Kryvos: Publicly Tally-Hiding Verifiable E-Voting

Nicolas Huber[1,2], Ralf Kuesters[1,2], Toomas Krips[3], Julian Liedtke[1], Johannes Mueller[4], Daniel Rausch[1], Pascal Reisert[1], and Andreas Vogt[5]

[1] Institute of Information Security, University of Stuttgart {firstname.lastname}@sec.uni-stuttgart.de
[2] Center for Integrated Quantum Science and Technology (IQ^{ST})
[3] University of Tartu toomas.krips@ut.ee
[4] SnT/University of Luxembourg johannes.mueller@uni.lu
[5] University of Applied Sciences and Arts Northwestern Switzerland andreas.vogt@fhnw.ch

Abstract. Elections are an important corner stone of democratic processes. In addition to publishing the final result (e.g., the overall winner), elections typically publish the full tally consisting of all (aggregated) individual votes. This causes several issues, including loss of privacy for both voters and election candidates as well as so-called Italian attacks that allow for easily coercing voters.
Several e-voting systems have been proposed to address these issues by hiding (parts of) the tally. This property is called *tally-hiding*. Existing tally-hiding e-voting systems in the literature aim at hiding (part of) the tally from everyone, including voting authorities, while at the same time offering verifiability, an important and standard feature of modern e-voting systems which allows voters and external observers to check that the published election result indeed corresponds to how voters actually voted. In contrast, real elections often follow a different common practice for hiding the tally: the voting authorities internally compute (and learn) the full tally but publish only the final result (e.g., the winner). This practice, which we coin *publicly tally-hiding*, indeed solves the aforementioned issues for the public, but currently has to sacrifice verifiability due to a lack of practical systems.
In this paper, we close this gap. We formalize the common notion of publicly tally-hiding and propose the first provably secure verifiable e-voting system, called Kryvos, which directly targets publicly tally-hiding elections. We instantiate our system for a wide range of both simple and complex voting methods and various result functions. We provide an extensive evaluation which shows that Kryvos is practical and able to handle a large number of candidates, complex voting methods and result functions. Altogether, Kryvos shows that the concept of publicly tally-hiding offers a new trade-off between privacy and efficiency that is different from all previous tally-hiding systems and which allows for a radically new protocol design resulting in a practical e-voting system.

1 Introduction

Elections are an import corner stone of democratic processes. Besides national or local political elections, elections also often are and have to be carried out in companies, organizations, associations, etc. There exists a multitude of different voting methods ranging from relatively simple voting methods, such as plurality/single-choice voting, to more complicated ones, such as cumulative voting with multiple votes, and very complex ones, like preferential elections and multi-round votings. In practice, not just simple voting methods are used but often also more complex ones as they allow for capturing the voters' preferences more accurately. For example, Instant-Runoff Voting (IRV) [1] is an important preferential voting method that is used in many countries for municipal or national political elections, including Australia, India, the UK, and the US [2, 3, 4, 5]. Also, there are many different result functions used in elections. For example, one might only be interested in the winner of the election (e.g., for presidential elections) or the n best or worst candidates (ranked or not ranked), e.g., to fill positions or decide who moves on to a runoff election.

Verifiability. A fundamental security property of elections is *verifiability* [6, 7, 8, 9, 10, 11, 12, 13, 14, 15], i.e., voters should be able to verify that their votes were actually counted and every observer, including voters, election officials, and external observers, should be able to verify whether the final election outcome indeed corresponds to the votes submitted by the voters. This property is not just required for traditional paper-based elections, where votes are typically tallied by several (mutually distrusting) talliers and observers can

monitor the tallying process, but it is particularly crucial for electronic voting (e-voting) systems: numerous problems with e-voting systems have been reported in many countries, where votes have been dropped or miscounted (see, e.g., [16, 17, 18, 19] and references therein). This does not come as a surprise since e-voting systems are among the most complex hardware and software systems. Hence, it is virtually impossible to avoid programming errors or subtle security vulnerabilities. Verifiability enables external and internal observers to detect and reject false election results, even when the underlying cause is an unknown programming error or vulnerability. Verifiability therefore has been studied intensively for e-voting systems (see, e.g., [20] for an overview), with Helios [21] being one of the most prominent verifiable e-voting systems used in practice.

Publishing the Full Tally. In purely paper-based elections and tallying processes, in order to obtain verifiability it appears unavoidable to publicly reveal the full tally consisting of all (aggregated) individual votes, rather than just the actual election result, such as the winner of the election or the n best candidates: arbitrary observers (both voters and external observers) should be able to verify the election outcome, and hence, they have to watch the tallying process and by this learn the full tally. While this is not strictly necessary for e-voting systems, currently most verifiable e-voting systems also have to publish the full tally by design, including, for example, the prominent Helios system. This, however, comes with several downsides:

- *Biased voters*: As mentioned above, some elections consist of several rounds. In particular, they might involve runoff elections. Revealing the intermediate tallies for the candidates during such an election introduces biases that are likely to influence voters' choices for the following rounds, which might be unintended.
- *Embarrassed candidates*: In some elections, for example, within companies or associations, it is unnecessarily embarrassing for the losing candidates to publish the (possibly low) number of votes they received.
- *Weak mandates*: If the winning margin is small, then revealing the full tally can undermine the power of the elected candidate. This might be undesirable.
- *Gerrymandering*: If the distribution of votes in different districts/groups is published, then this information can be used to adjust the specifications of those districts/groups for future elections so that one of the parties has an advantage.
- *Italian attacks*: In preferential voting (e.g., IRV), the individual choices of voters can be essentially unique and thus be regarded as "fingerprints", even if the number of candidates is moderate. This privacy loss can be exploited to perform so-called *Italian attacks* [22, 23]: even a passive adversary, who merely sees the full tally, is able to easily coerce any voter to vote in a specific (unlikely) way and then use the tally to check whether the voter complied.

These issues can be mitigated or resolved by hiding (part of) the tally, except for the election result. Voting systems with this property are called *tally-hiding*. We classify them as follows, depending on their specific objectives and approaches.

Fully and Partially Tally-Hiding. Several (verifiable) e-voting systems, see, e.g., [24, 25, 26, 27, 28, 29], have been proposed to address all of the aforementioned issues at once. They only publish the actual election result (e.g., only the name of the winner or the names of the n best candidates), and no party, neither internal nor external, gets to know any intermediate results. We call this strong notion of tally-hiding *fully tally-hiding*. They typically employ heavy-weight cryptography, such as universally verifiable Multi-Party Computation (MPC). As a result, existing fully tally-hiding e-voting systems are limited to rather simple voting methods and elections with only a few candidates. They quickly become inefficient for larger numbers of candidates or complex result functions and voting methods, such as ranked voting, say, via IRV (see Section 6 for more discussion).

There are also verifiable e-voting systems that focus on specific issues due to publishing the tally, most notably Italian attacks. They hide exactly those parts of the tally which cause the respective issues (see, e.g., [30, 22, 23, 1, 31]). We call this approach *partially tally-hiding*. Such systems can be more efficient than fully tally-hiding ones. For example, the e-voting system proposed in [1] can handle even complex IRV elections for real world data. But hiding the tally partially comes with the trade-off that some information is still revealed, such as the order of candidates, and hence, some of the problems of publishing the tally remain, such as biased voters and embarrassed candidates (see also Section 6). Thus, these systems are useful to solve certain aspects, but not others.

Publicly Tally-Hiding. In this work, we follow an approach that offers a trade-off between privacy and efficiency which is different from all previous fully and partially tally-hiding solutions. This approach is motivated by and enhances existing practices: There have been many cases where voting authorities chose to internally compute but not publish the full tally. They rather only published the final election result, e.g., the winner of the election or the n best candidates, as they wanted to mitigate some of the above issues, including privacy issues for voters (Italian attacks), for candidates (embarrassment, weak mandates), and/or manipulations (gerrymandering). Hence, while the talliers learn the full tally, the public learns only the election result. We call this approach *publicly tally-hiding* elections. Such publicly tally-hiding elections are carried out, among others, by ACM Special Interest Groups (SIG) [32], the German Computer Science Association [33], CrossRef [34], the Society for Industrial and Applied Mathematics (SIAM) [35], or the German Research Fund [36], as confirmed by websites and/or personal communication. Civica Election Services (CES), a large e-voting provider, conducts several dozen elections per year where costumers demand to obtain only the actual election result from CES [37], according to CES, for example, to protect against weak mandates or gerrymandering issues. So while publicly tally-hiding elections are quite common, they, however, so far do not offer verifiability, i.e., it is impossible for a voter or external observer to verify that the election result was computed correctly.

In this work, we close this gap by proposing the first *verifiable* voting system that follows the common practice of publicly tally-hiding elections. More specifically, we propose a publicly tally-hiding e-voting system which follows a radically different approach than all previous tally-hiding ones. We allow each tallier to learn the homomorphically aggregated votes, while the public merely learns the final election result.[6] This allows for a completely different design of the e-voting system and the use of different cryptographic techniques compared to previous systems for tally hiding elections. For example, we can employ relatively lightweight zero-knowledge proofs as opposed to more heavy-weight universally verifiable MPC. Our publicly tally-hiding e-voting system achieves practical efficiency for dramatically larger numbers of candidates and more complex voting methods than all previous fully tally-hiding systems, while still hiding the full tally from the public, unlike partially tally-hiding systems. This of course comes with the trade-off that now the talliers learn the aggregated votes. Altogether, our paper opens a new line of research that allows for enhancing already existing practices, where voting authorities compute but do not publish the tally, with the fundamental and crucial property of verifiability.

Contributions.

- We formalize the notion of *publicly tally-hiding*.
- We propose Kryvos, the first provably secure verifiable e-voting system that explicitly targets the common practice of publicly tally-hiding elections, with the advantages mentioned before. Our system is designed in a completely different way than previous e-voting systems for (fully) tally-hiding. It builds on general purpose zero-knowledge proofs, more specifically, the highly efficient Groth16 SNARKs [38]. The core idea is that talliers prove the correctness of the election outcome according to the result function using these SNARKs. While the idea itself is simple, putting this idea to practice is non-trivial:
  - Kryvos follows the general design philosophy of Helios, one of the most prominent and practical verifiable e-voting systems which forms the basis of many other systems. Unfortunately, it is not possible to apply the above idea directly to Helios (cf. Section 2.2). Therefore, Kryvos is a fundamental redesign of Helios.
  - To be of practical use, also with performance that improves upon existing fully tally-hiding systems, the design of Kryvos requires care and dealing with a number of pitfalls (see Section 2 and Section 4). We carefully evaluate various implementation and design options to come up with a suitable implementation.
- We provide instantiations of Kryvos for various voting methods, including plurality voting, various forms of cumulative elections with multiple votes, and ranked elections, such as IRV. These instantiations come with extensive evaluation and benchmarks, demonstrating the practicality of the system. Among others, we

[6] This aggregated tally might still reveal individual votes to the talliers, depending on how ballots are encoded and aggregations are performed. For simple voting methods, the aggregated tally effectively hides individual votes from the talliers, but it might not for complex voting methods, such as IRV.

test our system with real-world data of complex IRV elections from Australia. Our benchmarks show that Kryvos can handle all of these voting methods more efficiently than existing fully tally-hiding solutions.

