVC}}$ , a point is represented by one wire per coordinate. For affine coordinates, a third wire is used to indicate whether the given point is the special point at infinity. The number of constraints needed for implementing the Montgomery ladder algorithm then depends on the (maximal) size of the exponent. According to [25], an exponentiation with an arbitrary 255 bit randomness  $r$  requires 5,084 constraints. However, valid votes  $\mathbf{v}$  usually have much smaller entries  $v_i$ , typically just a few bits (depending on the choice space). Kryvos bounds the size of a (valid)  $v_i$  by 32 bits, which covers all interesting choice spaces and requires only 624 constraints for one exponentiation.

Based on this choice of  $\mathcal{G}$ , Kryvos designed and reported constraint numbers for the following subcircuits: (i) The aforementioned circuit for computing an exponentiation  $g^m$  of an elliptic curve point  $g$  with  $m \leq q$  using Montgomery’s ladder. This only gives the (projective)  $X$ - and  $Z$ -coordinate of  $g^m$ . (ii) A circuit for computing the (projective)  $Y$ -coordinate from output of the Montgomery ladder and the (projective)  $Y$ -coordinate of  $g$  following Okea and Sakurai [32] (39 constraints). (iii) A circuit for converting projective to affine coordinates (15 constraints). (iv) A circuit for multiplying two points given in affine coordinates (86 constraints). These subcircuits are then combined to obtain  $\mathfrak{C}_{\text{Enc}}^{\text{PVC}}$ .

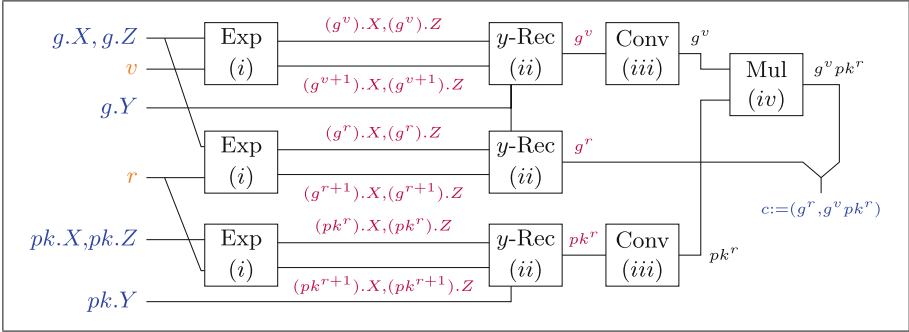
Observe that the exponentiation with large randomness is by far the most expensive step. This is why this approach scales particularly well for PVCs: For committing to a vector of size  $N$ , only a single expensive exponentiation ( $h^r$ ) is needed ( $N+1$  exponentiations overall). In contrast, EEG requires more exponentiations ( $3N$ ) for encrypting a vector of size  $N$  and  $2N$  of those exponentiations are for the large randomness. So this raises the question whether we can obtain reasonably efficient ballot validity SNARKS for EEG.

<sup>6</sup> We stick with multiplicative notion of the group law also for elliptic curve groups.





**Fig. 1.** The arithmetic circuit  $\mathcal{C}$  for proving ballot validity. Secret/witness values are shown in orange, public values are blue, and constants are black. (Color figure online)

