ZK-SNARKs for Ballot Validity: A Feasibility Study

Nicolas Huber ✉, Ralf Küsters, Julian Liedtke, and Daniel Rausch

University of Stuttgart
`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_{cand} candidates, a vote would be a vector of length n_{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_{cand} results in an encrypted ballot[1] consisting of n_{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 [3,21]. 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. Otherwise, a malicious voter could, without being detected, submit an encrypted ballot that contains malformed values for the candidates available in an election, e.g., by assigning more points than allowed to

[1] For simplicity of presentation, we will often only say "encrypted ballot" to refer to both cases, i.e., encryption or commitments.

one candidate while assigning negative points to all other candidates. The aggregated tally computed from such an invalid ballot would contain an incorrect election result, thus breaking verifiability. The standard approach to avoid this issue 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_{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:

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

A voter is supposed to choose her ballot b as a vector from this set, i.e., $\mathsf{b} \in C_{\mathsf{single}}$. The voter then computes an encrypted ballot $\boldsymbol{c}$ from b and submits $\boldsymbol{c}$ alongside a ZKP which shows that $\boldsymbol{c}$ was obtained by encrypting a ballot $\mathsf{b} \in C_{\mathsf{single}}$. 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 $\boldsymbol{c}$. 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 [3] and Belenios [21, 28] support C_{single} with component-wise EEG encryption. That is, $\boldsymbol{c}$ 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 [19, 23], which show that an EEG ciphertext encrypts a value from a specific set S. Concretely, for C_{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 plaintext v_i is from S, and *(ii)* one proof for the homomorphic sum of all n_{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_{max} votes to candidates of their choice (up to a limit t for any individual candidate) [3, 21, 28]. This can be achieved by using a larger set S. However, for larger values of n_{max} (and t) - and hence larger sets S - this quickly becomes too inefficient. In such cases, one can replace disjunctive Chaum-Pedersen proofs with range-proofs [44].

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. For standard Borda elections that do not permit ties within a ballot, i.e., where the ballot is computed from distinct ranks for all candidates, ZKPs for ballot validity based on arguments for the correctness of a shuffle have been proposed [31]. These ZKPs use a choice space where ballots encode a permutation and support encryption via various homomorphic encryption schemes, including EEG. More general versions of Borda allow voters to assign the same rank to multiple candidates. Since such ballots are no longer permutations of the distinct ranks, the aforementioned proofs cannot be used. In cases where ties are only allowed at the last place, there are constructions based on using multiple shuffle proofs [37]. To our knowledge, the only work that has considered ballot validity ZKPs for Borda ballots with ties at arbitrary positions is the Kryvos system [35], which leverages general purpose zero-knowledge proofs for this case (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 [22], 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 [34], they have so far mostly gone unnoticed in the area of e-voting. In [25], techniques based on inner product arguments (which are commonly used for constructing GPZKPs [17]) 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 our Kryvos system [35]. While not the primary focus, as a side result of [35] we have shown that and how the state-of-the-art Groth16 SNARK [32] 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, we 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, we did not further investigate the viability of GPZKPs for ballot validity beyond the uncommon case of PVCs there.

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. A key issue is that while only a single PVC is needed to commit to the entire vote vector, solutions using EEG require multiple ciphertexts, one for each entry of the vote vector. This increases the complexity and, hence, the resource requirements of a corresponding GPZKP instance, possibly to a point where voters cannot compute it at all or might require an unreasonable amount of time (see also Section 3.1 for a more detailed discussion of the underlying issue). 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 [35] 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 [35] 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.[2] This includes not only major existing ones: Single- and Multi-Vote, Borda Count, Condorcet methods, and Majority Judgment. 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.

[2] All of our implementations are available at [36].

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, 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 Exponetial 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

In this section, we briefly introduce GPZKPs and sketch how Groth16 works. For a more detailed and technical description, see Appendices A and B.

A general purpose zero-knowledge proof (GPZKP) system takes as input an arbitrary indicator function $f_R : \{0{,}1\}^* \times \{0{,}1\}^* \to \{0{,}1\}$ 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 [43].

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 [4,13,20,27,30,32,41], SNARKs fit these requirements best. Following [35], we use the highly efficient state-of-the-art Groth16 SNARK [32] that offers constant small proof size of less than 1 kilobyte with (almost) constant[3] verification time of about a few milliseconds on a standard PC[4] - 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 $\mathsf{Setup}(f_R)$ algorithm generates two common reference strings, $\mathsf{CRS}_{\mathsf{EK}}$ (*evaluation key* CRS) and $\mathsf{CRS}_{\mathsf{VK}}$ (*verification key* CRS) that depend on f_R. $\mathsf{CRS}_{\mathsf{VK}}$ is a much smaller substring of $\mathsf{CRS}_{\mathsf{EK}}$. This creates an instance of Groth16 that is specific to the function f_R. [5] The $\mathsf{CRS}_{\mathsf{EK}}$ can be used by anyone to create a proof $\pi \overset{\$}{\leftarrow} \mathsf{Prove}(\mathsf{CRS}_{\mathsf{EK}}, x, w)$ for $f_R(x{,}w) = 1$. One can use $\mathsf{Verify}(\mathsf{CRS}_{\mathsf{VK}}, x, \pi)$ to verify the proof, which requires only the smaller $\mathsf{CRS}_{\mathsf{VK}}$. Groth16 SNARKs are based on pairing groups of elliptic curves; a proof consists of 3 group elements. We use the common curve $\mathsf{BN254}$, which is defined over a base field of size $\sim 2^{254}$ and provides ~ 100 bits of security. Concretely, and following [35,40], we use the libsnark implementation [47] of Groth16 for obtaining our benchmarks. Other implementations [10,15] support curves for higher security levels, such as $\mathsf{BLS12\text{-}381}$ or $\mathsf{BLS24\text{-}317}$ for $128 - 160$ bits of security. See Appendix D for details on the instantiation of Groth16.

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) - we call

[3] Formally, verification time is linear in the size of x. However, in comparison with the constant term, the linear term is extremely small for any realistic size of x. The Groth16 SNARK is hence generally considered to have constant verification time. See also Appendix B.

[4] All of our benchmarks were obtained on an ESPRIMO Q957 (64-bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM).

[5] Some other SNARK constructions, such as [27] 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.

those wires *public 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.[6] This circuit is then converted to a set of so-called constraints that can, in turn, be compiled into a QAP instance. We give a more detailed description of R1CS and QAPs in Appendix B.1. 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 $\mathsf{CRS}_{\mathsf{EK}}$ of a Groth16 SNARK instance depend linearly on the number of variables/wires and the number of constraints/multiplication gates - see Appendix B for details. As the number of variables is usually related to the number of constraints, often only the number of constraints is considered. To get an idea, here are some figures using the libsnark instantiation over $\mathsf{BN254}$ for a standard PC (cf. Footnote 4): For 100,000, 500,000, and 1,250,000 constraints, the size of the $\mathsf{CRS}_{\mathsf{EK}}$ is about 162 MB, 810 MB, and 2 GB, respectively. Note that these $\mathsf{CRS}_{\mathsf{EK}}$ 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 $\mathsf{CRS}_{\mathsf{VK}}$ is a small subset of $\mathsf{CRS}_{\mathsf{EK}}$ which only depends on the number of public wires (e.g., for 5,000 such wires - far more than we will need - $\mathsf{CRS}_{\mathsf{VK}}$ is smaller than 500 KB). In this paper we therefore mainly focus on determining and optimizing prover runtime and size of $\mathsf{CRS}_{\mathsf{EK}}$, both of which are crucial for prover efficiency and hence for determining the viability of Groth16 for showing ballot validity.

CRS Generation and Soundness. We note that both, soundness and the zero-knowledge property of Groth16 SNARKs depend (to some extent) on an honestly generated CRS. We provide more insights and possible mitigations in Appendix C.

3 Proving Ballot Validity Using Groth16

To construct ballot validity proofs using Groth16, we follow the approach from Kryvos [35] for PVC-based encrypted ballots. In this section, we give a complete overview of the 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 Section 4.

Recall that voters choose their plain ballot b as a length-N-vector from some choice space C and then use an (additively) homomorphic encryption or commitment scheme $\mathsf{Enc}(\cdot)$ to obtain an encrypted ballot $c \leftarrow \mathsf{Enc}(\mathsf{b})$. 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 b and randomness $\mathfrak{r}_w$ such that $f_R(x,w) = 1$ iff $\mathsf{Enc}(\mathsf{b},\mathfrak{r}_w) = c$ and $\mathsf{b} \in C$.

We construct a corresponding arithmetic circuit $\mathfrak{C}$ for ballot validity from two separate sub-circuits as shown in Figure 1. The *encryption* subcircuit $\mathfrak{C}_{\mathsf{Enc}}$ re-computes the encrypted ballot from the plain ballot b and randomness $\mathfrak{r}_w$ contained in the witness w and from the public encryption key contained in a public input $\mathsf{aux}_{\mathsf{Enc}}$. The public encrypted ballot c is assigned to the output wires of $\mathfrak{C}_{\mathsf{Enc}}$, which implies that $\mathsf{Enc}(\mathsf{b},\mathfrak{r}_w) = c$ holds in a valid proof for this circuit. The *voting* subcircuit $\mathfrak{C}_{\mathsf{Voting}}$ takes as input the plain ballot b from the input witness w and then outputs a bit indicating whether $\mathsf{b} \in C$. The constant 1 is assigned to the output wire of $\mathfrak{C}_{\mathsf{Voting}}$, which implies that $\mathsf{b} \in C$ 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 $\mathfrak{C}$ simplifies circuit design and optimization while enabling the re-use of components shared by circuits for different voting methods, most notably $\mathfrak{C}_{\mathsf{Enc}}$, 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 $\mathfrak{C}$ is essentially the sum of $\mathfrak{C}_{\mathsf{Enc}}$ and $\mathfrak{C}_{\mathsf{Voting}}$. To compare their relative impact we therefore also provide benchmarks for all subcomponents.