Structure. In Section 2, we sketch our design rationale, discuss various design choices, and derive efficient realizations of core building blocks. We then, in Section 3, introduce Kryvos, with its implementation and detailed evaluation presented in Section 4. In Section 5, security and privacy of Kryvos is formally stated and proven. Related work is discussed in Section 6. We conclude in Section 7. Full details and proofs are given in the appendix, with the implementation provided in [39].

2 Design Rationale

Many verifiable e-voting systems follow the concept of the prominent Helios system [21]. Helios is based on a (t,n_t)-threshold IND-CPA-secure additively homomorphic public-key encryption scheme, such as exponential ElGamal. Essentially, given the encrypted votes of the voters, these encrypted votes are homomorphically aggregated to compute the publicly known encrypted tally $\mathsf{Enc}(\mathbb{T})$ consisting of a sequence of ciphertexts containing the total number of votes for each candidate/choice, i.e., there is one ciphertext per candidate/choice. By aggregating votes, Helios breaks the link between voters and their votes, thereby achieving ballot privacy, and additionally hides individual votes within the tally $\mathbb{T}$. The talliers then decrypt $\mathsf{Enc}(\mathbb{T})$, i.e., all ciphertexts therein, in a distributed way and publish $\mathbb{T}$ along with proofs of correct decryption, which reveals the full tally $\mathbb{T}$ and allows everyone to publicly compute the result of the election $f_{\text{res}}(\mathbb{T})$, e.g., the winner of the election. In systems which aim for full tally-hiding, i.e., where only $f_{\text{res}}(\mathbb{T})$, but not $\mathbb{T}$ is revealed, talliers essentially perform certain forms of MPC on $\mathsf{Enc}(\mathbb{T})$ in order to compute $f_{\text{res}}(\mathbb{T})$. As already mentioned and further discussed in Section 6, this has several shortcomings, which is why in this work we, for the first time, directly follow the common approach of public tally-hiding, i.e., while talliers may learn $\mathbb{T}$, everybody else should only learn $f_{\text{res}}(\mathbb{T})$.

Intuitively, our idea for Kryvos is to build a system similar to Helios where encrypted votes can still be publicly aggregated and the talliers can then compute $\mathbb{T}$ from $\mathsf{Enc}(\mathbb{T})$. The aggregation guarantees that talliers do not learn more about individual votes than talliers in a regular privacy-preserving (non tally-hiding) system, such as Helios. However, instead of publishing $\mathbb{T}$, talliers rather internally compute and then publish only the result $f_{\text{res}}(\mathbb{T})$. To still obtain verifiability in this situation, we employ non-interactive zero-knowledge proofs (NIZKPs) to let (one of) the talliers show that $f_{\text{res}}(\mathbb{T})$ was computed correctly from $\mathsf{Enc}(\mathbb{T})$. Since NIZKPs are rather lightweight compared to MPC, the hope and rationale is that this should allow for constructing more efficient e-voting systems that support more kinds of elections in a (publicly) tally-hiding way than existing (full) tally-hiding systems. To support a multitude of existing election types with different result functions, we make use of recent advances in the field of *general purpose zero knowledge proof systems* (cf. Section 2.1), which allow for proving statements with respect to essentially arbitrary functions. While this general idea might seem conceptually simple, it requires careful design and optimization to achieve not only a secure but also a practical solution as outlined in this section and Section 4.

2.1 General Purpose Proof Systems

A general purpose proof system takes as input an arbitrary indicator function $f_R : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ for some binary relation R, such that $f_R(x,w) = 1$ iff $(x,w) \in R$ for public statement x and secret knowledge/*witness* w. It then allows for computing a zero-knowledge proof that proves knowledge of w such that $f_R(x,w) = 1$. In our setting, a tallier can use $x = (\mathsf{Enc}(\mathbb{T}), y)$ as public input along with a suitable witness (which, among others, contains some tally $\mathbb{T}_w$) and the function f_R: $f_R(x,w) = 1$ iff $\mathbb{T}_w$ corresponds to the plaintext in $\mathsf{Enc}(\mathbb{T})$ and $y = f_{\text{res}}(\mathbb{T}_w)$. The chosen proof system should be able to handle complex result functions even for a large number of candidates/choices, which, among others, requires reasonably low proof creation and verification times. Of the various different general purpose proof systems [40, 41, 42, 43, 44, 38, 45], zero-knowledge succinct non-interactive arguments of knowledge (SNARKs, for short, where we implicitly assume the zero-knowledge property) currently fit these requirements best. More specifically,

we use the highly efficient state-of-the-art Groth16 SNARK [38] as, offers constant proof size with constant verification time (independently of the function f_R), while achieving quite fast polynomial proving time and thus scaling well even for highly complex functions. We briefly discuss our choice of the Groth16 SNARK in comparison with alternative proof systems in Section 6. We note that our construction is based on arithmetic circuits (see next paragraph) and hence is not limited to the Groth16 SNARK. In fact, every proof system which is based on arithmetic circuits (such as [46, 47, 48]) can be used to instantiate Kryvos.

A Groth16 SNARK uses the language of quadratic arithmetic programs (QAPs) to specify the underlying relation R respectively indicator function f_R. Typically, in order to obtain a QAP, f_R is first expressed as an arithmetic circuit which is then converted to a set of constraints that is finally compiled into a QAP instance. A constraint over n variables is an equation $\sum_{i=1}^{n} a_i u_i \cdot \sum_{i=1}^{n} a_i v_i = \sum_{i=1}^{n} a_i w_i$, where u_i, v_i and w_i are constants that define the constraint. The overall performance (runtime, bandwidth, memory overhead) of a Groth16 SNARK is directly determined by the number of constraints and hence the size of the arithmetic circuit.

A bit simplified, the Groth16 SNARK consists of three algorithms: (Setup,Prove,Verify). The $\mathsf{Setup}(f_R)$ algorithm generates two common reference strings, $\mathsf{CRS}_{\mathsf{EK}}$ (*evaluation key* CRS, needed to create proofs) and $\mathsf{CRS}_{\mathsf{VK}}$ (*verification key* CRS, needed to verify proofs) that depend on f_R. This creates an instance of Groth16 that is specific to the function f_R. The $\mathsf{CRS}_{\mathsf{EK}}$ can then be used by anyone to create a proof $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{CRS}_{\mathsf{EK}}, x, w)$ with properties as described above. One can use $\mathsf{Verify}(\mathsf{CRS}_{\mathsf{VK}}, x, \pi)$ to verify the proof, which requires only a very small $\mathsf{CRS}_{\mathsf{VK}}$. We note that Groth16 SNARKs are based on bilinear groups whose order depends on the size of the input values. The currently most efficient implementation [49], which we use, uses bilinear groups of size up to 256 bits and supports operations in $\mathbb{F}_p^*$ with p of size up to 255 bits.

2.2 Public Tally-Hiding for Systems With Homomorphic Encryption

Our approach of obtaining a verifiable publicly tally-hiding e-voting system by having talliers prove in ZK that $f_{\text{res}}(\mathbb{T})$ was computed correctly from $\mathsf{Enc}(\mathbb{T})$ does not directly apply to all existing e-voting systems, which often use homomorphic encryption for tallying votes. For example, simply modifying Helios by letting talliers publish only $f_{\text{res}}(\mathbb{T})$ plus a ZKP showing correctness of the result, instead of publishing $\mathbb{T}$, does not yield a secure system. Such a ZKP would have to be computed locally by one of the talliers, who would require as witnesses all private key shares in order to prove knowledge of $\mathbb{T}_w$ such that $\mathbb{T}_w = \mathsf{Dec}(\mathsf{Enc}(\mathbb{T}))$. Revealing those key shares would in turn allow that tallier to decrypt individual votes and hence break ballot privacy entirely. One interesting option to tackle this problem might be to use the very recent concept of distributed SNARKs which permit several parties, each holding a share of a witness, to compute a SNARK in a distributed and privacy preserving fashion [50, 51]. However, to protect the secrecy of witness shares, these constructions heavily rely on MPC components, so it is unclear whether and in how far this approach would yield voting systems that are more efficient than fully tally-hiding ones. Indeed, benchmarks by [51] indicate that currently available distributed SNARKs are still too slow for our purposes.

In this work, we therefore propose a different approach, namely, constructing an e-voting system based on an additively homomorphic commitment scheme. This follows a line of commitment-based e-voting systems initiated by [52]. In Section 3 we show that using this approach it is possible to build a publicly tally-hiding system such that a single tallier is able to locally and thus efficiently compute a SNARK showing correctness of the result.

2.3 Efficiently Proving Knowledge of the Tally

A SNARK can in principle be used to prove statements for arbitrary NP-relations, however, the resulting performance (in terms of runtime, memory overhead, bandwidth, etc.) quickly deteriorates and becomes impractical for large circuits such as, e.g., cryptographic algorithms (see, e.g., [53, 54]). Therefore, a main challenge in using SNARKs consists of carefully constructing suitable circuits with minimal numbers of constrains for the intended relations such that the resulting SNARKs are practical. While we discuss the overall circuits for our SNARKs in Section 4, we need as a central building block for all of our SNARKs an efficient circuit for verifying decommitments, which we construct in the following.

Commitment Scheme. We use Pedersen commitments over a group $(\mathcal{G}, \cdot)$ of order q with generator g as introduced in [55]. Recall that to compute a Pedersen commitment, one first takes an auxiliary value h defined by $h = g^a$ for a uniform $a \xleftarrow{\$} \mathbb{F}_q$ and then commits to a value $v \in \mathbb{F}_q$ with $\mathsf{Com}(v,r) = g^v h^r$ using a uniform $r \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$. Pedersen commitments are a standard solution that have a simple structure and hence also rather small computational complexity.