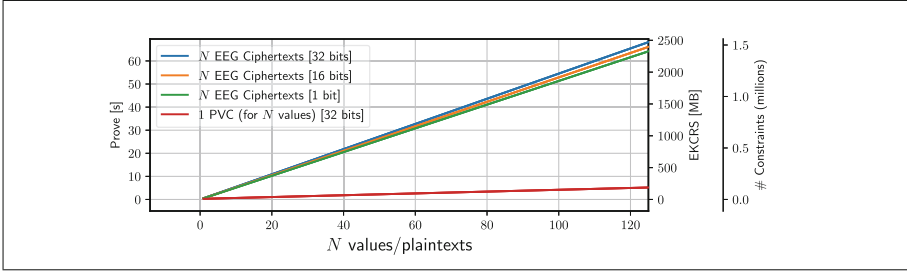


**Fig. 2.** Circuit  $\mathcal{C}_{\text{Enc}}$  for computing an EEG ciphertext  $c$  from plaintext  $v$  with randomness  $r$ . The secret witness (marked in orange) is  $w := (v, r)$ . The public statements (marked in blue) are the ciphertext  $c = (g^r, g^v pk^r)$  and  $\text{aux}_{\text{Enc}}$ , which contains the public key  $pk$  and the generator  $g$ . Where important, we show wires with individual coordinates, e.g.,  $g.X$  denotes the projective  $X$ -coordinate of  $g$ . We also use purple/black color for projective/affine coordinates. When no individual coordinate but just a point is given, e.g.,  $g^v$ , then this represents the three wires for that point’s coordinates. The numbers (i) – (iv) refer to the sub-circuits from Sect. 3.1. (Color figure online)

**Proving Plaintext Knowledge for a Vector of EEG Ciphertexts.** Again, let  $\mathcal{G}$  be a (multiplicative) group of prime order  $q$  and generator  $g$ . An EEG ciphertext for a plaintext  $v \in \mathbb{Z}_q$  and a given public key  $pk \in \mathcal{G}$  is obtained by sampling a randomness  $r \xleftarrow{\$} \mathbb{Z}_q$  and returning  $c = (c_0, c_1) = (g^r, g^v \cdot pk^r)$ .

We constructed a circuit  $\mathcal{C}_{\text{Enc}}^{\text{EEG}}$  for computing  $N$  such ciphertexts from  $N$  plaintexts  $v_i$ ,  $N$  randomnesses  $r_i$ , and a public key  $pk$  from the subcircuits established in Kryvos. We depict the resulting circuit in Fig. 2 for the case  $N = 1$ . For  $N > 1$ , this circuit is copied  $N$  times with separate input and output wires, except for the input wires corresponding to  $pk$  and  $g$  which are shared by all copies.

**Benchmarks and Comparison.** After implementing our new circuit  $\mathcal{C}_{\text{Enc}}^{\text{EEG}}$  for EEG ciphertexts, we have benchmarked the performance of Groth16 for this circuit and various sizes  $N$  of the plaintext vector, as well as various upper bounds on the bit length of individual plaintexts. We have also benchmarked the existing implementation of  $\mathcal{C}_{\text{Enc}}^{\text{PVC}}$  for PVCs on the same machine (see Footnote 3 on Page 111) to obtain a fair comparison, with all results shown in Fig. 3.



**Fig. 3.** Prover runtime,  $\text{CRS}_{\text{EK}}$  size, and constraints for  $\mathcal{C}_{\text{Enc}}^{\text{EEG}}$  and  $\mathcal{C}_{\text{Enc}}^{\text{PVC}}$

As expected, creating a SNARK proof for validating a vector of EEG ciphertexts instead of a PVC is much less efficient for large vector lengths  $N$ . However, and perhaps unexpected, even for a vector consisting of 50 EEG ciphertexts a voter can still compute a proof in less than 30 seconds using a  $\text{CRS}_{\text{EK}}$  of about 1 GB, which is already good enough to be viable in a wide range of settings and election types. For a more detailed discussion of practicality in various situations, see Sect. 4.

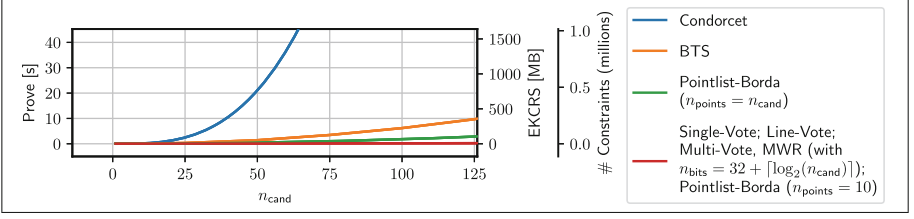
Interestingly, our prover runtimes for  $\mathcal{C}_{\text{Enc}}^{\text{PVC}}$  significantly outperformed the ones that we obtained in [25] by a factor of 2 to 3 on a comparable machine. After investigating the issue, it turns out that in [25], we accidentally used a custom version of the libsnark library that performs additional computations for debugging purposes and hence is much slower.