[6] 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.

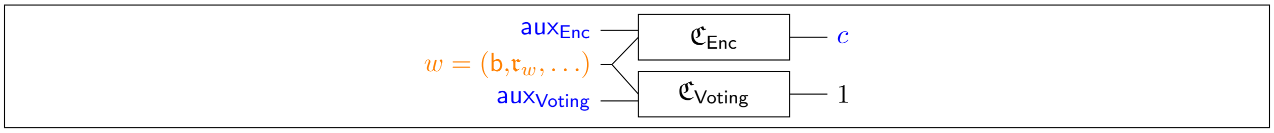

Fig. 1: The arithmetic circuit $\mathfrak{C}$ for proving ballot validity. Secret/witness values are shown in orange, public values are blue, and constants are black.

3.1 Constructing and Optimizing $\mathfrak{C}_{\mathsf{Enc}}$.

Due to the complexity of encryption/commitment schemes, designing an efficient $\mathfrak{C}_{\mathsf{Enc}}$ 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. For designing Kryvos [35] we spent much effort on designing a highly optimized $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$ for PVCs which we will first recall and then show how it can be transformed into a circuit $\mathfrak{C}^{\mathsf{EEG}}_{\mathsf{Enc}}$ 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 $\mathfrak{C}^{\mathsf{EEG}}_{\mathsf{Enc}}$ due to the reasons detailed at the end of the next paragraph.

Existing Building Blocks for PVCs from [35]. Let $\mathcal{G}$ be a (multiplicative) group of prime order q and let $h,g_1,\ldots,g_N$ be generators of $\mathcal{G}$ such that no relation between these generators is known. A PVC on a plaintext vector $\boldsymbol{v} = (v_1,\ldots,v_N) \in \mathbb{Z}_q^N$ is defined as $c = \mathsf{com}(\boldsymbol{v},r) = g_1^{v_1} \cdot \ldots \cdot g_N^{v_N} \cdot h^{\rho} \in \mathcal{G}$ for (uniform) randomness $\rho \xleftarrow{\$} \mathbb{Z}_q$. The case $N = 1$ gives a standard Pedersen commitment.

A major factor for the size and hence performance of $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$ is exponentiation. Building on results from [40], Kryvos uses an instantiation of the common Montgomery elliptic curve Curve25519 over the scalar field $\mathbb{F}_r = \mathbb{Z}_r$ of BN254 (the curve used for Groth16 by libsnark [47], see Section 2 and Appendix D), where r is a 254-bit prime. This allows for an efficient implementation of exponentiation via the Montgomery ladder algorithm.[7] More precisely, as described in [40] and as we recall in Appendix D.2, 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 $\mathbb{F}_r$. In $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$, 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. As reported in [35], an exponentiation with an arbitrary 255 bit randomness r requires 5,084 constraints. However, valid votes $\boldsymbol{v}$ 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. This can, in principle, be improved further by using a smaller bound determined from a specific choice space at hand.

Based on this choice of $\mathcal{G}$, for Kryvos we 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 [45] (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 $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$.

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.

[7] We stick with multiplicative notion of the group law also for elliptic curve groups.

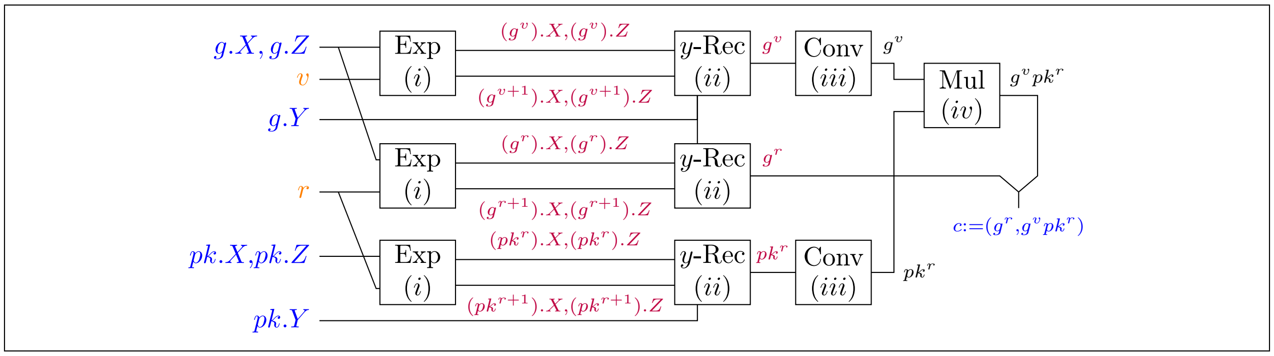

Fig. 2: Circuit $\mathfrak{C}_{\mathsf{Enc}}$ for computing an EEG ciphertext c from plaintext v with randomness r. The secret witness (marked in orange) is $w := (v,r)$. The public statements (marked in blue) are the ciphertext $c = (g^r, g^v pk^r)$ and $\mathsf{aux}_{\mathsf{Enc}}$, which contains 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 Section 3.1.

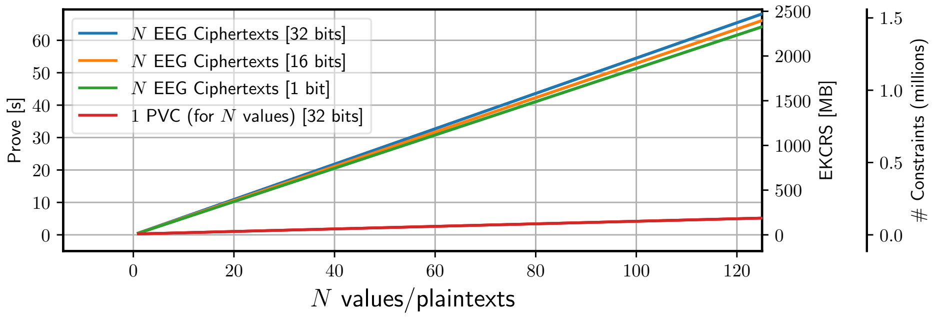

Fig. 3: Prover runtime, $\mathsf{CRS}_{\mathsf{EK}}$ size, and constraints for $\mathfrak{C}^{\mathsf{EEG}}_{\mathsf{Enc}}$ and $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$

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 $\mathfrak{C}^{\mathsf{EEG}}_{\mathsf{Enc}}$ 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 Figure 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. Given the caption of Figure 2, most of the circuit is self-explanatory, perhaps except for the final ciphertext output where $c_0 = g^r$ is returned directly in projective coordinates (without using the Conv subcircuit) since a conversion to affine coordinates, if desired, can also later be computed outside of the circuit/ZKP.

Benchmarks and Comparison. After implementing our new circuit $\mathfrak{C}^{\mathsf{EEG}}_{\mathsf{Enc}}$ 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 $\mathfrak{C}^{\mathsf{PVC}}_{\mathsf{Enc}}$ for PVCs on the same machine (see Footnote 4 on Page 4) to obtain a fair comparison, with all results shown in Figure 3.

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

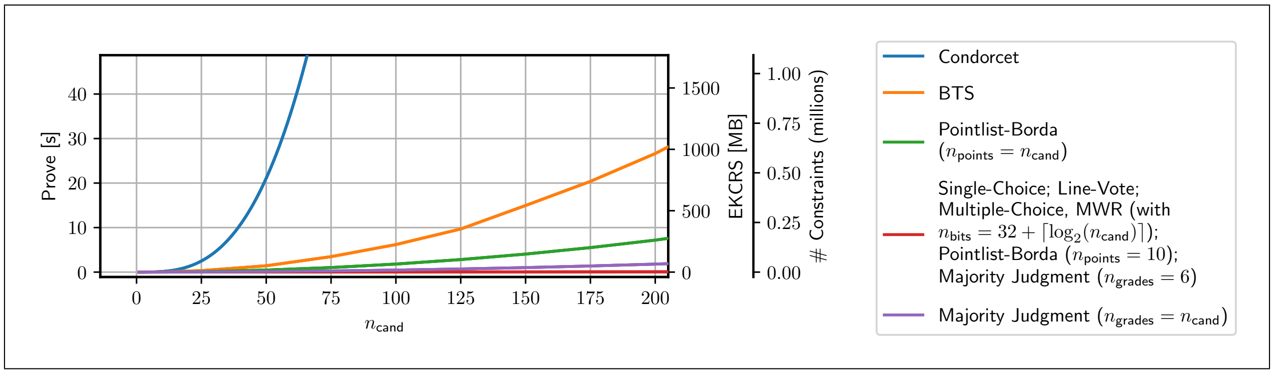

Fig. 4: Prover runtime, $\mathsf{CRS_{EK}}$ size, and constraints of $\mathfrak{C}_{\mathsf{Voting}}$ for several voting methods. For Pointlist-Borda, we use $\mathcal{L} = \{1, \ldots, n_{\mathsf{points}}\}$, for Multi-Vote and MWR, we use $t = 2^{32} - 1$ and $n_{\mathsf{max}} = n_{\mathsf{cand}} \cdot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

of 50 EEG ciphertexts a voter can still compute a proof in less than 30 seconds using a $\mathsf{CRS_{EK}}$ of about 1 GB, which is already good enough to be viable in a wide range of settings and election types. While reducing the maximal bit length of a plaintext slightly improves performance, the difference between 1, 16, and 32 bit is negligible due to the impact of the several exponentiations with large randomnesses. For a more detailed discussion of practicality in various situations, see Section 4.