The choice of the group $\mathcal{G}$ significantly affects efficiency of the corresponding SNARK, with some groups being more favorable than others. We use a Montgomery elliptic curve, specifically Curve25591, which is well known to provide good synergy with Groth16 SNARKs. Specifically, coordinates of points on this curve use only 255 bit as provided by [49]. Furthermore, the Montgomery ladder algorithm allows for relatively cheap exponentiation within the SNARK as discussed next.[7]

Designing the Circuit. A Pedersen commitment $g^v \cdot h^r$ requires exponentiations and multiplications, with exponentiations being the most expensive operation. We leverage results from the C∅C∅-framework [53], which observes that the highly efficient Montgomery ladder algorithm for exponentiations can be implemented with relatively few constraints. However, a major limitation of this approach is that the resulting values cannot directly be multiplied.

We therefore extend this approach to also support efficient multiplication of such values. Among others, we achieve this by designing small circuits for the fast y-coordinate recovery algorithm by Okeya and Sakurai [56], conversion between projective and affine coordinates, and point multiplication. We construct the overall circuit for computing a Pedersen commitment by combining these components, where combinations are chosen in such a way that they require the least number of constraints. We provide full details of our circuit, including a discussion of choices, in Appendix A.

Optimizing Constraints. A crucial and labor intensive step for obtaining an efficient SNARK is to optimize the implementation via constraints (see e.g., [53, 57]). While one can in principle, specify an arithmetic circuit defined in a suitable language and then automatically compute a representation via constraints, such a generic conversion results in needlessly large numbers of constraints and can lead to impractical SNARKs. Obtaining efficient SNARK proofs is crucial for our voting system. We therefore manually implemented and optimized our circuit via constraints. This includes individual gates and operations such as various comparison and branching operations, elliptic curve point conversions, and multiplications as well as combinations of those gates within specific algorithms, like the Montgomery ladder and the Okea-Sakurai algorithm, and within the overall commitment circuit.

As an example of the many optimizations that we have performed — all of which, even if small, add up quickly — consider a division gate. For divisions, one has to handle the special case of a divisor of value zero. Such cases can occur while evaluating a path not taken by the control flow during branching. Using multiple such divisions is costly. However, it is possible to optimize the number of constraints for the divisions by reusing intermediate results for specific values for multiple divisions. With this, one only needs to perform the error handling once. The same technique can be applied for various other error handling cases, for example for dealing with the point at infinity. Using knowledge of the nesting of the gates allows for even more efficient error handling. For example, if the output of a gate is an elliptic curve point that cannot be the point at infinity, independently of the input values, no such checks have to be performed for the following gate.

We have implemented these optimizations resulting in a library with optimized representations via constraints of many fundamental elliptic curve operations, available at [39]. This library should hence be useful also beyond our work, e.g., to obtain an efficient QAP instance of ElGamal.

Further Optimizing The Commitment Circuit. Even with the Montgomery ladder and an optimized QAP representation, the exponentiation step is still very expensive. In our and similar settings, we can further improve this step via two additional optimizations:

[7] To stay compatible with the usual notation for Pedersen commitments, we denote the group operation on an elliptic curve with $\cdot$ instead of the usual $+$. The wording *addition* and *multiplication* is consequentially changed to *multiplication* and *exponentiation*.

	N Pedersen Commitments				1 Pedersen Vector Commitment			
Choices/ N	Constraints	Prove [s]	$\mathsf{CRS}_{\mathsf{EK}}$ [GB]	$\mathsf{CRS}_{\mathsf{VK}}$ [MB]	Constraints	Prove [s]	$\mathsf{CRS}_{\mathsf{EK}}$ [GB]	$\mathsf{CRS}_{\mathsf{VK}}$ [MB]
1	6,549	0.903	0.01	0.98	6,549	0.903	0.01	0.98
5	32,745	3.472	0.05	4.9	10,077	1.249	0.01	1.5
10	65,490	6.684	0.09	9.8	14,487	1.681	0.02	2.17
25	163,725	13.107	0.18	24	27,717	2.546	0.03	4.15
50	327,450	32.376	0.46	49	49,767	5.142	0.07	7.45
100	654,900	64.491	0.91	98	93,867	9.467	0.13	14
150	982,350	96.606	1.37	147	137,967	13.792	0.19	20
200	1,309,800	128.722	1.82	196	182,067	18.117	0.25	27
250	1,637,250	160.837	2.28	245	226,167	22.442	0.31	34

Table 1: Comparison of Pedersen commitments and Pedersen vector commitments in a single Groth16 SNARK. We consider inputs v of at most 32 bits.

Firstly, we can upper bound the length of the input v of the commitment scheme with 32 bits (instead of the full 255 bits), which is sufficient for our application. We discuss the effects of this optimization in Appendix A.

Secondly, as part of our system, we will need to commit to multiple input values v_i, say, N many. If one implements this in a straightforward manner via standard Pedersen commitments, then there will be N separate commitments $\mathsf{com}(v_i,r_i)$, one per v_i. Each of these commitments introduces another exponentiation for some randomness r_i to obtain h^{r_i}, which becomes very costly for large N (cf. left side of Table 1). We solve this issue by using Pedersen *vector* commitments [58]. Essentially, these commitments allow for committing to a vector $\boldsymbol{v} = (v_1, \ldots, v_N)$ with entries in $\mathbb{F}_q$ by computing $\mathsf{Com}(\boldsymbol{v},r) = g_1^{v_1} \cdot \ldots \cdot g_N^{v_N} \cdot h^r$ for generators $g_1, \ldots, g_N$ of $\mathcal{G}$ chosen uniformly at random. In this case, we say that N is the *slot size* of the commitment. One observes that Pedersen vector commitments are also additively homomorphic, perfectly hiding, and computationally binding under the discrete logarithm assumption [59]. This construction requires just one exponentiation with randomness of 255 bits to commit to all N inputs. This in turn drastically improves both the computational cost but also the sizes of the CRSs of the SNARK as shown in Table 1. We will further discuss the optimal choice of N in the context of our voting scheme in Section 4, with Appendix D further illustrating possible tradeoffs.

3 The Kryvos System

In this section, we present the Kryvos e-voting system, which is the first verifiable e-voting system which directly follows the publicly tally-hiding paradigm. Kryvos is a generic framework that supports many different voting methods and result functions. We present its implementation and benchmarks for specific voting methods and result functions in Section 4.

3.1 System description

We now describe the Kryvos e-voting system. Our description focuses on those parts of Kryvos that are at the core of the public tally-hiding property. Well-known and standard enhancements to security, in particular distributed implementations of bulletin boards (e.g., [60, 61, 62]) and mechanisms to mitigate trust on the voting devices (e.g., [63, 64]), are orthogonal to and fully compatible with Kryvos.

Participants. The Kryvos protocol is run among a voting authority Auth, voters $\mathsf{V}_1, \ldots, \mathsf{V}_{n_v}$, talliers $\mathsf{T}_1, \ldots, \mathsf{T}_{n_t}$, and a bulletin board (BB). We assume that for each party there exists a mutually authenticated channel to the BB.

Voting method. Kryvos is parameterized by $n_{\mathsf{choices}} \in \mathbb{N}$ denoting the number of choices of an election and a set of valid voting options $C \subseteq (\mathbb{F}_q)^{n_{\mathsf{choices}}}$, the *choice space* over the prime field $\mathbb{F}_q$. Both parameters are instantiated depending on the specific voting method that should be captured. For example, for a single-vote

election, where each voter can give a single vote to one out of three candidates, we have $n_{\mathsf{choices}} = 3$ and the choice space $C_{\mathsf{single}} = \{(a,b,c) \mid a,b,c \in \{0,1\}, a+b+c = 1\}$.

Kryvos is further parameterized by a deterministic polynomial time function $f_{\text{res}} \colon \{0,1\}^* \to \{0,1\}^*$ which computes the *final result* of the election based on a set of aggregated votes, i.e., the tally of the election. For example, if we are interested only in the winner/winners of the election, then f_{res} returns the index/indices of the highest entry/entries. The tuple (C, f_{res}) defines the abstract voting method for which we describe Kryvos; specific instantiations are given in Section 4.

Setup phase. In this phase, all election parameters are fixed and posted on the BB by Auth: the list **id** of eligible voters, opening and closing times, the election ID $\mathsf{id}_{\text{election}}$, and the voting method (C, f_{res}). Additionally, Auth publishes a decomposition $n_{\mathsf{choices}} = n_{\mathsf{tuples}} \cdot N$ that induces a decomposition on the choice space $C \subseteq (\mathbb{F}_q)^{n_{\mathsf{choices}}} = (\mathbb{F}_q)^N \times \ldots \times (\mathbb{F}_q)^N$ yielding a splitting of a vote $v \in C$ into n_{tuples} tuples, each of size N. The reason for this splitting lies in the usage of homomorphic vector commitments as introduced in Section 2. Splitting allows for decomposing a commitment to a vote v into n_{tuples} vector commitments with N slots. We will argue in Section 4 that this decomposition can be necessary for very large values of n_{choices}. A vote $v = (v_1, \ldots, v_{n_{\mathsf{tuples}}})$ then contains in v_1 the respective entries for the first N choices, in v_2 the next N choices and so on.[8] This assignment is published by Auth on B. Furthermore, Auth creates and publishes the CRSs required for the Groth16 SNARK proofs on B. Each tallier T_k runs the key generation algorithm $\mathsf{KeyGen}_{\mathcal{E}}$ of an IND-CCA2-secure public-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Enc}, \mathsf{Dec})$ to generate its public/private (encryption/decryption) key pair $(\mathsf{pk}_k, \mathsf{sk}_k)$ and posts (k, pk_k) on B.

Voting phase. Let $m^i = (m^{i,1}, \ldots, m^{i,n_{\mathsf{choices}}}) \in C$ be the vote of voter V_i. The voter creates full-threshold secret sharings of every component $m^{i,j}$ of her vote to share $m^{i,j}$ among the n_t talliers:

$$(m_1^{i,j}, \ldots, m_{n_t}^{i,j}) \text{ with } \sum_{k=1}^{n_t} m_k^{i,j} \bmod q = m^{i,j}.$$

Then, for each tallier T_k, the voter decomposes $(m_k^{i,1}, \ldots, m_k^{i,n_{\mathsf{choices}}})$ into $(t_k^{i,1}, \ldots, t_k^{i,n_{\mathsf{tuples}}})$, where $n_{\mathsf{choices}} = n_{\mathsf{tuples}} \cdot N$ is as defined by Auth in the setup phase (see above). Now the voter creates a commitment to each tuple:

$$c_k^{i,l} \leftarrow \mathsf{Com}(t_k^{i,l}; r_k^{i,l}).$$

Observe that since the commitment scheme is additively homomorphic, the commitment $c^{i,l} := \sum_{k=1}^{n_t} c_k^{i,l}$ opens to the tuple $(m^{i,1+(l-1)\cdot N}, \ldots, m^{i,l\cdot N})$, i.e., the votes for the choices belonging to the l-th factor in the decomposition of $(\mathbb{F}_q)^{n_{\mathsf{choices}}}$ specified by Auth, using opening value $r^{i,l} := \sum_{k=1}^{n_t} r_k^{i,l}$. That is, one can obtain commitments on the original vote of V_i by combining all of the commitments on the full-threshold shares.