### 3.2 Constructing and Optimizing $\mathcal{C}_{\text{Voting}}$

Since the subcircuit  $\mathcal{C}_{\text{Voting}}$  checks that a (plain) ballot belongs to a given choice space, its design depends on the voting method/choice space. Here, we describe and benchmark circuits for the following common voting methods/choice spaces: Single-Vote, Multi-Vote, Borda Count, and Condorcet. To investigate the potential and limits of GPZKPs for developing and supporting new voting methods and systems, we further construct and benchmark circuits for two additional (somewhat artificial) complex choice spaces - both variants of Multi-Vote, which we call *Line-Vote* and *Multi-Vote with Rules*. We provide the benchmarks for  $\mathcal{C}_{\text{Voting}}$  for all of our choice spaces in Fig. 4.

Since  $\mathcal{C}_{\text{Voting}}$  is independent of the method used for encrypting ballots - thanks to the modularity of  $\mathcal{C}$  - we can reuse the existing sub-circuits for Single-Vote, Multi-Vote, Borda Count, and Condorcet from [25] with some minor optimizations and extensions. We briefly recall their designs for completeness and provide some additional details, such as constraint numbers. We also provide the first benchmarks for these subcircuits, which were not benchmarked separately in [25].

**Single-Vote.** Recall that in a single-vote election, a voter can give only one vote for their preferred candidate, with the corresponding choice space  $C_{\text{single}}$



**Fig. 4.** Prover runtime,  $\text{CRS}_{\text{EK}}$  size, and constraints of  $\mathcal{C}_{\text{Voting}}$  for several voting methods. For Pointlist-Borda, we use  $\mathcal{L} = \{1, \dots, n_{\text{points}}\}$ , for Multi-Vote and MWR, we use  $t = 2^{32} - 1$  and  $n_{\text{max}} = n_{\text{cand}} \cdot t$ .

defined in Sect. 1. Checking that  $b \in C_{\text{single}}$  entails two substeps: (i) Checking that each ballot entry is a bit, which requires one constraint per candidate, and (ii) checking that the sum of all ballot entries equals 1, which requires one constraint. To allow abstention by casting a ballot without a vote, one can instead check that the sum is a bit, which also requires one constraint.

For  $n_{\text{cand}}$  candidates,  $\mathcal{C}_{\text{Voting}}^{\text{single}}$  thus consists of  $n_{\text{cand}} + 1$  constraints. This yields a very small  $\text{CRS}_{\text{EK}}$  of less than 1 MB and proof times of less than 0.05 seconds for any realistic number of candidates (see Fig. 4).

**Multi-vote.** Multi-vote generalizes single-vote by letting voters allocate up to  $n_{\text{max}}$  votes among  $n_{\text{cand}}$  candidates, with a maximum of  $t$  votes assigned to any candidate. Analogous to  $C_{\text{single}}$ , we define the following choice space:

$$C_{\text{multi}}(n_{\text{max}}, t) := \left\{ (v_1, \dots, v_{n_{\text{cand}}}) \mid \forall i : v_i \in \{0, 1, \dots, t\} \wedge 0 \leq \sum_{i=1}^{n_{\text{cand}}} v_i \leq n_{\text{max}} \right\}.$$

The circuit  $\mathcal{C}_{\text{Voting}}^{\text{multi}}$  for  $C_{\text{multi}}$  then checks that each  $v_i$  is in the allowed range (between 0 and  $t$ ) and that the sum of all  $v_i$  is in the correct range (between 0 and  $n_{\text{max}}$ ). Such range checks require converting the respective value into individual bits. Therefore, the number of constraints depends on the maximal possible bit size  $n_{\text{bits}}$  of  $\sum_{i=1}^{n_{\text{cand}}} v_i$  which is, in turn, determined by the bit sizes of  $t$  and  $n_{\text{max}}$ . The complete circuit requires about  $(n_{\text{bits}} + 1) \cdot (n_{\text{cand}} + 1)$  constraints (the exact number depends on  $t$  and  $n_{\text{max}}$ ), which - even for unrealistically high values of  $n_{\text{bits}}$  such as  $n_{\text{bits}} = 41$  - is still very small for any realistic number of candidates. Hence, performance of  $\mathcal{C}_{\text{Voting}}^{\text{multi}}$  is essentially the same as for  $\mathcal{C}_{\text{Voting}}^{\text{single}}$  (see Fig. 4).