3.2 Constructing and Optimizing $\mathfrak{C}_{\mathsf{Voting}}$

Since the subcircuit $\mathfrak{C}_{\mathsf{Voting}}$ 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, Condorcet, and Majority Judgment. 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 $\mathfrak{C}_{\mathsf{Voting}}$ for all of our choice spaces in Figure 4.

Since $\mathfrak{C}_{\mathsf{Voting}}$ is independent of the method used for encrypting ballots - thanks to the modularity of $\mathfrak{C}$ - we can reuse the existing sub-circuits for Single-Vote, Multi-Vote, Borda Count, Condorcet, and Majority Judgment from [35] 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 [35].

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_{single} defined in Section 1. Checking that $b \in C_{\mathsf{single}}$ 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_{cand} candidates, $\mathfrak{C}^{\mathsf{single}}_{\mathsf{Voting}}$ thus consists of $n_{\mathsf{cand}} + 1$ constraints. This yields a very small $\mathsf{CRS_{EK}}$ of less than 1 MB and proof times of less than 0.05 seconds for any realistic number of candidates (see Figure 4).

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

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

The circuit $\mathfrak{C}^{\mathsf{multi}}_{\mathsf{Voting}}$ for C_{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_{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_{bits} of $\sum_{i=1}^{n_{\mathsf{cand}}} v_i$ which is, in turn, determined by the bit sizes of t and n_{max}. The complete circuit requires about $(n_{\mathsf{bits}}+1) \cdot (n_{\mathsf{cand}}+1)$ constraints (the exact number depends on t and n_{max}), which - even for unrealistically high values of n_{bits} such as $n_{\mathsf{bits}} = 41$ - is still very small for any realistic number of candidates. Hence, performance of $\mathfrak{C}^{\mathsf{multi}}_{\mathsf{Voting}}$ is essentially the same as for $\mathfrak{C}^{\mathsf{single}}_{\mathsf{Voting}}$ (see Figure 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_{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. A use case might be voting for core office hours: let options be hours of the day with voters/workers using YES-votes to indicate a single continuous period of availability. The choice space can be formalized as follows:

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

A corresponding circuit $\mathfrak{C}^{\mathsf{line}}_{\mathsf{Voting}}$ can be built easily analogous to $\mathfrak{C}^{\mathsf{multi}}_{\mathsf{Voting}}$: $\mathfrak{C}^{\mathsf{line}}_{\mathsf{Voting}}$ 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 $2n_{\mathsf{cand}}$ 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:

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

A potential application of this and similar rules are surveys with conditional follow-up questions. E.g., $v_1 \in \{0,1\}$ might be a vote on a YES-/NO-question about a change in law, with 0 corresponding to yes. Then, only those in favor of a change are asked about different new laws v_2 and v_3 of which they must accept at least one (accept also corresponding to 0). The corresponding circuit $\mathfrak{C}^{\mathsf{MWR}}_{\mathsf{Voting}}$ is again easy to construct: use $\mathfrak{C}^{\mathsf{multi}}_{\mathsf{Voting}}$ 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 Figure 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 [46] and the Eurovision Song Contest (ESC) [26]. As suggested in [35], 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_{points} many distinct positive numbers. Voters then construct their ballots by assigning each number in $\mathcal{L}$ to one candidate and, if $n_{\mathsf{points}} < n_{\mathsf{cand}}$, 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 $n_{\mathsf{cand}} - n_{\mathsf{points}}$ candidates tied for the last place. Formally, the choice space is as follows:

$$\begin{aligned} C_{\mathsf{BordaPointList}}(\mathcal{L}) := \Big\{(v_1, \dots, v_{n_{\mathsf{cand}}}) \Big| & (\forall p \in \mathcal{L}\ \exists i : v_i = p) \\ & \wedge |\{i \in [1, n_{\mathsf{cand}}] \mid v_i = 0\}| = n_{\mathsf{cand}} - n_{\mathsf{points}} \Big\}. \end{aligned}$$

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

There are many different ways to design Borda methods that additionally allow for ties between candidates at arbitrary positions. For example, for Kryvos [35] we considered such a Borda method that we called *Borda tournament style (BTS)*. Since there are no specialized ballot validity ZKPs for BTS, it serves as a great case study for the potential of GPZKPs.

To build a BTS ballot, voters first choose a ranking $r = (r_1, \ldots, r_{n_{\mathsf{cand}}}) \in \mathbb{N}^{n_{\mathsf{cand}}}$ of candidates, where $r_i > r_j$ means that candidate i is ranked worse than candidate j. Note that the voter can tie arbitrarily many candidates at arbitrary ranks. In the ballot, a candidate i then receives $a \in \mathbb{N}$ points for each candidate that this voter ranked lower than i and $b \in \mathbb{N}$ points for each candidate that i is tied with. Here, a and b are some pre-defined parameters of the election with $a > b$, e.g., $a = 2$ and $b = 1$. Formally, this gives the following choice space:

$$\begin{aligned} C_{\mathsf{BordaTournamentStyle}} := \Big\{ (x_1, \ldots, x_{n_{\mathsf{cand}}}) \Big| \exists (r_1, \ldots, r_{n_{\mathsf{cand}}}) \in \mathbb{N}^{n_{\mathsf{cand}}} \text{ s.t. } \forall i : \\ x_i = a \cdot |\{j \in \{1, \ldots, n_{\mathsf{cand}}\} | r_j > r_i\}| + \\ b \cdot |\{j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\} | r_j = r_i\}| \Big\}. \end{aligned}$$

To improve efficiency, the corresponding circuit $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{BTS}}$ takes a witness w as input that contains not just the plain ballot b but also the ranking $r = (r_1, \ldots, r_{n_{\mathsf{cand}}})$. From r, $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{BTS}}$ then computes the corresponding ballot b' and verifies whether $\mathsf{b} = \mathsf{b}'$.[8] The performance of $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{BTS}}$ is also quadratic in n_{cand} but worse than $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{BordaPointList}}$ due to increased complexity of the choice space. However, performance still remains practical even for $n_{\mathsf{cand}} = 100$ (see Figure 4).

Condorcet Methods. In Condorcet methods, which are, e.g., used for internal elections of the Debian project [24], 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 [22, 33, 35].

Specifically, given a ranking $r = (r_1, \ldots, r_{n_{\mathsf{cand}}}) \in \mathbb{N}^{n_{\mathsf{cand}}}$ of candidates (where $r_i > r_j$ means that candidate i is ranked worse than candidate j), a voter constructs her ballot as an $n_{\mathsf{cand}} \times n_{\mathsf{cand}}$ matrix A with 1 at position (i,j) if candidate i is ranked better than candidate j and 0 otherwise. Note that hence n_{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 [22]. The choice space then is:

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

[8] As a slight optimization, in the combined circuit $\mathfrak{C}$ the check $\mathsf{b} = \mathsf{b}'$ can be omitted by using the wires containing b' (instead of b) as input for the subcircuit $\mathfrak{C}_{\mathsf{Enc}}$.

The circuit $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{Condorcet}}$ extends the one proposed in [35], 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. [9] 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 are expressed through 1-entries instead of 0-entries. For this, $\mathfrak{C}_{\mathsf{Voting}}$ computes a "check matrix" B with $B_{ij} := 1 - A_{ji}$, which does not require any new constraints (intuitively, because this computation can be performed as part of other already existing 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 $\mathsf{CRS}_{\mathsf{EK}}$ of about 90 MB and a proof time of about 2.5 seconds (see Figure 4).

Majority Judgment. Majority Judgment is an election method where a voter can grade each candidate independently according to a pre-defined list of n_{grades} grades. The election winner(s) are then determined based on the candidate(s) who received the highest median grade. Majority Judgment is a common method in political research polling, but variants have also found application for judging sports competitions such as Olympic figure skating. Following Canard et al. [18], we consider a ballot as an $(n_{\mathsf{cand}} \times n_{\mathsf{grades}})$-matrix A, where assigning the j-th grade to the i-th candidate is represented by the i-th row of A containing zeros everywhere except for the j-th column, where we set $A_{ij} := 1$. More precisely, as described in [35], the choice space for Majority Judgment is given as

$$C_{\mathsf{MajorityJudgement}}(n_{\mathsf{grades}}) := \left\{ A \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{grades}}} \mid \forall i \in [n_{\mathsf{cand}}] : \sum_{j \in [n_{\mathsf{grades}}]} A_{ij} = 1 \right\}.$$

We note that $C_{\mathsf{MajorityJudgement}}$ can be seen as n_{cand}-fold product of C_{single}, which is why our circuit $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{MajorityJudgement}}$ for $C_{\mathsf{MajorityJudgement}}$ essentially applies $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{single}}$ to each row of a ballot. Thus, the performance of $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{MajorityJudgement}}$ is n_{grades} times worse than the performance of $\mathfrak{C}_{\mathsf{Voting}}^{\mathsf{single}}$. We have benchmarked two variations, where the first uses $n_{\mathsf{grades}} = n_{\mathsf{cand}}$, leading to a quadratic scaling in the number of candidates, and the second variant uses $n_{\mathsf{grades}} = 6$, as in the common letter grade scale, see Figure 4.

4 Overall benchmarks for Proving Ballot Validity

Following the outline given in Section 3, we can now combine the encryption subcircuit $\mathfrak{C}_{\mathsf{Enc}}$ with a suitable plaintext bit size from Section 3.1 and a voting subcircuit $\mathfrak{C}_{\mathsf{Voting}}$ from Section 3.2 to obtain complete circuits $\mathfrak{C}$ for proving ballot validity. Our benchmarks of prover runtime, $\mathsf{CRS}_{\mathsf{EK}}$ size, and constraints for these circuits using EEG encryption and depending on the number of candidates n_{cand} are given in the top half of Figure 5. For comparison, in the bottom half of Figure 5 we provide our benchmarks for ballots computed as PVCs using the constructions of [35]. As mentioned in Section 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 $\mathsf{CRS}_{\mathsf{VK}}$ is a subset of $\mathsf{CRS}_{\mathsf{EK}}$ we do not provide separate benchmarks, but its size is always in the order of ~ 20 KB and hence negligible.