To guarantee the well-formedness of all commitments and the vote contained therein, the voter creates a Groth16 SNARK proof Π^i_{ballot} proving that $(c^{i,1}, \cdots, c^{i,n_{\mathsf{tuples}}})$ commits to a vote $m^i \in C$; we provide more information, including formal definitions of the relations shown by the proof, in Appendix B.

For each tallier T_k, the voter uses T_k's public key pk_k to securely send the opening values for $(c_k^{i,1}, \ldots, c_k^{i,n_{\mathsf{tuples}}})$ to T_k:

$$e_k^i \leftarrow \mathsf{Enc}(\mathsf{pk}_k, ((t_k^{i,1}, \ldots, t_k^{i,n_{\mathsf{choices}}}), (r_k^{i,1}, \ldots, r_k^{i,n_{\mathsf{tuples}}}))).$$[9]

To complete the voting process, V_i submits her ballot $\mathsf{b}^i = (i, (c_k^{i,l})_{l,k}, \Pi^i_{\mathsf{ballot}}, (e_k^i)_k)$ to the bulletin board. B then verifies that i) the voter is eligible to vote, ii) has not submitted a valid ballot before, iii) the voter's SNARK proof is valid, and iv) no voter has previously submitted a vector containing any of the ciphertexts in $(e_k^i)_k$.[10] If all checks succeed, then the BB adds b^i to the list of ballots **b** and publicly updates **b**.

[8] If n_{choices} is not divisible by N, we can artificially grow the choice space so that n_{choices} is a multiple of N by appending zeros to a vote.

[9] We assume that all plaintexts have fixed length. This can easily be guaranteed here. This is necessary to provide privacy for the votes.

[10] This so-called *ballot weeding* process prevents malicious voters from submitting ballots that are related to the ballots from honest voters, which would break privacy. In particular, due to the CCA2-secure encryption, which

Now, the list $\mathbf{b}$ needs to be prepared by the talliers for the tallying phase. Each tallier T_k decrypts every e_k^i posted on B:

$$((\tilde{t}_k^{i,1}, \ldots, \tilde{t}_k^{i,n_{\mathsf{tuples}}}), (\tilde{r}_k^{i,1}, \ldots, \tilde{r}_k^{i,n_{\mathsf{tuples}}})) \leftarrow \mathsf{Dec}(\mathsf{sk}_k, e_k^i).$$

Then T_k checks whether each pair $(\tilde{t}_k^{i,l}, \tilde{r}_k^{i,l})$ is a valid opening for $c_k^{i,l}$. If this is not the case, then T_k publishes a NIZKP of correct decryption of e_k^i on B so that one can verify that e_k^i is invalid; as a consequence, the corresponding ballot will not be counted.[11] All of the following steps, including the tallying phase, are performed only for those ballots that have passed this check; for simplicity of presentation, we therefore assume that all voters have submitted a valid ballot.

Tallying phase. Everyone can homomorphically aggregate the public commitments on B by computing

$$c^{\perp,l} \leftarrow \sum_{k=1}^{n_t} \sum_{i=1}^{n_v} c_k^{i,l}$$

for each $1 \leq l \leq n_{\mathsf{tuples}}$. In parallel, the talliers homomorpically aggregate the corresponding opening values. First, each tallier T_k internally computes for each $1 \leq l \leq n_{\mathsf{tuples}}$

$$t_k^{\perp,l} \leftarrow \sum_{i=1}^{n_v} t_k^{i,l}; \quad r_k^{\perp,l} \leftarrow \sum_{i=1}^{n_v} r_k^{i,l},$$

where the $t_k^{i,l}$'s and $r_k^{i,l}$'s are the opening values decrypted above. Next, each tallier T_k shares $(t_k^{\perp,l})_l$ and $(r_k^{\perp,l})_l$ with the other talliers so that for each $1 \leq l \leq n_{\mathsf{tuples}}$, the trustees can internally compute

$$t^{\perp,l} \leftarrow \sum_{k=1}^{n_t} t_k^{\perp,l}; \quad r^{\perp,l} \leftarrow \sum_{k=1}^{n_t} r_k^{\perp,l}.$$

After this step, all talliers can compute the (aggregated) tally of all votes $\mathbb{T} := (m^{\perp,1}, \ldots, m^{\perp,n_{\mathsf{choices}}})$, where $m^{\perp,j}$ is the total number of votes for choice j.

The final step is then performed by a designated tallier. First, this tallier computes the election result $\mathsf{res} \leftarrow f_{\mathrm{res}}(\mathbb{T})$. She then computes a Groth16 SNARK on public inputs $c^{\perp,1}, \ldots, c^{\perp,n_{\mathsf{tuples}}}$ and res that proves knowledge of $\mathbb{T}$ (as well as knowledge of randomness) such that $c^{\perp,1}, \ldots, c^{\perp,n_{\mathsf{tuples}}}$ is a list of commitments on $\mathbb{T}$ and res is the output of $f_{\mathrm{res}}(\mathbb{T})$ (cf. Appendix B.1 for the formal definition of the relation). The result res and the SNARK proof are then published on the BB.

Public verification phase. In this phase, every participant, including the voters or external observers, can verify the correctness of the tallying procedure, in particular, the correctness of all ZKPs. Note in particular that homomorphic aggregations of commitments can be re-computed by external observers without knowing any openings.

3.2 Discussion

Using SNARKs to prove ballot validity. Using a general Groth16 SNARK to prove ballot validity has the advantage that Kryvos can support essentially arbitrary choice spaces and hence voting methods, including very complex ones such as ranked voting methods and methods that assign points from a specific set and according to certain rules to each candidate. As we show in Section 4.2, this approach is indeed practical. This support for arbitrary ballot formats is also of interest beyond the area of (publicly) tally-hiding systems. For example, as far as we are aware Kryvos is the first e-voting system that supports ZKPs

must contain the correct randomness for the commitments, the only way to submit a valid related ballot is by creating a new ballot that re-uses some of the (unmodified) ciphertexts from the honest ballots. This is caught by the ballot weeding procedure if at least one tallier is honest (which is always assumed for privacy).

[11] For example, one can combine the IND-CCA2-secure PKE by Cramer-Shoup [65] with the NIZKP by Camenisch-Shoup [66]. See Appendix B.1 for the precise relation to be proven.

for Borda tournament style ballots. For Condorcet ballots, Kryvos improves over previous efficient ZKPs that even required talliers to perform part of the proof [67].

Kryvos can in principle also be combined with other existing ZKPs for ballot correctness that are designed for a specific choice space, where such ZKPs are available (e.g., [68]).

Generating CRSs. Note that our system requires honestly generated CRSs for the Groth16 SNARKs. There are several well-known mechanisms that are practical with the parameters we need, such as distributed generation of the CRS by multiple parties. We briefly discuss such mechanisms in Appendix C for completeness. For simplicity and since this is an orthogonal issue, we here compute CRSs on a local computer.

4 Implementation of Kryvos and Evaluation

We provide a proof of concept implementation of Kryvos that we have instantiated for a wide range of both simple and complex voting methods as well as various result functions; our implementation is available at [39]. We make use of the libsnark library [49] for the Groth16 SNARK. While in Section 3 we gave a high-level description of Kryvos, there are various ways in how Kryvos can be implemented, involving among others fixing a slot size N, a suitable choice space, designing and optimizing circuits for both ballot and tallying SNARKs. Coming up with an efficient implementation needs care. We follow various implementation paths, carefully evaluate them, and by this develop a practical implementation.

All of the following benchmarks, were obtained using Ubuntu 20.04 with 16 GB of RAM and eight cores. All benchmarks are performed without leveraging parallelism, i.e., we use just a single core to make the results independent of the specific core count, thereby making it easier to compare with other benchmarks. Of course, in practice one would parallelize, e.g., if multiple SNARK proofs need to be computed, thereby reducing the overall runtime depending on the number of cores available.

4.1 The Optimal Slot Size

Recall from Section 3.1 that Kryvos uses Pedersen vector commitments, where the number of inputs (slot size) N can be chosen arbitrarily and in particular independently of the number of choices n_{choices}. Before we present our concrete instantiations, we first discuss and evaluate the choice of N in relationship to n_{choices}.

The slot size N heavily influences the circuit size for proving a correct opening of the commitments, which is also the main determining factor for the size and hence the performance of the overall ballot and tallying SNARKs (cf. in Section 4.2). Hence, the runtime and bandwidth of ballot/tallying SNARKs for $N = 1$ and $N = n_{\mathsf{choices}}$ roughly corresponds to the left and right columns of Table 1, with each row indicating a possible value for n_{choices}. We have evaluated the exact performance of showing an opening of an entire ballot/tally (possibly consisting of several separate commitments) via a SNARK not just for these extremes but also for possible intermediate choices, e.g., having two commitments with $N = \frac{n_{\mathsf{choices}}}{2}$, and provide full results with an in-depth discussion in Appendix D and Figure 6.

To summarize our findings, the combined runtime and bandwidth used for ballot and tallying SNARKs for both proving correctness of a ballot and correctness of SNARKs is minimal for $N = n_{\mathsf{choices}}$. As per Table 1, this yields practical performance up to $n_{\mathsf{choices}} = 250$ choices, which is sufficient for most of our use cases and will therefore be used in what follows. The only exception are IRV elections, where $n_{\mathsf{choices}} \gg 250$ (e.g., $n_{\mathsf{choices}} = 1{,}957$ for 6 candidates). The limiting factor in this setting is the ballot SNARK: For $N = n_{\mathsf{choices}}$, computing the SNARK would require downloading a CRS of size 2.4 GB and computing 170 seconds, which is typically not considered acceptable for a voter. By setting N to instead be a large fraction of n_{choices}, say, $N = \frac{n_{\mathsf{choices}}}{8}$, the overall runtime for proving knowledge of a ballot will be slightly larger *but* one can compute this statement in parallel using multiple much smaller SNARKs. This leads to a smaller CRS and reduces overall runtime to an acceptable level for voters with multi core CPUs. We discuss this in detail in Section 4.3.