### Supporting New Choice Spaces: Line-Vote and Multi-Vote with Rules.

We consider two modifications of multi-vote that are somewhat artificial but represent cases where one might want to use GPZKPs: they are novel choice spaces, so no ballot validity ZKPs exist, and as they are obtained by adding non-trivial interdependencies between the votes for individual candidates, it is hard to construct new specialized ZKPs.

*Line-Vote:* In Line-Vote, voters are given  $n_{\text{cand}}$  many (ordered) options to vote YES or NO. Voters can vote YES for any number of those options subject to the

restriction that all YES-votes must form a continuous line, i.e., if two options receive a YES-vote, then all options in-between must receive a YES-vote as well. The choice space can be formalized as follows:

$$C_{\text{line}} := \{(v_1, \dots, v_{n_{\text{cand}}}) \mid v_i \in \{0, 1\} \wedge (i < j \wedge v_i, v_j = 1 \Rightarrow \forall i < k < j : v_k = 1)\}.$$

A corresponding circuit  $\mathfrak{C}_{\text{Voting}}^{\text{line}}$  can be built easily analogous to  $\mathfrak{C}_{\text{Voting}}^{\text{multi}}$ :  $\mathfrak{C}_{\text{Voting}}^{\text{line}}$  uses an additional “helper” wire which is first set to  $v_1$  and is then incremented for all non-zero  $v_i$  that occur directly after a zero entry  $v_{i-1}$ . A ballot is then valid iff all  $v_i$  and the helper wire are bits. This circuit consists of  $2n_{\text{cand}}$  constraints. *Multi-Vote with Rules (MWR)*: In MWR, we consider multi-vote ballots whose entries are subject to additional arithmetic rule(s). One can add arbitrary (numbers of) rules. As a concrete example, we consider a rule where the product of the second and the third ballot entry equals the first one:

$$C_{\text{MWR}}(n_{\text{max}}, t) := \{b = (v_1, \dots, v_{n_{\text{cand}}}) \in C_{\text{multi}}(n_{\text{max}}, t) \mid v_1 = v_2 \cdot v_3\}.$$

The corresponding circuit  $\mathfrak{C}_{\text{Voting}}^{\text{MWR}}$  is again easy to construct: use  $\mathfrak{C}_{\text{Voting}}^{\text{multi}}$  as a basis and add additional constraints for each rule. In the above example, just 2 additional constraints are needed.

Altogether, both examples confirm that it is indeed simple to support new choice spaces via GPZKPs and that, depending on the additional conditions imposed on the  $v_i$ , this might not even come at a noticeable cost (see Fig. 4).

**Pointlist-Borda and Borda Tournament Style (BTS).** Borda is a ranked election method where voters rank the candidates according to their preference and, based on this ranking, points are assigned to each candidate. Variants of Borda are used, e.g., for parliamentary elections in Nauru [33] and the Eurovision Song Contest (ESC) [16]. As suggested in [25], such variants used in practice can be captured as instances of what they call *Pointlist-Borda*. A Pointlist-Borda instance is defined via a fixed point list  $\mathcal{L}$  that contains  $n_{\text{points}}$  many distinct positive numbers. Voters then construct their ballots by assigning each number in  $\mathcal{L}$  to one candidate and, if  $n_{\text{points}} < n_{\text{cand}}$ , 0 points to all remaining candidates. Observe that this represents a ranking where the highest-ranked candidate receives the most points and so on with  $n_{\text{cand}} - n_{\text{points}}$  candidates tied for the last place. Formally, the choice space is as follows:

$$C_{\text{BordaPointList}}(\mathcal{L}) := \left\{ (v_1, \dots, v_{n_{\text{cand}}}) \mid (\forall p \in \mathcal{L} \exists i : v_i = p) \wedge |\{i \in [1, n_{\text{cand}}] \mid v_i = 0\}| = n_{\text{cand}} - n_{\text{points}} \right\}.$$