The performance of Groth16 for the combined circuit $\mathfrak{C}$ is essentially the sum of the subcircuits $\mathfrak{C}_{\mathsf{Enc}}$ and $\mathfrak{C}_{\mathsf{Voting}}$ and thus dominated by the much slower $\mathfrak{C}_{\mathsf{Enc}}$. Note that the performance of $\mathfrak{C}_{\mathsf{Enc}}$ in Figure 3 was given depending on the number N of plaintexts, while for the combined circuit $\mathfrak{C}$ we consider performance depending on number n_{cand} of candidates. All but two choice spaces use one plaintext per candidate, i.e., $N = n_{\mathsf{cand}}$, so the benchmarks given in Figure 5 mostly retain the linear behavior of $\mathfrak{C}_{\mathsf{Enc}}$, potentially plus some small non-linear overhead caused by $\mathfrak{C}_{\mathsf{Voting}}$. The exceptions are Condorcet and Majority Judgment ballots, where $N = n_{\mathsf{cand}}^2$ resp. $N = n_{\mathsf{cand}} \cdot n_{\mathsf{grades}}$. For Condorcet and Majority Judgment with $n_{\mathsf{grades}} = n_{\mathsf{cand}}$ this causes visibly quadratic behavior in the combined circuit due to $\mathfrak{C}_{\mathsf{Enc}}$ (plus some smaller cubic overhead

[9] One can instead check that $A_{ij} + A_{ji} = 1$ to prevent ties as proposed in [35]. This yields the same number of constraints.

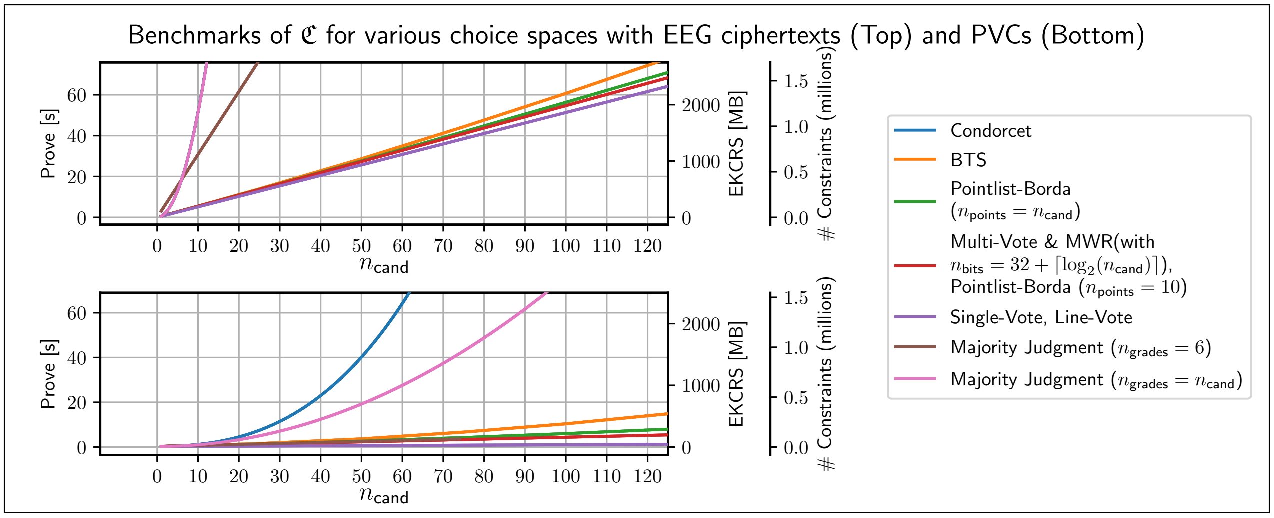

Fig. 5: Comparison of full ballot validity proofs. Majority Judgment, Condorcet, Single-, and Line-Vote use $\mathfrak{C}_{\mathsf{Enc}}$ with 1-bit plaintexts; all other choice spaces use 32-bit plaintexts. Note that in the upper figure, the lines for Condorcet and Majority Judgment with $n_{\mathsf{cand}} = n_{\mathsf{grades}}$ overlap.

for Condorcet ballots due to $\mathfrak{C}^{\mathsf{Condorcet}}_{\mathsf{Voting}}$ that explains the distance between the two graphs in the case of PVCs).

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 $\mathfrak{C}_{\mathsf{Enc}}$, 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 outliers are Condorcet and Majority Judgment (with large values for n_{grades}), for which computing a proof quickly becomes impractical due to the higher number of ciphertexts. We note, however, that real-world Condorcet elections, such as [24], rarely have more than 10 candidates. Similarly, a Majority Judgment election with more than 20 grades and/or candidates seems unrealistic - typical applications use 6 to 10 grades to grade only a few candidates [6]. For such cases, a proof of ballot validity can still be computed in less than a minute. As for the size of $\mathsf{CRS}_{\mathsf{EK}}$, 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 $\mathsf{CRS}_{\mathsf{EK}}$ 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., Lipmaa, H., Siim, J., Zajac, M.: On Subversion-Resistant SNARKs. IACR Cryptol. ePrint Arch. **2020**, 668 (2020)
2. Abdolmaleki, B., et al.: UC-Secure CRS Generation for SNARKs. In: AFRICACRYPT 2019, Proceedings. LNCS, vol. 11627, pp. 99–117. Springer (2019)
3. 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)

4. Ames, S., et al.: Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In: ACM CCS 2017. pp. 2087–2104 (2017)
5. Aranha, D.F., Karabina, K., Longa, P., Gebotys, C.H., López, J.: Faster explicit formulas for computing pairings over ordinary curves. In: Paterson, K.G. (ed.) Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 48–68. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_5, https://doi.org/10.1007/978-3-642-20465-4_5
6. Balinski, M., Laraki, R.: Election by majority judgment: experimental evidence. In: In situ and laboratory experiments on electoral law reform: French presidential elections, pp. 13–54. Springer (2010)
7. Barreto, P.S.L.M., Lynn, B., Scott, M.: Constructing elliptic curves with prescribed embedding degrees. In: Cimato, S., Galdi, C., Persiano, G. (eds.) Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers. Lecture Notes in Computer Science, vol. 2576, pp. 257–267. Springer (2002). https://doi.org/10.1007/3-540-36413-7_19, https://doi.org/10.1007/3-540-36413-7_19
8. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S.E. (eds.) Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3897, pp. 319–331. Springer (2005). https://doi.org/10.1007/11693383_22, https://doi.org/10.1007/11693383_22
9. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion. In: Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10032, pp. 777–804 (2016)
10. 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)
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 103–128. Springer (2019). https://doi.org/10.1007/978-3-030-17653-2_4, https://doi.org/10.1007/978-3-030-17653-2_4
12. 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)
13. Ben-Sasson, E., et al.: Scalable, Transparent, and Post-Quantum Secure Computational Integrity. IACR Cryptology ePrint Archive **2018**, 46 (2018)
14. Bernstein, D.J.: Curve25519: New diffie-hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3958, pp. 207–228. Springer (2006). https://doi.org/10.1007/11745853_14, https://doi.org/10.1007/11745853_14
15. Botrel, G., et al.: Consensys/gnark: v0.9.0 (Feb 2023). https://doi.org/10.5281/zenodo.5819104
16. 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)
17. Bünz, B., et al.: Bulletproofs: Short Proofs for Confidential Transactions and More. In: SP 2018. (2018)
18. Canard, S., Pointcheval, D., Santos, Q., Traoré, J.: Practical Strategy-Resistant Privacy-Preserving Elections. In: Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11099, pp. 331–349. Springer (2018)
19. Chaum, D., Pedersen, T.P.: Wallet Databases with Observers. In: CRYPTO '92. LNCS, vol. 740, pp. 89–105. Springer (1992)
20. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In: EUROCRYPT 2020. (2020)
21. Cortier, V., Gaudry, P., Glondu, S.: Belenios: a simple private and verifiable electronic voting system. Foundations of Security, Protocols, and Equational Reasoning: Essays Dedicated to Catherine A. Meadows pp. 214–238 (2019)
22. 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)
23. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: CRYPTO 1994. Springer (1994)
24. Debian Project: Debian Voting Information. https://www.debian.org/vote/ (2024)