4.2 Implementation and Evaluation of Kryvos for various voting methods/result functions

We have instantiated and evaluated Kryvos for a variety of voting systems and result functions. As it turns out, there are essentially two main requirements for a voting method to be supported by Kryvos in practice: Firstly, the voting method must support a ballot format that allows for homomorphic aggregation of ballots. This is necessary to protect the privacy of voters against the talliers. Secondly, it must be possible to define a choice space such that the resulting ballots, including the corresponding ballot SNARKS, can be created efficiently.

We note that a voting method essentially defines a choice space which is in turn required for proving ballot validity. More precisely, proving ballot validity includes proving that a vote belongs to the respective choice space. An election result function, however, does not (only) depend on the voting method and its corresponding choice space. Some result functions only make sense for certain choice spaces though. For systems like Single-Vote and Multi-Vote, we consider various result functions that e.g. compute only the candidate with the most votes or all candidates that gained at least a certain threshold of votes. The chosen result function only influences the tallying SNARK but not the ballot SNARK.

Concretely, we have implemented the following popular and widely used voting methods and result functions for Kryvos with full details provided in Appendix E: *(i)* Single-Vote and Multi-Vote with the result functions Most Votes, Vote Threshold and Best n (unordered), *(ii)* Borda Count (as used in the Eurovision Song Contest [69] and for parliamentary elections in the Republic of Nauru [70]) with the same result functions as Single-Vote and Multi-Vote, *(iii)* Majority Judgement [71] (used for example in political research polling in the US, France and Germany [72]) with result functions to compute a Median Grade and complete Majority Judgement Evaluation, *(iv)* Condorcet Methods (as used for internal elections by the Debian Project [73]) with a result function for the computation of the Smith Set, *(v)* and Instant Runoff-Voting (IRV) (as used used for example in political elections in Australia [2], India [3], the UK [4], and the US [5]) where we implemented the result function used for the New South Wales Legislative Assembly election [2].

In what follows, we give an overview of our benchmarking results. The IRV election method is then discussed in detail in Section 4.3. Full details of the remaining voting methods and result functions, including descriptions of those methods and formal definitions of the corresponding choice spaces, are available in Appendix E.

Tallying for various result functions. The tallying phase mainly consists of aggregating the ballots and computing the final Groth16 SNARK proof that shows correctness of the election result. Aggregation of Pedersen (vector) commitments is very fast and can mostly be performed already during the previous voting phase. Hence, the driving factor and focus of our benchmarks lies in the final Groth16 SNARK creation. For all result functions, the tallying SNARK essentially takes the aggregated commitment(s) on the tally and the election result as public input, the commitment opening(s) as secret input, and then proves that the opening corresponds to the commitment and that the election result was obtained by applying f_{res} to the tally contained in the opening. In Appendix B.1, we provide formal definitions of all relations shown by the SNARKs and NIZKPs (for the encryptions of the ballots) used for our instantiations.

The runtime for computing the resulting tallying SNARKS for all of our result functions, including IRV, is shown in Figure 1, where the runtime scales linearly in the number of choices n_{choices}. Our benchmarks show that SNARK computation and hence the tallying phase of Kryvos is very fast even for large numbers of choices, e.g., almost all voting methods and result functions (green line) require only about 100s for $n_{\mathsf{choices}} = 1000$ choices. Furthermore, we can see that in most cases the result function has only a negligible impact on the overall runtime, which is rather dominated by proving knowledge of a decommitment and hence practically identical in almost all cases. The only exception is the majority judgment voting method with 6 grades and full result evaluation returning a single winner. The algorithm for computing the result is so complex that it has a small but noticeable effect on the overall runtime, as can between seen in Figure 1. Figure 1 also shows the runtime of the tallying phase of two instantiations of Ordinos, the most current fully implemented and provably secure verifiable and *fully tally-hiding* voting system, which illustrates that Kryvos indeed achieves its goal of much better performance over fully tally-hiding systems. We give a more detailed comparison of Kryvos with Ordinos and the system by Canard et al. [28] for Majority Judgement in Section 6.

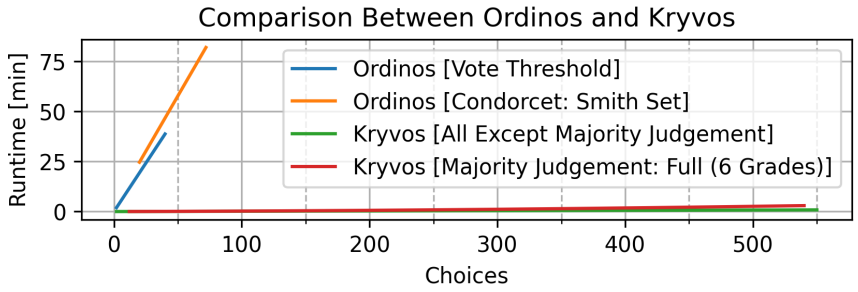

Fig. 1: Benchmarks of all implemented election result functions in comparison to Ordinos. Note that the x-axis uses the number of choices instead of the number of candidates.

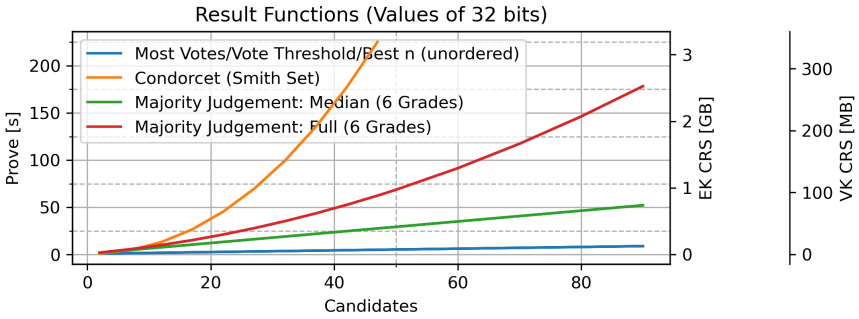

Fig. 2: Benchmarks of Different Election Result Functions.

While in many voting methods, such as single and multi vote elections, a single choice directly corresponds to one candidate, this is not always the case. E.g., the in case of majority judgement with r grades, we need r choices to represent each candidate (essentially, the voter sets one out of r choices to 1 with all other choices being 0 to indicate a grade in such a way that it can be aggregated). In the case of Condorcet voting, which is a ranked voting method where every candidate is compared with every other candidate, the number of choices is even quadratic in the number of candidates. Therefore, Figure 2 translates the runtime results of Figure 1 from numbers of choices n_{choices} to the corresponding number of supported candidates. Additionally, Figure 2 also shows the sizes of the CRSs. As can be seen, in most cases even elections with more than 80 candidates can still be tallied in under 3 minutes. This candidate size should already cover most elections from practice. But our system also scales quite well for much larger numbers. The only exceptions are IRV (not shown in this figure but discussed separately in Section 4.3) and Condorcet (with the Smith set evaluation function, but the same should also apply for other result functions), which scales worse due to the quadratic translation from choices to candidates. We note, however, that Condorcet elections are typically performed only for small to moderate numbers of candidates (less than a dozen) since voters have to fully rank *all* candidates on their ballot, which beyond already 20 candidates becomes very tedious and impractical. So for practical purposes the performance of Kryvos when applied to Condorcet appears to be sufficient.

Ballot validity. To show validity of a ballot (for all voting methods except IRV, which is discussed in Section 4.3), a voter computes a single Groth16 SNARK that takes the single Pedersen vector commitment of slot size $N = n_{\mathsf{choices}}$ obtained by aggregating the commitments on all vote shares of a voter as public input, the opening as secret input, and then proves that the opening is from the correct choice space and corresponds to the commitment. Just as for the tallying SNARK, the runtime for computing a ballot SNARK is dominated by proving knowledge of a decommitment. Hence, runtime is linear in the number of choices n_{choices} and essentially independent of the choice space. In Figure 3, we show the runtime translated to the

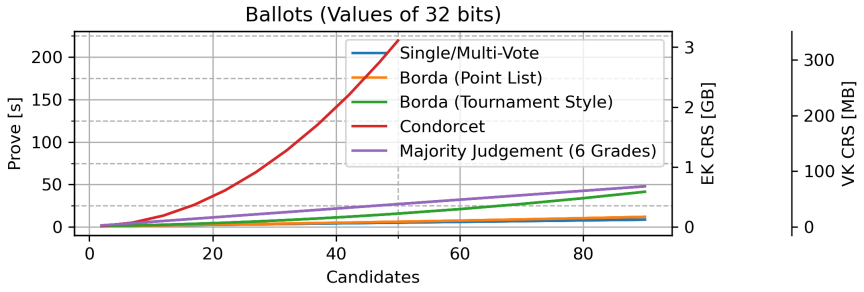

Fig. 3: Benchmarks of SNARKs for proving Ballot validity. Note that the line for Single-Vote coincides with the line for Multi-Vote.

number of candidates as well as the sizes of the CRSs. As this figure shows, voters can efficiently prove the validity of their ballots for all voting methods. Even in the worst case of Condorcet in an election with, say, 20 candidates (which is at the very high end for Condorcet), a voter can prove ballot validity in less than 40 seconds using a $\mathsf{CRS}_{\mathsf{EK}}$ of less than 0.5 GB.

Public Verification Phase. Each Groth16 SNARK proof can be verified in ~ 15 ms using the smaller $\mathsf{CRS}_{\mathsf{VK}}$, where a single proof is of size ~ 5 KB. As an example, consider a single vote election (C_{single}) with an arbitrary number of talliers and up to $n_{\mathsf{choices}} = 1{,}000$ candidates and $n_v = 100{,}000$ voters. Such an election requires 100,000 Groth16 SNARK proofs for showing well-formedness of ballots and a single Groth16 SNARK proof for showing correctness of the election result. In total, the size of all proofs is $\sim$500 MB with a total sequential verification time of $\sim$25 minutes. Verification requires two $\mathsf{CRS}_{\mathsf{VK}}$s, both of them having size less than 150 MB. We note that this verification can be highly parallelized and also already performed while other phases, such as the voting phase, are still running.

4.3 Instantiation of Instant-Runoff Voting

Instant-runoff-voting (IRV) is a quite complex ranked voting method used for example in political elections in Australia [2], India [3], the UK [4], and the US [5]. In IRV, the ballot of a voter contains a (full or partial) ranking of all of the candidates according to her preferences. Tallying of these ballots is then done in multiple rounds: in each round, if a candidate is ranked first by an absolute majority of voters, this candidate immediately wins the election. Otherwise, the candidate who is ranked first by the least number of voters is eliminated. As a result of this elimination, all ballots are edited by removing the eliminated candidate, allowing all of the following candidates to move up in the ranking. This process is then iterated. To handle possible ties, IRV versions include different tie-breaker mechanisms to decide how the round proceeds. One of the more complex versions of IRV is used for the New South Wales Legislative Assembly election [2] (called NSW-IRV in the following), which not only supports partial rankings of candidates but also includes a sophisticated tie-breaking mechanism (cf. Appendix F). We therefore use NSW-IRV for our benchmarks.