The size of  $\mathfrak{C}_{\text{Voting}}^{\text{BordaPointList}}$  depends on  $n_{\text{points}} = |\mathcal{L}|$  but is not affected by the concrete values in  $\mathcal{L}$  (hence, we simply take  $\mathcal{L} = [1, n_{\text{points}}]$  for benchmarking). For small constants, such as  $n_{\text{points}} = 10$ , the size of  $\mathfrak{C}_{\text{Voting}}^{\text{BordaPointList}}$  scales linearly in  $n_{\text{cand}}$ , similar to single-/multi-vote. The worst case is  $n_{\text{points}} = n_{\text{cand}}$ , which scales quadratically in  $n_{\text{cand}}$  but remains practical. For example, in an extreme case of  $n_{\text{points}} = n_{\text{cand}} = 100$ , computing a proof still only requires less than 2 seconds and a  $\text{CRS}_{\text{EK}}$  of less than 100 MB (see Fig. 4 for both cases).

There are many ways to design generalized (academic) Borda variants that allow for ties between candidates at arbitrary positions. For example, [25] considers an even more complex Borda variant they call *Borda tournament style (BTS)* which we include in our benchmarks using their circuit  $\mathfrak{C}_{\text{Voting}}^{\text{BTS}}$  (see Fig. 4). Due to space limitations, we refer to [25] for details of BTS.

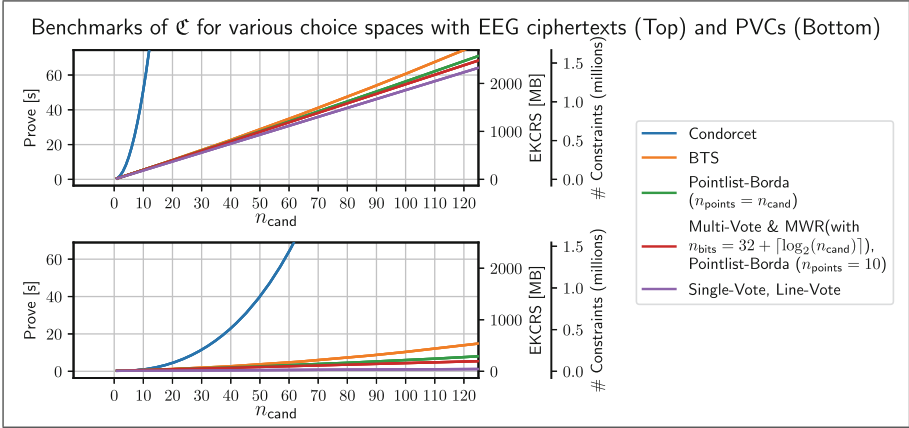
**Condorcet Methods.** In Condorcet methods, which are, e.g., used for internal elections of the Debian project [14], a voter submits a ranking of candidates. Condorcet methods differ in how they determine the winner but, if such a candidate exists, they will return the candidate who wins against all other candidates in a direct comparison. To make rankings compatible with aggregation of ballots, they are typically represented as comparison matrices [12, 23, 25].

Specifically, given a ranking  $r = (r_1, \dots, r_{n_{\text{cand}}}) \in \mathbb{N}^{n_{\text{cand}}}$  of candidates (where  $r_i > r_j$  means that candidate  $i$  is ranked worse than candidate  $j$ ), a voter constructs her ballot as an  $n_{\text{cand}} \times n_{\text{cand}}$  matrix  $A$  with 1 at position  $(i, j)$  if candidate  $i$  is ranked better than candidate  $j$  and 0 otherwise. Note that hence  $n_{\text{cand}}^2$  many values are used for a ballot, unlike all aforementioned voting methods that used one value per candidate. Also note that, if candidates are tied, then this is represented by  $A_{ij}$  and  $A_{ji}$  both being 0, i.e., a ballot is a *positive preference matrix* as defined in [12]. The choice space then is:

$$C_{\text{Condorcet}} = \left\{ A \in \{0, 1\}^{n_{\text{cand}} \times n_{\text{cand}}} \mid \exists (r_1, \dots, r_{n_{\text{cand}}}) \in \mathbb{N}^{n_{\text{cand}}} \text{ s.t. } \forall i, j \in [1, n_{\text{cand}}] : \right. \\ \left. \begin{aligned} r_i > r_j &\Rightarrow A_{ij} = 0, A_{ji} = 1 \wedge \\ r_i = r_j &\Rightarrow A_{ij} = A_{ji} = 0 \end{aligned} \right\}$$

The circuit  $\mathfrak{C}_{\text{Voting}}^{\text{Condorcet}}$  extends the one proposed in [25], which did not support ties. It first checks that all matrix entries are bits and that for  $i \neq j$  also  $A_{ij} + A_{ji}$  is a bit.<sup>7</sup> It remains to check transitivity (i.e., that, for any triple  $(i, j, k)$  of distinct candidates, it holds that  $r_i \leq r_j$  and  $r_j \leq r_k$  imply  $r_i \leq r_k$ , with  $r_i = r_k$  iff  $r_i = r_j$  and  $r_j = r_k$ ). Checking both cases, i.e.,  $\leq$  and  $=$ , turns out to be easier if ties through 1-entries instead of 0-entries. For this,  $\mathfrak{C}_{\text{Voting}}$  computes a “check matrix”  $B$  with  $B_{ij} := 1 - A_{ji}$ , which does not require any new constraints. Note that  $B$  equals  $A$  everywhere except that 1-entries replace the 0-entries that represent ties in  $A$ . Then, the circuit checks whether  $B_{ij} \cdot B_{jk} \cdot (1 - B_{ik}) = 0$ , which is true iff  $A$  is transitive (observe that this check indeed covers both the  $\leq$  and the  $=$  case). The resulting circuit scales cubically in the number of candidates, where, e.g., 25 candidates require a  $\text{CRS}_{\text{EK}}$  of about 90 MB and a proof time of about 2.5 seconds (see Fig. 4).

<sup>7</sup> One can instead check that  $A_{ij} + A_{ji} = 1$  to prevent ties as proposed in [25]. This yields the same number of constraints.



**Fig. 5.** Comparison of full ballot validity proofs. Condorcet, Single-, and Line-Vote use  $\mathcal{C}_{\text{Enc}}$  with 1-bit plaintexts; all other choice spaces use 32-bit plaintexts.

## 4 Overall Benchmarks for Proving Ballot Validity

Following the outline given in Sect. 3, we can now combine the encryption subcircuit  $\mathcal{C}_{\text{Enc}}$  with a suitable plaintext bit size from Sect. 3.1 and a voting subcircuit  $\mathcal{C}_{\text{Voting}}$  from Sect. 3.2 to obtain complete circuits  $\mathcal{C}$  for proving ballot validity. Our benchmarks of prover runtime,  $\text{CRS}_{\text{EK}}$  size, and constraints for these circuits using EEG encryption and depending on the number of candidates  $n_{\text{cand}}$  are given in the top half of Fig. 5. For comparison, in the bottom half of Fig. 5 we provide our benchmarks for ballots computed as PVCs using the constructions of [25]. As mentioned in Sect. 2, the proof size is less than 1 KB, and verification requires only about 7 ms as both are mostly independent of the circuit. Since the  $\text{CRS}_{\text{VK}}$  is a subset of  $\text{CRS}_{\text{EK}}$  we do not provide separate benchmarks, but its size is always in the order of  $\sim 20$  KB and hence negligible.