25. 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)
26. European Broadcasting Union: Eurovision Song Contest - How it works. https://eurovision.tv/about/how-it-works (2024)
27. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Non-interactive arguments of Knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)
28. Gaudry, P.: Some zk security proofs for belenios (2017)
29. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 626–645. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_37, https://doi.org/10.1007/978-3-642-38348-9_37
30. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In: USENIX Security Symposium 2016. pp. 1069–1083. USENIX Association (2016)
31. Groth, J.: Non-interactive Zero-Knowledge Arguments for Voting. In: ACNS 2005. LNCS, vol. 3531, pp. 467–482 (2005)
32. Groth, J.: On the Size of Pairing-Based Non-interactive Arguments. In: EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer (2016)
33. 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)
34. Hopwood, D.E., et al.: Zcash Protocol Specification. https://zips.z.cash/protocol/protocol.pdf (2024)
35. Huber, N., et al.: Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In: CCS 2022. pp. 1443–1457. ACM (2022)
36. Huber, N., et al.: Implementation of our Circuits. https://github.com/HicolasNuber/ballotsnarks (2024)
37. Joaquim, R.: How to prove the validity of a complex ballot encryption to the voter and the public. JISA **19**(2), 130–142 (2014)
38. Kim, T., Barbulescu, R.: Extended tower number field sieve: A new complexity for the medium prime case. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 543–571. Springer (2016). https://doi.org/10.1007/978-3-662-53018-4_20, https://doi.org/10.1007/978-3-662-53018-4_20
39. Kohlweiss, M., Maller, M., Siim, J., Volkhov, M.: Snarky ceremonies. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 98–127. Springer (2021). https://doi.org/10.1007/978-3-030-92078-4_4, https://doi.org/10.1007/978-3-030-92078-4_4
40. Kosba, A., et al.: C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs. Cryptology ePrint Archive (2015)
41. 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)
42. Masson, S., Sanso, A., Zhang, Z.: Bandersnatch: a fast elliptic curve built over the BLS12-381 scalar field. IACR Cryptol. ePrint Arch. p. 1152 (2021), https://eprint.iacr.org/2021/1152
43. 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)
44. Morais, E., et al.: A survey on zero knowledge range proofs and applications. SN Applied Sciences **1**, 1–17 (2019)
45. 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: CHES 2001. LNCS, vol. 2162, pp. 126–141. Springer (2001)
46. Republic of Nauru: Electoral Act No. 15. http://ronlaw.gov.nr/nauru_lpms/files/acts/d83250a1ebdc56c1701fa7aa245af5b1.pdf (2024)
47. scipr-lab: libsnark. https://github.com/scipr-lab/libsnark (2024)
48. Sean Bowe: BLS12-381: New zk-SNARK elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/ (2017)
49. Vercauteren, F.: Optimal pairings. IEEE Trans. Inf. Theory **56**(1), 455–461 (2010). https://doi.org/10.1109/TIT.2009.2034881, https://doi.org/10.1109/TIT.2009.2034881

A Succinct Non-Interactive Arguments of Knowledge (SNARKs)

In this section, we recall (and slightly adapt) the definition of succinct non-interactive arguments of knowledge (SNARKs) given in [32]. We note that several further definitions exist; in particular, the notion of succinctness differs among various authors.

Formally, we consider a relation generator $\mathcal{R}$ that, given security parameter η, outputs a binary relation R that is decidable in time polynomial in η. This relation generator may also output some auxiliary information z for an adversary.

Definition 1 (Non-Interactive Argument [32]). *A* non-interactive argument *for a relation $R \leftarrow \mathcal{R}(\eta)$ is a quadruple of probabilistic algorithms* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *running in time polynomial in η such that*

- $\big(\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{EK}}, \mathsf{CRS}_{\mathsf{VK}}), \mathsf{st}\big) \leftarrow \mathsf{Setup}(R)$*, i.e.,* Setup *produces a common reference string* CRS *and a simulation trapdoor* st *for the relation R.*
- $\pi \leftarrow \mathsf{Prove}(R, \mathsf{CRS}_{\mathsf{EK}}, (\mathsf{Stmt}, \mathsf{Wtns}) \in R)$*, i.e.,* Prove *takes* $\mathsf{CRS}_{\mathsf{EK}}$ *and statement-witness pair* $(\mathsf{Stmt},\mathsf{Wtns})$ *and returns an argument π.*
- $0/1 \leftarrow \mathsf{Verify}(R, \mathsf{CRS}_{\mathsf{VK}}, \mathsf{Stmt}, \pi)$*, i.e.,* Verify *takes* $\mathsf{CRS}_{\mathsf{VK}}$*, statement* Stmt *and argument π and returns* 0 *(reject) or* 1 *(accept).*
- $\pi \leftarrow \mathsf{Sim}(R, \mathsf{st}, \mathsf{Stmt})$*, i.e,* Sim *takes statement* Stmt *and simulation trapdoor* st *and returns an argument π.*

Definition 2 (Perfect Zero-Knowledge Succinct Non-Interactive Argument of Knowledge [32]). *A non-interactive argument as defined in Definition 1 is a* perfect zero-knowledge succinct argument of knowledge*, or a* (perfect-)zk-SNARK[10] *for short, if the following additional properties hold:*

- Perfect Completeness: *When using an honestly-generated common reference string (i.e., a* CRS *obtained from running* Setup*), an honestly-generated proof for* $(\mathsf{Stmt}, \mathsf{Wtns}) \in R$ *is always accepted. Formally that is* $\forall \eta \in \mathbb{N}, R \in \mathcal{R}(\eta), (\mathsf{Stmt}, \mathsf{Wtns}) \in R:$

$$\mathsf{Pr}\Big[\big(\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{EK}}, \mathsf{CRS}_{\mathsf{VK}}), \mathsf{st}\big) \leftarrow \mathsf{Setup}; \pi \leftarrow \mathsf{Prove}(R, \mathsf{CRS}_{\mathsf{EK}}, \mathsf{Stmt}, \mathsf{Wtns}) : \mathsf{Verify}(R, \mathsf{CRS}_{\mathsf{VK}}, \mathsf{Stmt}, \pi) = 1\Big] = 1.$$

- Perfect Zero-Knowledge: *When using an honestly generated* CRS*, no adversary* A *can distinguish between proofs generated from a statement-witness pair* $(\mathsf{Stmt}, \mathsf{Wtns})$ *using* Prove *and proofs generated by the simulator* Sim *using the simulation trapdoor* st *(even when* A *is given access to the simulation trapdoor* st*).*
  Formally that is $\forall \eta \in \mathbb{N}, (R, z) \leftarrow \mathcal{R}(\eta), (\mathsf{Stmt}, \mathsf{Wtns}) \in R$ *and all adversaries* A *the following holds true:*

$$\begin{aligned}&\mathsf{Pr}\left[\big(\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{EK}}, \mathsf{CRS}_{\mathsf{VK}}), \mathsf{st}\big) \leftarrow \mathsf{Setup}; \pi \leftarrow \mathsf{Prove}(R, \mathsf{CRS}_{\mathsf{EK}}, \mathsf{Stmt}, \mathsf{Wtns}) : \mathsf{A}(R,z,\mathsf{CRS},\mathsf{st},\pi) = 1\right]\\ =&\mathsf{Pr}\left[\big(\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{EK}}, \mathsf{CRS}_{\mathsf{VK}}), \mathsf{st}\big) \leftarrow \mathsf{Setup}; \pi \leftarrow \mathsf{Sim}(R, \mathsf{st}, \mathsf{Stmt}) : \mathsf{A}(R,z,\mathsf{CRS},\mathsf{st},\pi) = 1\right].\end{aligned}$$

- Computational Knowledge-Soundness: *Up to an error probability that is negligible in η, a polynomial-time adversary A with access to an honestly-generated* CRS *can only create a proof for statement* Stmt *such that* Verify *accepts if* A *knows a witness* Wtns *such that* $(\mathsf{Stmt}, \mathsf{Wtns}) \in R$*. This is formalized by requiring that for all polynomial-time adversaries* A *there exists a polynomial-time extractor algorithm* $\mathcal{K}(\mathsf{A})$ *such that the following holds:*

$$\begin{aligned}\mathsf{Pr}\Big[&(R, z) \leftarrow \mathcal{R}(\eta), \big(\mathsf{CRS} = (\mathsf{CRS}_{\mathsf{EK}}, \mathsf{CRS}_{\mathsf{VK}}), \mathsf{st}\big) \leftarrow \mathsf{Setup};\\ &((\mathsf{Stmt}, \pi); \mathsf{Wtns}) \leftarrow (\mathsf{A} \| \mathcal{K}(\mathsf{A}))(R, z, \mathsf{CRS}) :\\ &(\mathsf{Stmt}, \mathsf{Wtns}) \notin R \textit{ and } \mathsf{Verify}(R, \mathsf{CRS}_{\mathsf{VK}}, \mathsf{Stmt}, \pi) = 1\Big] = \mathsf{negl}(\eta),\end{aligned}$$

[10] We usually use the term SNARK when referring to a (perfect-)zk-SNARK and implicitly assume the zero-knowledge property.

where $((\mathsf{Stmt}, \pi); \mathsf{Wtns}) \leftarrow (\mathsf{A} \| \mathcal{K}(\mathsf{A}))(R, z, \mathsf{CRS})$ *denotes that* A *on input* (R, z, CRS) *outputs* (Stmt, π), *and* $\mathcal{K}(\mathsf{A})$ *on the same input (including random coins) outputs* Wtns.[11]

– Succinctness: *The runtime of* Verify *is polynomial in* $\eta + |\mathsf{Stmt}|$ *and the proof size* $|\pi|$ *is polynomial in* η.[12]

We emphasize that the security notions of *computational knowledge-soundness* and *perfect zero-knowledge* both require an honestly generated CRS. Moreover, *computational knowledge-soundness* is only defined to hold against adversaries that do not learn the simulation trapdoor st. Indeed, if an adversary was to learn st, it can easily create a proof π for a statement Stmt without knowing a witness Wtns such that $(\mathsf{Stmt}, \mathsf{Wtns}) \in R$ by invoking the simulation algorithm Sim. For this reason, the simulation trapdoor st is sometimes called the *toxic waste* of the Setup algorithm. While it is required for generating CRS, it is crucial that no malicious party can learn st. A SNARK proven secure in the above model is said to require a *trusted setup*. While SNARK constructions that do not require a trusted setup exist - e.g., [4, 11] - the Groth16 SNARK, which we use for efficiency reasons, does fall into the above definition and, hence, requires a trusted setup. We will discuss implications and mitigations of this in Appendix C.

B The Groth16 SNARK.

In this section, we recall the Groth16 SNARK construction [32] and its instantiation in more detail in order to explain how it can be used with our circuits for ballot validity.