For an IRV instantiation of Kryvos, we need to be able to aggregate the tally. This ensures a minimal level of vote privacy towards the talliers and that runtime of the tallying phase is independent of the number of voters. We achieve this by using the choice space C_{single} from the single vote method but interpret each choice as a (full or partial) ranking of candidates. For example, for $n_{\mathsf{cand}} = 5$, we have $n_{\mathsf{choices}} = 120 (= 5!)$ choices for fully ranked IRV, where each choice represents a permutation of the full set of candidates, and $n_{\mathsf{choices}} = 326$ for partially ranked IRV (as in NSW-IRV), where each choice represents a permutation of a partial set of candidates. The result function f_{res} of Kryvos is set to capture the multiple tallying rounds, including modifications to the ballots after each round as well as tie breaker mechanisms of the specific IRV version.

However, this choice space grows exponentially in the number of candidates and thus quickly leads to situations where voters cannot prove validity of their ballots in reasonable time (for the slot size $N = n_{\mathsf{choices}}$).

E.g., while the case of $n_{\mathsf{cand}} = 5$ and hence $n_{\mathsf{choices}} = 326$ is still manageable for voters (the ballot SNARK requires only 30 seconds to compute), for $n_{\mathsf{cand}} = 6$ and hence $n_{\mathsf{choices}} = 1{,}957$ computing the ballot SNARK already requires 3 minutes, which is typically not considered acceptable. One option to try and improve this situation would be to replace the ballot SNARKs with traditional specialized ZKPs that are optimized for the choice space C_{single}. However, as we show in Appendix F, this approach only improves ballot ZKP runtime for very small slot sizes N close to 1, in which case the runtime of the tallying SNARK becomes entirely impractical.

However, we can actually improve the runtime of the ballot SNARK by splitting the ballot into a small number of parts and then showing multiple SNARKs in parallel. Specifically, encoding a ballot via $n_{\mathsf{tuples}} > 1$ separate commitments, each having slot size $N \approx n_{\mathsf{choices}}/n_{\mathsf{tuples}}$, one can show the well-formedness of a ballot belonging to $C_{\mathsf{single}} := \{(x_1, \ldots, x_{n_{\mathsf{choices}}}) \mid x_i \in \{0,1\}, \sum_{i=1}^{n_{\mathsf{choices}}} x_i = 1\}$[12] via multiple sub-statements, where each statement is concerned only with one commitment, namely:

(a) For each commitment $c^{i,j}, 1 \leq j < n_{\mathsf{tuples}}$:
all slots $x_{(j-1)\cdot N+1}, \ldots, x_{j\cdot N}$ in $c^{i,j}$ are either 0 or 1.
(b) For $c^{i,n_{\mathsf{tuples}}}$: the first slots $x_{(n_{\mathsf{tuples}}-1)\cdot N+1}, \ldots, x_{n_{\mathsf{choices}}}$ are either 0 or 1, and all remaining slots $x_{n_{\mathsf{choices}}+1}, \ldots, x_{n_{\mathsf{tuples}}\cdot N}$ are 0.
(c) For $c^i := \sum_{j=1}^{n_{\mathsf{tuples}}} c^{i,j}$: The commitment contains a single slot with value 1 and all other slots are 0.[13]

In Appendix F we describe a mechanism that allows for using single CRS for showing all of the above statements, instead of having three separate CRS each of them with roughly the same size.

Using this method, we have to determine a suitable slot size N (and hence corresponding n_{tuples}). A smaller N allows for higher parallelism up to the number of CPU cores available and hence improves runtime for the ballot SNARK. At the same time, a smaller N increases the combined runtime of all ballot sub-SNARKs (as shown in Section 4.1 but also since (c) introduces an additional decommitment over a new aggregated commitment, which is not necessary if all statements are shown in a single SNARK) and of the single tallying SNARK, which cannot be parallelized. Table 2 shows the benchmarks for various possible slot sizes for an IRV election with 6 candidates. The line "Ballot SNARK: Prove" indicates the runtime for computing a single substatement and hence also indicates the overall runtime if all statements (1 statement for $n_{\mathsf{tuples}} = 1$ and $n_{\mathsf{tuples}} + 1$ statements otherwise due to the addition of (c)) can be computed in parallel. The line "Ballot SNARK: Total Time" gives the combined sequential runtime. As the table shows, even if we use only $n_{\mathsf{tuples}} = 3$ and hence $N \approx n_{\mathsf{choices}}/3$, then a voter with a commonly available quad core CPU can already construct a ballot in less than a minute, which is manageable. The performance progressively improves as a larger number of cores is available, while the tallying SNARK remains almost as efficient as for $n_{\mathsf{tuples}} = 1$, i.e., $N = n_{\mathsf{choices}}$. Hence, using the above construction Kryvos can also handle IRV elections with 6 candidates.

Following the case study of [1], we illustrate the overall performance of Kryvos by using real election data from the 2015 New South Wales state election for the Legislative Assembly [74]. More specifically, we consider the electoral districts of Albury (five candidates) and Auburn (six candidates), which were also target elections for a partially tally-hiding election system proposed in [1]. Our results are given in Table 3, showing that such (complex) elections can be performed in a publicly tally-hiding way using Kryvos. considered.

5 Security

In this section, we formally show that Kryvos enjoys verifiability and privacy properties as desired. For this purpose, we, for the first time, define the notion of publicly tally-hiding e-voting (Section 5.3).

[12] This approach is also applicable to certain other choice spaces.

[13] While (a), (b), and (c) in conjunction prove the well-formedness of a ballot in ZK, they actually do not prove knowledge of a witness. Intuitively, this is because one can re-arrange the components of a ballot with valid proofs to create a new ballot with valid proofs, even without knowing the contents of the commitments. However, Kryvos actually only requires a zero knowledge proof of correctness in order to be secure (cf. Section 5). Alternatively, one could add the position of each commitment as additional public input.

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	3	5	10
Slot Size N	1,957	979	653	392	196
Ballot SNARK: # Constraints	1,741,532	874,046	584,884	353,377	179,525
Ballot SNARK: $\mathsf{CRS}_{\mathsf{EK}}$ [GB]	2.42	1.22	0.81	0.49	0.25
Ballot SNARK: $\mathsf{CRS}_{\mathsf{VK}}$ [MB]	261	131	88	53	27
Ballot SNARK: Prove [s]	171	85	57	34	17
Ballot SNARK: Total Time	171	257	230	209	196
Tallying SNARK: # Constraints	1,748,926	1,755,475	1,762,024	1,774,240	1,802,575
Tallying SNARK: $\mathsf{CRS}_{\mathsf{EK}}$ [GB]	2.43	2.44	2.45	2.47	2.5
Tallying SNARK: $\mathsf{CRS}_{\mathsf{VK}}$ [MB]	262	263	264	266	270
Tallying SNARK: Prove [s]	171	172	173	174	177

Table 2: Comparison of various slot sizes for NSW-IRV with $n_{\mathsf{cand}} = 6$ and thus $n_{\mathsf{choices}} = 1{,}957$.

District	Albury		Auburn	
Number of Candidates	5		6	
Number of Voters	46347		43783	
	Kryvos	[1]	Kryvos	[1]
Tallying	30 s	2h	177 s	15 h
Tallying: SNARK Proof [s]	30		177	
Tallying: SNARK $\mathsf{CRS}_{\mathsf{EK}}$ Size [MB]	450		2,500	
Tallying: SNARK $\mathsf{CRS}_{\mathsf{VK}}$ Size [MB]	47		270	

Table 3: Benchmarks of NSW-IRV. We note that the benchmarks of [1] are taken directly from [1] which uses a setup comparable to the one we used for Kryvos, cf. Appendix F. For Kryvos, we used a slot size of 196.

5.1 Computational Model of Kryvos

We start by formally modeling Kryvos using a general and established computational framework (see, e.g., [75, 76, 20]) that we can use both for analyzing verifiability and privacy of Kryvos.

The computational model introduces the notion of a process which can be used to model protocols (we recall some details in Appendix H). Essentially, a process $\hat{\pi}_{\mathsf{P}}$ modeling some protocol P is a set of interacting ppt Turing machines which capture the honest behavior of protocol participants. The protocol P runs alongside an adversary A, modeled via another process π_{A}, which controls the network and may corrupt protocol participants; here we assume static corruption. We write $\pi{=}(\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}})$ for the combined process.

Kryvos can be modeled in a straightforward way as a protocol $\mathsf{P}_{\mathsf{Kryvos}}(n_v,n_t,C,f_{\mathrm{res}},\mu)$ in the above sense. Recall from Section 3 that we denote the number of voters by n_v, the number of talliers by n_t, and the voting method by (C,f_{res}). By μ we denote a probability distribution over C according to which each honest voter makes her choice. (Note that by this we model that the adversary knows this distribution.) This choice is called the *actual choice* of the voter. Besides the parties mentioned above, Kryvos contains a BB. In our model of Kryvos, the voting authority Auth is part of an additional agent, the scheduler S. Besides playing the role of the authority, S schedules all other agents in a run according to the protocol phases. The trust assumptions are as explained and motivated at the beginning of Section 3.1 and the end of Section 3.2. In particular, we assume that the voting authority Auth (more generally, the scheduler S) and the BB are honest, i.e., are not corrupted by the adversary.

5.2 Verifiability

In this section, we establish the level of verifiability provided by Kryvos. To this end, we use the generic and widely used verifiability definition proposed in [75] which we briefly recall first. This definition has already been applied to numerous e-voting protocols and building blocks thereof [24, 77, 8, 78, 79, 76, 75, 80, 81].

Verifiability Framework. The verifiability definition [75] assumes a "virtual" entity, called the *judge* J, whose role is to either accept or reject a protocol run. In a real election, the program of the judge can be

executed by any party, including external observers and even voters themselves. The judge takes as input solely public information (e.g., the zero-knowledge proofs in Kryvos published on the BB to perform certain checks). In the context of e-voting, for verifiability to hold, the judge should only accept a run if "the announced election result corresponds to the actual choices of the voters". This statement is formalized via the notion of a *goal* γ of a protocol P. A goal γ is simply a set of protocol runs for which the mentioned statement is true, where the description of a run contains the description of the protocol, the adversary with which the protocol runs, and the random coins used by these entities.