The performance of the Groth16 proof for the combined circuit  $\mathcal{C}$  is essentially the sum of the subcircuits  $\mathcal{C}_{\text{Enc}}$  and  $\mathcal{C}_{\text{Voting}}$  and thus dominated by the much slower  $\mathcal{C}_{\text{Enc}}$ . Note that the performance of  $\mathcal{C}_{\text{Enc}}$  in Fig. 3 was given depending on the number  $N$  of plaintexts, while for the combined circuit  $\mathcal{C}$  we consider performance depending on number  $n_{\text{cand}}$  of candidates. All but one choice space use one plaintext per candidate, i.e.,  $N = n_{\text{cand}}$ , so the benchmarks given in Fig. 5 mostly retain the linear behavior of  $\mathcal{C}_{\text{Enc}}$ , potentially plus some small non-linear overhead caused by  $\mathcal{C}_{\text{Voting}}$ . The exception are Condorcet ballots, where  $N = n_{\text{cand}}^2$ . This causes visibly quadratic behavior in the combined circuit due to  $\mathcal{C}_{\text{Enc}}$  (plus some much smaller cubic overhead due to  $\mathcal{C}_{\text{Voting}}^{\text{Condorcet}}$ ).

To summarize our benchmarks, for most election types with EEG, Groth16 ballot validity proofs can be computed by voters within a reasonable time on standard PCs, even for large numbers of candidates. Since runtime is dominated by  $\mathcal{C}_{\text{Enc}}$ , it stays mostly the same even for new ballot formats with potentially

very complex validity rules, as shown by Line-Vote, MWR, and BTS. The only outlier is Condorcet, for which computing a proof quickly becomes impractical due to the quadratic number of ciphertexts. We note, however, that real-world Condorcet elections, such as [14], rarely have more than 10 candidates. For such cases, a proof of ballot validity can still be computed in less than a minute. As for the size of  $\text{CRS}_{\text{EK}}$ , it is non-negligible in all cases but still within ranges that can reasonably be downloaded once as part of the election software. Also, recall that the presented  $\text{CRS}_{\text{EK}}$  sizes are uncompressed sizes. We also note that the same CRS can then be re-used for multiple elections.

In conclusion, our results establish that Groth16 and, hence, GPZKPs are a viable option for showing ballot validity in EEG-based voting systems. We have further shown the potential of GPZKPs for supporting new voting methods with novel complex ballot formats. While specialized ZKPs, where available, can still be preferable to GPZKPs, e.g., due to better efficiency, our results show that GPZKPs can be a viable and, importantly, quite generic and uniform option. A detailed performance comparison between GPZKPs and specialized ZKPs for various ballot formats and group choices would be an interesting future work.

**Acknowledgements.** This research was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 411720488.

## References

1. Abdolmaleki, B., et al.: UC-Secure CRS Generation for SNARKs. In: AFRICACRYPT 2019, Proceedings. LNCS, vol. 11627, pp. 99–117. Springer (2019)
2. Adida, B., et al.: Electing a university president using open-audit voting: analysis of real-world use of helios. In: USENIX/ACCURATE Electronic Voting Technology (EVT 2009) (2009)
3. Ames, S., et al.: Ligerio: lightweight sublinear arguments without a trusted setup. In: ACM CCS 2017, pp. 2087–2104 (2017)
4. Bellés-Muñoz, M., et al.: Circom: a circuit description language for building zero-knowledge applications. *IEEE Trans. Dependable Secur. Comput.* **20**(6), 4733–4751 (2023)
5. Ben-Sasson, E., et al.: Secure sampling of public parameters for succinct zero knowledge proofs. In: IEEE SP 2015, pp. 287–304. IEEE Computer Society (2015)
6. Ben-Sasson, E., et al.: Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive* **2018**, 46 (2018)
7. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-SNARK parameters in the random beacon model. *IACR Cryptol. ePrint Arch.* **2017**, 1050 (2017)
8. Bünz, B., et al.: Bulletproofs: short proofs for confidential transactions and more. In: SP 2018, pp. 315–334 (2018)
9. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO '92. LNCS, vol. 740, pp. 89–105. Springer (1992)
10. Chiesa, A., Ojha, D., Spooner, N.: Fractal: post-quantum and transparent recursive proofs from holography. In: EUROCRYPT 2020, pp. 769–793 (2020)