B.1 Rank-1-Constraint Systems and Quadratic Arithmetic Programs

As mentioned in Section 2, we base our implementation on arithmetic circuits over some finite field $\mathbb{F}_r$. These circuits are converted to a system of so-called constraints, more precisely a *rank-1-constraint system (R1CS)* over $\mathbb{F}_r$. A constraint over n variables $a_1, \ldots, a_n$ (which correspond to the number of wires in the arithmetic circuit) is an equation of the form

$$\left(u_0 + \sum_{i=1}^{n} a_i u_i\right) \cdot \left(v_0 + \sum_{i=1}^{n} a_i v_i\right) = w_0 + \sum_{i=1}^{n} a_i w_i,$$

where $u_i, v_i, w_i \in \mathbb{F}_r$ are constants defining the constraint. A *valid assignment* to an R1CS is an assignment of the n variables $a_1, \ldots, a_n$ to elements of $\mathbb{F}_r$ such that all constraints in the R1CS are satisfied. If the R1CS is derived from an arithmetic circuit, this corresponds to a valid assignment to the circuit's wires.

For simplicity, we write an R1CS with μ constraints over ω variables as three matrices $U, V, W \in \mathbb{F}_r^{\mu \times (\omega+1)}$, where each row corresponds to one constraint.

Such an R1CS can then be compiled into a *Quadratic Arithmetic Program (QAP)* over $\mathbb{F}_r$ [29], which is a set $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$ of polynomials in $\mathbb{F}_r[X]$, where $T(X) = \prod_{i=1}^{\mu}(X - r_i)$ for some mutually distinct $r_i \in \mathbb{F}_r$ - usually one choses the r_i to be μ-th roots of unity in $\mathbb{F}_r$ for efficiency reasons - and $\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j$ are the unique polynomials of degree $\leq \mu - 1$ such that

$$\forall i = 1, \ldots, \mu : \mathcal{U}_j(r_i) = u_{ij}, \mathcal{V}_j(r_i) = v_{ij}, \text{ and } \mathcal{W}_j(r_i) = w_{ij}.$$

[11] We recall that a distinguishing feature between zero-knowledge *arguments* and zero-knowledge *proofs* is that the soundness notion of the former only considers polynomially-bounded adversaries, while in the latter one typically allows any adversaries with bounded runtime. In practice, however, the term ZKP is often also used for zero-knowledge *arguments*, which is why we also often use the term ZKP in this work.

[12] Succinctness captures the properties of having small proof sizes and an efficient Verify algorithm. While the concrete interpretations of "small" and "efficient" differ among the literature, we follow Groth [32] in this definition of succinctness.

A QAP is called *satisfiable* if there are elements $\lambda_1, \dots, \lambda_\omega \in \mathbb{F}_r$ such that the polynomial

$$P_\lambda(X) := \left(\mathcal{U}_0 + \sum_{j=1}^{\omega} \lambda_j \mathcal{U}_j\right) \cdot \left(\mathcal{V}_0 + \sum_{j=1}^{\omega} \lambda_j \mathcal{V}_j\right) - \left(\mathcal{W}_0 + \sum_{j=1}^{\omega} \lambda_j \mathcal{W}_j\right)$$

of degree $\deg(P_\lambda) \le 2\mu - 2$ is divisible by the polynomial T in $\mathbb{F}_r[X]$. In this case, we call $\lambda = (\lambda_1, \dots, \lambda_\omega)$ a *valid assignment*. This corresponds to a valid assignment for the R1CS and, in turn, to a valid assignment for the arithmetic circuit.

B.2 The Groth16 Protocol

Given a QAP $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$ over $\mathbb{F}_r$ with $\mu := \deg(T)$, the goal of the Groth16 protocol is for a prover $\mathcal{P}$ to compute a ZKP π that shows that $\mathcal{P}$ knows a valid assignment $\lambda = (\lambda_1, \dots, \lambda_\omega) \in \mathbb{F}_r^\omega$ for QAP.

Typically, some of the assignment values, say the first ω_{Stmt} values $\lambda_{\mathsf{Stmt}} := (\lambda_1, \dots, \lambda_{\omega_{\mathsf{Stmt}}})$, are public (*statement values*) while the remaining $\omega_{\mathsf{Wtns}} = \omega - \omega_{\mathsf{Stmt}}$ values $\lambda_{\mathsf{Wtns}} = (\lambda_{\omega_{\mathsf{Stmt}}+1}, \dots, \lambda_\omega)$ are private (*witness values*) and only known to $\mathcal{P}$. In particular, nobody should learn the witness values from π.

For this purpose, let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be (multiplicative) groups of prime order $r := |\mathbb{F}_r|$ such that there is an efficiently computable, non-degenerate bilinear pairing $e(\cdot,\cdot) : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let g_1,g_2 be generators of $\mathbb{G}_1$ resp. $\mathbb{G}_2$, and $e(g_1, g_2)$ be a generator of $\mathbb{G}_T$. We denote $\mathsf{params} := \{\mathbb{G}_1,\mathbb{G}_2,g_1,g_2,e(\cdot,\cdot),\mathbb{F}_r\}$; $\mathbb{F}_r$ is sometimes called the *scalar field* of the pairing groups $\mathbb{G}_1, \mathbb{G}_2$. Then, as mentioned in Section 2, the Groth16 protocol consists of three algorithms - Setup, Prove, and Verify - which we will describe in the following.

The Setup Algorithm The Setup algorithm starts by sampling the *simulation trapdoor* st consisting of 5 uniformly random field elements $\alpha, \beta, \gamma, \delta, \tau \xleftarrow{\$} \mathbb{F}_r$. Using st and QAP, it then computes the (evaluation key) common reference string $\mathsf{CRS}_{\mathsf{EK}}$ as $\mathsf{CRS}_{\mathsf{EK}} = (\mathsf{CRS}_{\mathsf{EK}1}, \mathsf{CRS}_{\mathsf{EK}2})$, where $\mathsf{CRS}_{\mathsf{EK}1}$ consisting of elements of $\mathbb{G}_1$ and $\mathsf{CRS}_{\mathsf{EK}2}$ consisting of elements $\mathbb{G}_2$ are defined as follows, where we color the different elements in order to make them easily recognizable at later points:[13]

$$\begin{aligned}
\mathsf{CRS}_{\mathsf{EK}1} = \Bigg\{ & g_1^{\alpha}, g_1^{\beta}, g_1^{\delta}, \left(g_1^{\tau^j}\right)_{j=0}^{\mu-1}, \left(g_1^{\mathcal{U}_i(\tau)}\right)_{i=0}^{\omega}, \left(g_1^{\mathcal{V}_i(\tau)}\right)_{i=0}^{\omega}, \left(g_1^{\frac{\tau^j \cdot T(\tau)}{\delta}}\right)_{j=0}^{\mu-2}, \\
& \left(g_1^{\frac{\beta\mathcal{U}_j(\tau)+\alpha\mathcal{V}_j(\tau)+\mathcal{W}_j(\tau)}{\gamma}}\right)_{j=0}^{\omega_{\mathsf{Stmt}}}, \left(g_1^{\frac{\beta\mathcal{U}_{j+\omega_{\mathsf{Stmt}}}(\tau)+\alpha\mathcal{V}_{j+\omega_{\mathsf{Stmt}}}(\tau)+\mathcal{W}_{j+\omega_{\mathsf{Stmt}}}(\tau)}{\delta}}\right)_{j=1}^{\omega_{\mathsf{Wtns}}} \Bigg\}, \\
\mathsf{CRS}_{\mathsf{EK}2} = \Bigg\{ & g_2^{\beta}, g_2^{\gamma}, g_2^{\delta} \left(g_2^{\tau^j}\right)_{j=0}^{\mu-1}, \left(g_2^{\mathcal{V}_i(\tau)}\right)_{i=0}^{\omega} \Bigg\}.
\end{aligned}$$

As mentioned in Section 2, running Prove requires the full $\mathsf{CRS}_{\mathsf{EK}}$. However, for running Verify only the smaller $\mathsf{CRS}_{\mathsf{VK}} = (\mathsf{CRS}_{\mathsf{VK}1},\mathsf{CRS}_{\mathsf{VK}2})$ is required, where $\mathsf{CRS}_{\mathsf{VK}1}$ consisting of elements of $\mathbb{G}_1$ and $\mathsf{CRS}_{\mathsf{VK}2}$ consisting of elements $\mathbb{G}_2$ are defined as follows:

$$\begin{aligned}
\mathsf{CRS}_{\mathsf{VK}1} &= \left\{ g_1^{\alpha}, \left(g_1^{\frac{\beta\mathcal{U}_j(\tau)+\alpha\mathcal{V}_j(\tau)+\mathcal{W}_j(\tau)}{\gamma}}\right)_{j=0}^{\omega_{\mathsf{Stmt}}} \right\} \subset \mathsf{CRS}_{\mathsf{EK}1}, \\
\mathsf{CRS}_{\mathsf{VK}2} &= \left\{ g_2^{\beta}, g_2^{\gamma}, g_2^{\delta} \right\} \subset \mathsf{CRS}_{\mathsf{EK}2}
\end{aligned}$$

[13] The values $\left(g_1^{\mathcal{U}_i(\tau)}\right)_{i=0}^{\omega}$, $\left(g_1^{\mathcal{V}_i(\tau)}\right)_{i=0}^{\omega}$, and $\left(g_2^{\mathcal{V}_i(\tau)}\right)_{i=0}^{\omega}$ are often not considered part of $\mathsf{CRS}_{\mathsf{EK}}$ as they can be computed from the description of QAP and other values in $\mathsf{CRS}_{\mathsf{EK}}$. This offers a trade-off between the size of $\mathsf{CRS}_{\mathsf{EK}}$ and the prover runtime. For ease of presentation, we include them here in $\mathsf{CRS}_{\mathsf{EK}}$ and refer to the discussion in [32] for details.