Now, following [75] (see also Appendix J), we say that a goal γ is *verifiable* by the judge J in a protocol P, if the probability that J accepts a run r of P even though the goal γ is violated (i.e., $r \notin \gamma$) is negligible in the security parameter.

Analysis. We prove the verifiability result for Kryvos under the following assumptions:

(V1) The encryption scheme is correct, the commitment scheme is homomorphic and computationally binding, and all NIZKPs are (computationally) sound.

(V2) The scheduler S, the judge J (see Appendix K), and the BB are honest: $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$.[14]

Note that an arbitrary number of voters and talliers may be controlled by the adversary.

We instantiate the verifiability definition with the goal $\gamma(\varphi)$ proposed in [20], which captures the intuition of γ given before and with φ as above (see Appendix J).

Theorem 1 (Verifiability). *Under the assumptions (V1) and (V2), the goal $\gamma(\varphi)$ is verifiable by the judge J in the protocol $\mathsf{P}_{\mathsf{Kryvos}}(n_v, n_t, C, f_{res}, \mu)$.*

Our formal proof of the verifiability theorem is provided in Appendix K. This result mainly uses the soundness of the NIZKPs/SNARKs employed in Kryvos. The underlying relations of these proofs (see Appendix B.1) yield a "global" relation which effectively ensures the goal $\gamma(\varphi)$.

5.3 Privacy

Our privacy analysis of Kryvos makes use of the privacy definition for e-voting protocols proposed in [76]. This definition is particularly useful for our purposes as it allows us to *measure* the level of privacy a protocol provides, especially compared to protocols that publish the full tally, unlike other definitions (see, e.g., [82]). We first briefly recall the privacy definition from [76] (Definition 1). We then formally introduce the novel notion of publicly tally-hiding e-voting (Definition 2) and show that Kryvos meets this notion.

Privacy Framework. The definition proposed in [76] formalizes privacy of an e-voting protocol as the inability of an adversary to distinguish whether some voter $\mathsf{V}_{\mathsf{obs}}$ (the *voter under observation*) who runs her honest program voted for choice m_0 or choice m_1.

To define this notion formally, we first introduce the following notation for an arbitrary e-voting protocol P for voting method (C, f_{res}). Given a voter $\mathsf{V}_{\mathsf{obs}}$ and $m \in C$, we consider instances of P of the form $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m) \| \pi^* \| \pi_{\mathsf{A}})$ where $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m)$ is the honest program of the voter $\mathsf{V}_{\mathsf{obs}}$ under observation who takes m as her choice, π^* is the composition of programs of the remaining parties in P, and π_{A} is the program of the adversary. In the case of Kryvos, π^* includes the scheduler, the BB, all other voters, and all talliers.

Let $\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1]$ denote the probability that the adversary writes the output 1 on some dedicated tape in a run of $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m) \| \pi^* \| \pi_{\mathsf{A}})$ with security parameter ℓ and some $m \in C$, where the probability is taken over the random coins used by the parties in $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m) \| \pi^* \| \pi_{\mathsf{A}})$.

Now, vote privacy is defined as follows, where for Kryvos we quantify over all adversaries π_{A} which neither corrupt the BB nor the scheduler S.

Definition 1 (Privacy). *Let P be a voting protocol, $\mathsf{V}_{\mathsf{obs}}$ be the voter under observation, and $\delta \in [0,1]$. Then,* P achieves δ-privacy*, if for all choices $m_0, m_1 \in C$ and all adversaries π_{A} the difference*

$$\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_0) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_1) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1]$$

[14] See Appendix J for more details on φ.

is δ-bounded as a function of the security parameter 1^ℓ.[15]

In other words, the level δ is an upper bound of an arbitrary adversary's advantage to "break" vote privacy. Therefore, δ should be as small as possible. Note, however, that even for an ideal e-voting protocol with a completely passive adversary, δ might not be 0 (depending on the result function): for example, if the full tally is published as part of the result, then there might be a non-negligible chance that all honest voters, including the voter under observation, voted for the same candidate, in which case the adversary can easily derive from the final tally how the voter under observation voted.

Ideal Privacy. Often, and this will also be the case for Kryvos, formal privacy results are formulated w.r.t. the privacy level $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{res}})$ an ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(n_v, n_v^h, C, f_{\text{res}}, \mu)$ for voting method (C,f_{res}) provides. In this protocol (cf. Figure 11 in Appendix I), honest voters pick their choices according to the distribution μ. In every run, there are n_v^h many honest voters and n_v voters overall. The ideal protocol collects the votes of the honest voters and the dishonest ones (where the latter ones are independent of the votes of the honest voters) and outputs the result according to the function f_{res}.

A generic formula of $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{res}})$ for arbitrary voting methods (C, f_{res}) was published in [24]. In this work, for the first time, we instantiate $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{res}})$ for complex ranked-choice voting methods, such as IRV. We depict concrete values of the ideal privacy level for the IRV function f_{IRV} for two different distributions in Figure 4. While one distribution is uniform, the other is modeled more realistically: we sorted the candidates into a political spectrum and assumed that if a voter chooses a candidate as her first preference, she will most likely rank a candidate with a similar political opinion as rank 2 (see Appendix G for details). As the figure shows, revealing the full tally leads to a very low level of privacy, as expected, causing severe privacy issues, like for example Italian attacks. Comparing these levels with the ones revealing only the winner demonstrates that voting systems which hide the tally for ranked-choice voting methods provide dramatically better vote privacy than those voting systems which always reveal the complete tally. Even more: we see that hiding the tally is in fact *necessary* to achieve a reasonable privacy level for IRV elections that include more than just a few candidates. For simple voting methods, privacy levels comparing tally-hiding and non-tally-hiding systems can be found in [24].

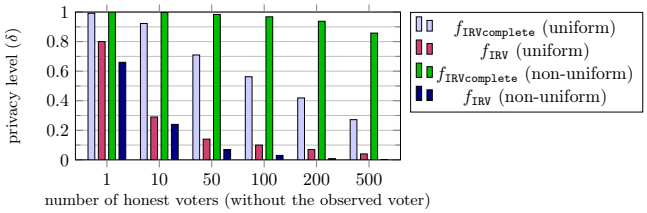

Fig. 4: Level of privacy (δ) for the ideal protocol with 5 candidates, IRV voting, and no dishonest voters. $f_{\texttt{IRVcomplete}}$ reveals the complete tally and $f_{\texttt{IRV}}$ only reveals the winner of the election.

Publicly Tally-Hiding. We now introduce the novel notion of publicly tally-hiding e-voting. Intuitively, an e-voting protocol P (like Kryvos) is publicly tally-hiding for some voting method (C, f_{res}) if the following two conditions hold true:

1. *Public privacy:* Under the assumption that all talliers are honest, P provides the same level of privacy as the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}$ for voting method (C, f_{res}), which reveals nothing but the actual election result by definition. That is, no one, except for the talliers, learns anything beyond the published election result.

[15] A function f is *δ-bounded* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \delta + \ell^{-c}$ for all $\ell > \ell_0$. Also see e.g. [83] for an explicit formula for δ.

2. *Internal privacy:* Under the assumption that less talliers than a certain threshold t are dishonest, P provides the same level of privacy as the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}$ for voting method (C, f_{complete}), where f_{complete} is the function that returns the *full tally* (e.g., the number of votes for all choices/candidates). In other words, the talliers do not learn more than they would for a non-tally-hiding secure e-voting protocol.

We now define this notion formally. We assume some set $\mathcal{T}$ of talliers and some threshold t; for Kryvos we have $\mathcal{T} = \{\mathsf{T}, \ldots, \mathsf{T}_{n_t}\}$ and $t = n_t$.

Definition 2 (Publicly Tally-Hiding). *Let P be a voting protocol with a set of talliers $\mathcal{T}$ and $t \leq |\mathcal{T}|$. We say that P is (δ_p, δ_i)-*publicly tally-hiding w.r.t. $(\mathcal{T}, t)$ *iff the following two conditions hold true:*

Public Privacy: *If all parties $T \in \mathcal{T}$ are honest, then P achieves δ_p-privacy.*
Internal Privacy: *If at most $t-1$ parties $T \in \mathcal{T}$ are dishonest, then P achieves δ_i-privacy.*

We call δ_p the public and δ_i the internal privacy level.

As explained, the public and internal privacy levels of a secure protocol should correspond to privacy levels of the ideal protocols mentioned above. However, we do not explicitly require this in the definition in order to be able to use this definition to measure the privacy level of an e-voting system. Moreover, we note that for all "reasonable" publicly tally-hiding voting protocols, the public privacy level δ_p should be much better (closer to 0) than the internal privacy level δ_i.

Analysis. To analyze the publicly tally-hiding property of Kryvos, we make the following assumptions about the primitives we use (see also Section 3):

(P1) The public-key encryption scheme is IND-CCA2-secure, the SNARKs and the NIZKPs are perfectly zero-knowledge, and the commitment scheme is perfectly hiding.

(P2) An adversary π_{A} does neither corrupt the scheduler S nor the BB, and at least n_v^h voters are honest.

Theorem 2 (Publicly Tally-Hiding). *Let $\mathcal{T} = \{\mathsf{T}, \ldots, \mathsf{T}_{n_t}\}$ and $t = n_t$. Then, assuming (P1) and (P2) hold, the voting protocol $\mathsf{P}_{\mathsf{Kryvos}}(n_v, n_t, C, f_{res}, \mu)$ is $(\delta^{ideal}_{(n_v,n_v^h,\mu)}(C, f_{res}), \delta^{ideal}_{(n_v,n_v^h,\mu)}(C, f_{complete}))$-publicly tally-hiding w.r.t. $(\mathcal{T}, n_t)$.*

The previous theorem essentially states that the public privacy level δ_p of Kryvos is the ideal one for (C, f_{res}), and its internal privacy level δ_i is the ideal one for (C, f_{complete}) where f_{complete} returns the number of votes for each choice/candidate. Figure 4 illustrates that for IRV Kryvos dramatically improves public privacy compared to non-tally-hiding systems (for which $\delta_i = \delta_p \geq \delta^{\text{ideal}}(C, f_{\text{complete}})$).

The formal proof of Theorem 2 is provided in Appendix I. The proof is based on two sequences of games, one for internal and one for public privacy, respectively.

6 Related Work

In this section, we discuss and compare Kryvos with related e-voting systems that were designed to be tally-hiding [25, 26, 29, 27, 28, 24, 84, 85] or to protect against Italian attacks [30, 22, 23, 1, 29, 31], respectively. We also briefly discuss our choice of the Groth16 SNARK in comparison with alternative proof systems.