11. Cortier, V., Gaudry, P., Glondou, S.: Belenios: a simple private and verifiable electronic voting system. In: Guttman, J.D., Landwehr, C.E., Meseguer, J., Pavlovic, D. (eds.) *Foundations of Security, Protocols, and Equational Reasoning*. LNCS, vol. 11565, pp. 214–238. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-19052-1\\_14](https://doi.org/10.1007/978-3-030-19052-1_14)
12. Cortier, V., Gaudry, P., Yang, Q.: A toolbox for verifiable tally-hiding e-voting systems. In: *ESORICS 2022*. LNCS, vol. 13555, pp. 631–652. Springer (2022)
13. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: *CRYPTO 1994*, pp.174–187. Springer (1994)
14. Debian Project: Debian Voting Information (2024). <https://www.debian.org/vote/>
15. Devillez, H., Pereira, O., Peters, T.: How to verifiably encrypt many bits for an election? In: *ESORICS 2022*. LNCS, vol. 13555, pp. 653–671. Springer (2022)
16. European Broadcasting Union: Eurovision Song Contest - How it works (2024). <https://eurovision.tv/about/how-it-works>
17. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.* **2019**, 953 (2019)
18. Gaudry, P.: Some ZK security proofs for belenios (2017)
19. Gautam Botrel and Others: Consensus/gnark: v0.10.0 (2024). <https://doi.org/10.5281/zenodo.11034183>
20. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: faster Zero-Knowledge for boolean circuits. In: *USENIX Security Symposium 2016*, pp. 1069–1083. USENIX Association (2016)
21. Groth, J.: Non-interactive zero-knowledge arguments for voting. In: *ACNS 2005*. LNCS, vol. 3531, pp. 467–482 (2005)
22. Groth, J.: On the size of pairing-based non-interactive arguments. In: *EUROCRYPT 2016*. LNCS, vol. 9666, pp. 305–326. Springer (2016)
23. Hertel, F., et al.: Extending the tally-hiding ordinos system: implementations for borda, hare-niemeyer, condorcet, and instant-runoff voting. In: *E-Vote-ID 2021*, pp. 269–284. University of Tartu Press (2021)
24. Hopwood, D.E., et al.: Zcash Protocol Specification (2024). <https://zips.z.cash/protocol/protocol.pdf>
25. Huber, N., et al.: Kryvos: publicly tally-hiding verifiable e-voting. In: *CCS 2022*, pp. 1443–1457. ACM (2022)
26. Huber, N., et al.: Implementation of our Circuits (2024). <https://github.com/HicolasNuber/ballotsnarks>
27. Joaquim, R.: How to prove the validity of a complex ballot encryption to the voter and the public. *JISA* **19**(2), 130–142 (2014)
28. Kosba, A., et al.:  $C\mathcal{O}C\mathcal{O}$ : A framework for building composable zero-knowledge proofs. *Cryptology ePrint Archive* (2015)
29. Maller, M., et al.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: *Proceedings of the 2019 ACM CCS*, pp. 2111–2128 (2019)
30. Mestel, D., Müller, J., Reisert, P.: How efficient are replay attacks against vote privacy? A formal quantitative analysis. *J. Comput. Secur.* **31**(5), 421–467 (2023)
31. Morais, E., Koens, T., van Wijk, C., Koren, A.: A survey on zero knowledge range proofs and applications. *SN Appl. Sci.* **1**(8), 1–17 (2019). <https://doi.org/10.1007/s42452-019-0989-z>



32. Okeya, K., Sakurai, K.: Efficient elliptic curve cryptosystems from a scalar multiplication algorithm with recovery of the  $y$ -coordinate on a montgomery-form elliptic curve. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 126–141. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44709-1\\_12](https://doi.org/10.1007/3-540-44709-1_12)
33. Republic of Nauru: Electoral Act No. 15 (2024). [http://ronlaw.gov.nr/nauru\\_lpms/files/acts/d83250a1ebdc56c1701fa7aa245af5b1.pdf](http://ronlaw.gov.nr/nauru_lpms/files/acts/d83250a1ebdc56c1701fa7aa245af5b1.pdf)
34. scipr-lab: libsark (2024). <https://github.com/scipr-lab/libsark>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