To summarize the CRS sizes, consider an R1CS with μ constraints over ω variables, $\omega_{\mathsf{Stmt}} \leq \omega$ of which are associated with public values. Then, $\mathsf{CRS}_{\mathsf{EK}}$ consists of $2\mu + 3\omega + 5$ elements from $\mathbb{G}_1$ and $\omega + \mu + 3$ elements from $\mathbb{G}_2$.[14] Meanwhile, $\mathsf{CRS}_{\mathsf{VK}}$ consists of only $\omega_{\mathsf{Stmt}} + 2$ elements from $\mathbb{G}_1$ and only three elements from $\mathbb{G}_2$.

The Prove Algorithm Recall that the goal of the prover $\mathcal{P}$ is to compute a ZKP π that shows that $\mathcal{P}$ knows a valid assignment $\lambda = (\lambda_1, \ldots, \lambda_\omega)$ for $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$. Since an assignment λ is valid if and only if P_λ (as defined above) divides T, this is equivalent to showing that $\mathcal{P}$ knows a polynomial $H_\lambda \in \mathbb{F}_r[X]$ of degree $\deg(H_\lambda) \leq \deg(T) - 2 = \mu - 2$ such that $H_\lambda \cdot T = P_\lambda$. In order to do so, the Prove algorithm proceeds as described in Figure 6.

$\mathsf{Prove}(\mathsf{params}, \mathsf{QAP}, \mathsf{CRS}_{\mathsf{EK}}, \lambda = (\lambda_1, \ldots, \lambda_\omega))$: $\pi \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$

1. Compute $H_\lambda := \frac{P_\lambda}{T}$ and denote $H_\lambda(X) = \sum_{k=0}^{\mu-2} H_{\lambda k} X^k$. That is, $H_{\lambda k}$ is the k-th coefficient of H_λ.
2. Compute $g_1^{\frac{H_\lambda(\tau)\cdot T(\tau)}{\delta}} = \prod_{i=0}^{\mu-2} \left(g_1^{\frac{\tau^i \cdot T(\tau)}{\delta}} \right)^{H_{\lambda i}}$.
3. Compute $g_1^{\mathsf{Wtns}} := \prod_{i=\omega_{\mathsf{Stmt}}+1}^{\omega} \left(g_1^{\frac{\beta\mathcal{U}_i(\tau)+\alpha\mathcal{V}_i(\tau)+\mathcal{W}_i(\tau)}{\delta}} \right)^{\lambda_i}$.
4. Pick $\rho, \sigma \xleftarrow{\$} \mathbb{F}_r$.
5. Compute $g_1^A := \left(g_1^\delta\right)^\rho g_1^\alpha \cdot g_1^{\mathcal{U}_0(\tau)} \cdot \prod_{i=1}^{\omega} \left(g_1^{\mathcal{U}_i(\tau)}\right)^{\lambda_i}$, $g_1^{\mathcal{V}} := \left(g_1^\delta\right)^\sigma g_1^\beta \cdot g_1^{\mathcal{V}_0(\tau)} \cdot \prod_{i=1}^{\omega} \left(g_1^{\mathcal{V}_i(\tau)}\right)^{\lambda_i}$,
   $$g_1^C := g_1^{\mathsf{Wtns}} \cdot g_1^{\frac{H_\lambda(\tau)\cdot T(\tau)}{\delta}} \cdot \left(g_1^A\right)^\sigma \cdot \left(g_1^{\mathcal{V}}\right)^\rho \cdot \left(g_1^\delta\right)^{-\sigma\cdot\rho},$$
   $$g_2^B := \left(g_2^\delta\right)^\sigma g_2^\beta \cdot g_2^{\mathcal{V}_0(\tau)} \cdot \prod_{i=1}^{\omega} \left(g_2^{\mathcal{V}_i(\tau)}\right)^{\lambda_i}.$$
6. Return $\pi = (g_1^A, g_1^C, g_2^B)$.

Fig. 6: The Prove algorithm for the Groth16 SNARK [32] with $\mathsf{params} = \{\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e(\cdot,\cdot), \mathbb{F}_r\}$, $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$, assignment $\lambda = (\lambda_{\mathsf{Stmt}}, \lambda_{\mathsf{Wtns}}) = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}}, \lambda_{\omega_{\mathsf{Stmt}}+1}, \ldots, \lambda_{n=\omega_{\mathsf{Stmt}}+\omega_{\mathsf{Wtns}}})$, and $\mathsf{CRS}_{\mathsf{EK}} = (\mathsf{CRS}_{\mathsf{EK}1}, \mathsf{CRS}_{\mathsf{EK}2})$ as described above.

Remark. The Prove algorithm described in Figure 6 uses $\mu + 2 \cdot \omega + \omega_{\mathsf{Wtns}} + 4$ exponentiations in $\mathbb{G}_1$ and $\omega + 1$ exponentiations in $\mathbb{G}_2$. Since, in general, $\mu \ll \omega$, this is - in principle - worse than the Prove algorithm that is initially described by Groth in [32], which requires $3\mu + \omega_{\mathsf{Wtns}} + 4$ exponentiations in $\mathbb{G}_1$ and $\mu + 1$ exponentiations in $\mathbb{G}_2$. However, due to reasons related to the ones described in Footnote 13 and as already remarked in [32], the initial proof algorithm additionally uses more FFT computations that - while asymptotically dominated by the exponentiations - may be more costly in practice. For this reason and for simplicity of presentation, we depicted this modified Prove algorithm that was also already sketched by Groth in [32].

The Verify Algorithm In order to do verify a Groth16 proof $\pi = (\pi_A, \pi_C, \pi_B) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$, the Verify algorithm proceeds as described in Figure 7. This takes $\omega_{\mathsf{Stmt}} + 1$ exponentiation in $\mathbb{G}_1$ and the computation of 4 pairings (which can be reduced to 3 pairings if one includes $e(g_1^\alpha, g_2^\beta)$ in $\mathsf{CRS}_{\mathsf{VK}}$). Since ω_{Stmt} is very small for typical circuits, this linear factor barely affects the runtime of Verify, which is hence often considered a constant-time algorithm.

[14] If the additional CRS elements from Footnote 13 are excluded, $\mathsf{CRS}_{\mathsf{EK}}$ consists of $2\mu + \omega + 3$ elements from $\mathbb{G}_1$ and $3 + \mu$ elements from $\mathbb{G}_2$.

```
$\mathsf{Verify}(\mathsf{params}, \mathsf{QAP}, \mathsf{CRS}_{\mathsf{VK}}, \lambda_{\mathsf{Stmt}} = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}}), \pi = (\pi_A, \pi_C, \pi_B))$: 0,1

 1. Compute $g_1^{\mathsf{Stmt}} := g_1^{\frac{\beta\mathcal{U}_0(\tau)+\alpha\mathcal{V}_0(\tau)+\mathcal{W}_0(\tau)}{\gamma}} \cdot \prod_{i=1}^{\omega_{\mathsf{Stmt}}} \left( g_1^{\frac{\beta\mathcal{U}_i(\tau)+\alpha\mathcal{V}_i(\tau)+\mathcal{W}_i(\tau)}{\gamma}} \right)^{\lambda_i}$.
 2. If $e(\pi_A, \pi_B) = e(g_1^{\alpha}, g_2^{\beta}) \cdot e(g_1^{\mathsf{Stmt}}, g_2^{\gamma}) \cdot e(\pi_C, g_2^{\delta})$, then return 1, else return 0.
```

Fig. 7: The Verify algorithm for the Groth16 SNARK [32] with $\mathsf{params} = \{\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e(\cdot,\cdot), \mathbb{F}_r\}$, $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$, statement values $\lambda_{\mathsf{Stmt}} = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}})$, and $\mathsf{CRS}_{\mathsf{VK}} = (\mathsf{CRS}_{\mathsf{VK}1}, \mathsf{CRS}_{\mathsf{VK}2})$ as described above.

Theorem 1 (Security of the Groth16 Protocol [32]). *Assuming an adversary that only uses a polynomial number of generic group operations and pairings, the Groth16 protocol* (Setup, Prove, Verify) *as described above together with the simulation algorithm* Sim *depicted in Figure 8 is a SNARK for the relation* $R_{\mathsf{QAP}} = \{\lambda = (\lambda_{\mathsf{Stmt}}, \lambda_{\mathsf{Wtns}}) \,|\, \lambda$ *is a satisfying assignment for* $\mathsf{QAP}\}$ *in the sense of Appendix A. That is, it is a succinct non-interactive argument that guarantees perfect completeness, perfect zero-knowledge, and computational knowledge-soundness in the generic bilinear group model.*

```
$\mathsf{Sim}(\mathsf{params}, \mathsf{QAP}, \lambda_{\mathsf{Stmt}} = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}}), \mathsf{st} = (\alpha, \beta, \gamma, \delta, \tau))$: $\pi \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2$

 1. Pick $A, B \xleftarrow{\$} \mathbb{F}_r$.
 2. Compute $C = \frac{AB - \alpha\beta - (\beta\mathcal{U}_0(\tau)+\alpha\mathcal{V}_0(\tau)+\mathcal{W}_0(\tau)) - \sum_{i=1}^{\omega_{\mathsf{Stmt}}} \lambda_i \cdot (\beta\mathcal{U}_i(\tau)+\alpha\mathcal{V}_i(\tau)+\mathcal{W}_i(\tau))}{\delta}$.
 3. Return $\pi = (g_1^A, g_1^C, g_2^B)$.
```

Fig. 8: The Sim algorithm for the Groth16 SNARK [32] with $\mathsf{params} = \{\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, e(\cdot,\cdot), \mathbb{F}_r\}$, $\mathsf{QAP} = \{T, (\mathcal{U}_j, \mathcal{V}_j, \mathcal{W}_j)_{j=0}^{\omega}\}$, statement values $\lambda_{\mathsf{Stmt}} = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}})$, and simulation trapdoor st.

C Trusted Setup and Mitigations

As mentioned above, the security of Groth16 crucially depends on an honestly generated CRS and on the simulation trapdoor st not being known to any malicious party. That is, Groth16 requires a *trusted setup*. Indeed, if $\mathsf{st} = (\alpha, \beta, \gamma, \delta, \tau)$ was known to an adversary A, it could easily compute a proof for any assignment $\lambda_{\mathsf{Stmt}} = (\lambda_1, \ldots, \lambda_{\omega_{\mathsf{Stmt}}})$ to the statement values by running the simulation algorithm Sim as depicted in Figure 8. The resulting proof π will be accepted with probability 1 although no valid assignment to the witness values λ_{Wtns} might exist. That is, knowing st allows for creating fake proofs and, hence, breaking (knowledge) soundness of the Groth16 SNARK.

Moreover, a malicious party running the Setup algorithm could generate a malformed (*subverted*) CRS, which can enable breaking the zero-knowledge property of the Groth16 SNARK. In order to prevent this, methods have been proposed [1,9] that, by adding a few random elements to the CRS, allow the construction of an efficient CRS verification algorithm for ensuring that using it for creating a proof indeed provides the zero-knowledge property. We note that these additional elements are only required for CRS verification and, hence, do not need to be downloaded to generate proof.

However, as shown by Bellare et al. [9], it is impossible to achieve both knowledge and soundness against a subverted CRS simultaneously. In order to mitigate the soundness issue, one can run a multi-party protocol [2, 12, 16, 39] in order to distribute the CRS generation among several parties, which guarantees soundness as long as at least one of these parties is honest (and discards her contribution to the computation of st after executing the setup). That being said, it is impossible to eliminate every trust assumption regarding the setup of Groth16 while maintaining its zero-knowledge property. If this is desired, other GPZKPs, such as [4, 17] could be used, which do not require a trusted CRS generation and are in principle compatible with our constructions, as they are also based on R1CS resp. QAPs. However, they are less efficient in computation and proof size when compared to Groth16. It would be an interesting future work to see how they compare for the circuits for ballot validity that we proposed here.

It is, of course, desirable to minimize trust assumptions for verifiability. However, in practice, one often still has some trust assumptions, e.g., trusted bulletin boards, authentication/registration servers, or a trusted PKI. Moreover, for typical e-voting protocols, one assumes that not all tallying parties are dishonest in order to achieve vote privacy - otherwise, they could jointly decrypt any ciphertext. For these reasons, using the approach of distributed CRS generation for Groth16 SNARKs does not weaken the overall security model commonly used in electronic voting.

D Instantiating Groth16 and Choosing the ElGamal Group $\mathcal{G}$

As the Groth16 protocol is based on pairings, the groups $\mathbb{G}_1, \mathbb{G}_2$ are instantiated as groups over pairing-friendly elliptic curves. In the following, we describe a standard instantiation of $\mathbb{G}_1$ and $\mathbb{G}_2$ and its implication on the choice of the ElGamal group $\mathcal{G}$. Finally, we discuss other existing instantiations of $\mathbb{G}_1, \mathbb{G}_2$ that could be interesting alternatives.

D.1 Instantiation of the Pairing Groups $\mathbb{G}_1$ and $\mathbb{G}_2$

A standard instantiation [34,47] of $\mathbb{G}_1$ and $\mathbb{G}_2$ and, hence, of the Groth16 SNARK, uses the Barreto-Naehrig curve $\mathsf{BN254}$ (which sometimes is called $\mathsf{BN128}$ or $\mathsf{alt\_BN\_128}$). While $\mathsf{BN254}$ was originally intended to offer 128 bits of security, the attacks proposed by Kim and Barbulescu [38] reduced its security level to ~ 100 bits, which is, however, still sufficient for many applications. While we discuss other curve choices below, we chose to stick with $\mathsf{BN254}$ for benchmarking our implementations, as we used the libsnark implementation [47] of Groth16, which does not support other curves.

The curve $E := \mathsf{BN254}$ is defined by the (affine) Weierstrass equation $y^2 = x^3 + 3$ over the field $\mathbb{F}_p$ with p being the 254-bit prime

$$p = 21888242871839275222246405745257275088696311157297823662689037894645226208583.$$

Its order is the 254-bit prime

$$r = 21888242871839275222246405745257275088548364400416034343698204186575808495617,$$

which determines the size of $\mathbb{G}_1 = E(\mathbb{F}_p)$ and $\mathbb{G}_2$ (an order-r subgroup of $E(\mathbb{F}_{p^{12}})$) and hence the scalar field $\mathbb{F}_r$ over which QAPs can be defined in order to be compatible with the Groth16 protocol for these parameters. Over these groups, the Weil pairing and variants can be defined and efficiently computed. In practice, the optimal ate-pairing [5, 49] is used for efficiency reasons.

We note that elements of $\mathbb{G}_1$ are just elliptic curve points over the base field $\mathbb{F}_p$ and hence - when using projective coordinates - can be represented using three elements from $\mathbb{F}_p$. On the other hand, elements from $\mathbb{G}_2$ are elliptic curve points over the field extension $\mathbb{F}_{p^{12}}$, a naive representation of which would require twelve field elements for each coordinate. A more compressed representation of elements from $\mathbb{G}_2$ has been described by Barreto and Naehrig in [8], which reduces the number of required field elements to just 2. Still, representing elements in $\mathbb{G}_2$ requires twice as much space as representing elements from $\mathbb{G}_1$. Concretely, an element of $\mathbb{G}_1$ resp. $\mathbb{G}_2$ requires ~ 100 resp. ~ 200 bytes, which leads to proof sizes of well-below 1 KB for this instantiation.

D.2 Choosing The ElGamal Group $\mathcal{G}$

The above curve choice and the resulting group sizes of the pairing groups $\mathbb{G}_1$, $\mathbb{G}_2$ for instantiating the Groth16 protocol necessitate that the arithmetic circuits, that is, the QAPs, that we consider are defined over the field $\mathbb{F}_r$ of size

$$r = 21888242871839275222246405745257275088548364400416034343698204186575808495617.$$

Moreover, as explained in Section 3.1, we need to use these arithmetic circuits to compute several expensive exponentiations for computing a PVC, respectively, an EEG ciphertext. In other words, elements of and computations within the ElGamal group $\mathcal{G}$ need to be expressed using the arithmetic of $\mathbb{F}_r$. In order to address this efficiently, in Kryvos [35], leveraging results from [40], we used a subgroup of a Montgomery curve $\mathcal{E}$ defined over the base field $\mathbb{F}_r$ - namely, an instantiation of Curve25519 [14]. Such a curve $\mathcal{E}$ defined over the scalar field of another elliptic curve is sometimes called an *embedded curve*. Concretely, the elliptic curve is defined as $\mathcal{E} := \{(x,y) \in \mathbb{F}_r^2 \mid y^2 = x^3 + 126932x^2 + x\}$, which has order

$$|\mathcal{E}| = 2^3 \cdot 2736030358979909402780800718157159386074658810754251464600343418943805806723.$$

Let q be the large prime factor of $|\mathcal{E}|$, i.e.,

$$q := 2736030358979909402780800718157159386074658810754251464600343418943805806723,$$

which is a 251-bit prime. Then, we choose the subgroup $\mathcal{G} \subseteq \mathcal{E}$ of order q as our ElGamal group. By construction, elements of $\mathcal{G}$ are represented using two to three elements from $\mathbb{F}_r$ (depending on whether we represent them in affine or projective coordinates), which sums up to less than $50 - 100$ bytes per group element and, hence, $100 - 200$ bytes per EEG ciphertext. We note that these uncompressed sizes can be further reduced using standard compression methods. Regarding security of $\mathcal{G}$, in [40], the authors report a security level of ~ 125 bits.

D.3 Other Curve Choices

Neither the curve from which $\mathbb{G}_1$ and $\mathbb{G}_2$ are derived (BN254 in our case) nor the Montgomery curve $\mathcal{E}$ from which we derived $\mathcal{G}$ is intrinsic to the construction. However, as illustrated above, the construction of $\mathcal{G}$ depends on $\mathbb{G}_1$ and $\mathbb{G}_2$ in the sense that elements and computations in $\mathcal{G}$ need to be expressible (efficiently) by using $\mathbb{F}_r$-arithmetic, where $\mathbb{F}_r$ is determined by $\mathbb{G}_1$ and $\mathbb{G}_2$. Hence, when using BN254 to instantiate $\mathbb{G}_1$

and $\mathbb{G}_2$, the choices for $\mathcal{G}$ are limited to groups that work well with arithmetic over the field of size

$$r = 21888242871839275222246405745257275088548364400416034343698204186575808495617.$$

We believe that the instantiation of $\mathcal{E}$ from [40] is fairly optimal in this sense. However, other common instantiations for $\mathbb{G}_1$ and $\mathbb{G}_2$ exist, most notably those derived from the Barreto-Lynn-Scott curve BLS12-381 [7,48], which offers a slightly higher security level than BN254. For this particular curve, two embedded curves have been found that potentially allow for efficiently expressing the computation of EEG ciphertexts within circuits over the scalar field of BLS12-381 [34,42]. We did not consider a Groth16-instantiation over BLS12-381 in this work, as we based our implementations on libsnark [47], which only supports BN254. However, other Groth16 implementations, such as circom [10] and gnark [15] do support BLS12-381. Adapting our constructions to BLS12-381 and analyzing their efficiency and security could be interesting future work.