Fully tally-hiding e-voting. The idea of fully tally-hiding e-voting and the first such system was proposed by Benaloh [25]. Hevia and Kiwi [26] published a fully tally-hiding e-voting system specifically tailored to jury votings. Wen and Buckland were the first to show how the tally can be fully hidden in IRV elections [29]. However, unlike Kryvos, the aforementioned protocols [25, 26, 29] were neither formally proven secure nor were they implemented to show their practicality (e.g., the computational complexity of [29] suggests that it is actually not truly practical). Another fully tally-hiding e-voting protocol has been proposed by Szepieniec and Preneel [27], but unfortunately, their protocol is insecure. The authors discuss some mitigations but do not solve the problem (see [27], Appendix A, for details). Cortier et al. [85] have proposed fully tally-hiding MPC components for the tallying phase of a multitude of result functions and studied their asymptotic

complexity but do not provide an implementation for showing their practicality. We therefore concentrate our discussion regarding fully tally-hiding systems on [28, 24] in what follows. The design of these systems is quite different to Kryvos as they are based on MPC.

Canard et al. [28] proposed a fully tally-hiding e-voting system specifically for Majority Judgement. However, there is a non-negligible chance that the system does not output a result.[16] They implemented their system and provide benchmarks which demonstrate that their system can handle large numbers of voters, just like Kryvos. While their system needs almost 20 minutes to tally 5 candidates with 5 possible grades and up to $2^{20} - 1$ voters, Kryvos can handle all practically relevant numbers of candidates and grades with up to $2^{32} - 1$ voters, as shown in Figure 2. For example, 7 candidates and 6 grades are evaluated in 5.678 seconds. Hence, Kryvos shows for the first time that publicly tally-hiding systems can not only realize majority judgement but also, as initially hoped, indeed achieve much better efficiency than a fully tally hiding system (by providing weaker privacy towards talliers). We further note that, in contrast to Kryvos, the security of [28] was not formally analyzed and the implementation was not tested in a distributed network but on a single computer only. Due to the online complexity of the underlying MPC protocol employed in [28], it is not clear how [28] performs in real-world distributed tallying scenarios.

Ordinos [24, 84] is the first provably secure verifiable fully tally-hiding e-voting system. It has been implemented for a wide range of voting methods and result functions with detailed benchmarks provided in [24, 84]. The main difference between Kryvos and Ordinos is their balance between efficiency and the tally-hiding property they provide: Ordinos provides the stronger notion of *fully* tally-hiding and is practical for rather simple voting methods (single and certain multi votes), simple result functions, and only a limited number of choices (yet large numbers of voters), whereas Kryvos provides the relaxed notion of *public* tally-hiding and is practical even for very complex voting methods (e.g., Borda or Condorcet methods) and result functions (e.g., IRV) and large numbers of possible choices. This is also illustrated in Figure 1 where we compare the runtime of various result functions of Ordinos and Kryvos. Unlike for Kryvos, the result function has a strong impact on performance in Ordinos. The benchmarks of the vote threshold result function of Ordinos provide a lower bound for all other result functions that Ordinos supports. We also include benchmarks of the Condorcet Smith set result function from the extension of Ordinos [84] to illustrate that this is indeed the case.

We again note that both publicly tally-hiding and fully tally-hiding protocols offer the same level of privacy w.r.t. the public. However, fully tally-hiding protocols offer a better level of privacy w.r.t. talliers, whereas publicly tally-hiding systems can provide better efficiency, as demonstrated by Kryvos.

Partially tally-hiding e-voting. A number of partially tally-hiding protocols have been proposed that solve the long-standing issue of Italian attacks for complex voting methods, such as Condorcet, Borda, IRV, and STV (e.g., [30, 22, 23, 1, 31]). In fact, as far as we are aware, all existing partially tally-hiding systems focus on mitigating Italian attacks. Some of these systems are quite efficient and able to handle real world elections, such as [1], which is one of the most efficient ones and has been shown to be practical for the 2015 New South Wales IRV elections.

As already described in Section 1, partially and publicly tally-hiding systems offer different trade-offs between privacy and efficiency compared to fully tally-hiding ones. These trade-offs lead to incomparable privacy properties. Specifically, while publicly tally hiding protocols hide the full tally from the public, partially tally-hiding systems still reveal some intermediate information about the tally to the public. For example, in the voting protocol for IRV elections by Ramchen et al. [1], the public learns, among others, the order of the weakest candidates, which might embarrass those candidates. On the other hand, partially tally-hiding protocols hide parts of the tally even from internal parties, whereas publicly tally-hiding protocols reveal the full aggregated tally to the talliers. Hence, Kryvos does not protect against Italian attacks performed by (one of) the talliers. This is in contrast to the partially and fully tally-hiding systems mentioned above, which protect against Italian attacks also in this situation.

[16] The reason for this is due to the underlying evaluation algorithm that does not provide a result for every possible tally. Kryvos uses a different algorithm that always outputs the correct result.

In terms of efficiency, Table 3 illustrates that Kryvos is well able to handle the same real world IRV elections as [1]. Hence, both protocols are practical solutions for IRV elections that provide different incomparable balances in terms of privacy.

Regarding the choice of the Groth16 SNARK. There are many other possibilities for instantiating the zero-knowledge proofs for Kryvos. However, in comparison with Groth16, many existing constructions have significantly worse benchmarks in terms of proof size and/or verification time [46, 47]. Furthermore, constructing a secure e-voting system requires care and not every proof system is directly applicable in this context. For example, [86] tries to construct a secure e-voting system based on Commit-and-Prove SNARKs [87] but faces several challenges when breaking the link between a casted ballot and a voter.

However, we emphasize that the (results for the) efficient combination of algorithms for computing Pedersen commitments will also carry over to other SNARKs as long as the base field of the elliptic curve for the Pedersen commitments is compatible with the base field of the SNARK. This is the case for Groth16, but might require some work for other SNARKs.

7 Conclusion

With Kryvos, we proposed the first provably secure verifiable e-voting system which directly follows the common practice of publicly tally-hiding elections. This enabled us to come up with a radically different design which offers a new, practically relevant balance between privacy and efficiency compared to all previous approaches.

Along the way, we presented and carefully evaluated various design and implementation options for publicly tally-hiding systems, which does motivate and justify the current system.

We finally note that Kryvos is of interest also beyond its tally-hiding property since it is the first verifiable homomorphic e-voting system that supports very complex ballot formats, along with complex well-formedness conditions. By this, Kryvos also opens new options for homomorphic e-voting systems in general.

Acknowledgements

This research was supported by the DFG through grant KU 1434/11-1, by the CRYPTECS project which has received funding from the German BMBF through grant 16KIS1441, and from the French ANR through grant ANR-20-CYAL-0006, by European Social Fund via IT Academy programme, by the Carl Zeiss Foundation, and by the Centre for Integrated Quantum Science and Technology (IQ^{ST}).

Johannes Müller was supported by the Luxembourg National Research Fund (FNR), under the CORE Junior project FP2 (C20/IS/14698166/FP2/Mueller).

References

1. K. Ramchen, C. Culnane, O. Pereira, and V. Teague, "Universally Verifiable MPC and IRV Ballot Counting," in *Financial Cryptography and Data Security - FC 2019, Revised Selected Papers*, ser. LNCS, vol. 11598. Springer, 2019, pp. 301–319.
2. NSW Government, "Constitution Act No 32," https://legislation.nsw.gov.au/~/view/act/1902/32, 2020.
3. Government of India, "Constitution of India," https://www.india.gov.in/my-government/constitution-india, 2020.
4. The National Archives, "Greater London Authority Act 1999," https://www.legislation.gov.uk/ukpga/1999/29/contents, 2011.
5. Maine State Legislature, "Ranked Choice Voting in Maine," http://legislature.maine.gov/lawlibrary/ranked-choice-voting-in-maine/9509, 2020.
6. J. Benaloh, "Verifiable secret ballot elections," Ph.D. dissertation, Yale University, 1987.
7. A. Juels, D. Catalano, and M. Jakobsson, "Coercion-Resistant Electronic Elections," in *Proceedings of Workshop on Privacy in the Eletronic Society (WPES 2005)*. ACM Press, 2005, pp. 61–70.
8. R. Küsters, J. Müller, E. Scapin, and T. Truderung, "sElect: A Lightweight Verifiable Remote Voting System," in *CSF 2016*, 2016, pp. 341–354.
9. M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a Secure Voting System," in *S&P 2008*, 2008, pp. 354–368.
10. B. Adida, "Helios: Web-based Open-Audit Voting," in *USENIX 2008*, 2008, pp. 335–348.
11. D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman, "Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes," in *USENIX/ACCURATE Electronic Voting Technology (EVT 2008)*. USENIX Association, 2008, see also http://www.scantegrity.org/elections.php.
12. S. Bell, J. Benaloh, M. Byrne, D. DeBeauvoir, B. Eakin, G. Fischer, P. Kortum, N. McBurnett, J. Montoya, M. Parker, O. Pereira, P. Stark, D. Wallach, and M. Winn, "STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System," *USENIX Journal of Election Technology and Systems (JETS)*, vol. 1, pp. 18–37, August 2013.
13. V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election Verifiability for Helios under Weaker Trust Assumptions," in *ESORICS 2014*, 2014, pp. 327–344.
14. A. Kiayias, T. Zacharias, and B. Zhang, "End-to-End Verifiable Elections in the Standard Model," in *Advances in Cryptology - EUROCRYPT 2015*, ser. Lecture Notes in Computer Science, vol. 9057. Springer, 2015, pp. 468–498.
15. ——, "DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 352–363. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813727
16. J. Epstein, "Weakness in Depth: A Voting Machine's Demise," *IEEE Secur. Priv.*, vol. 13, no. 3, pp. 55–58, 2015.
17. D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security Analysis of the Estonian Internet Voting System," in *Proceedings of the 2014 ACM CCS*, 2014, pp. 703–715.
18. S. Wolchok, E. Wustrow, J. A. Halderman, H. K. Prasad, A. Kankipati, S. K. Sakhamuri, V. Yagati, and R. Gonggrijp, "Security Analysis of India's electronic Voting Machines," in *Proceedings of the 17th ACM CCS*, 2010, pp. 1–14.
19. T. Haines, S. J. Lewis, O. Pereira, and V. Teague, "How Not to Prove Your Election Outcome," in *2020 IEEE SP 2020*. IEEE, 2020, pp. 644–660.
20. V. Cortier, D. Galindo, R. Küsters, J. Müller, and T. Truderung, "SoK: Verifiability Notions for E-Voting Protocols," in *S&P 2016*, 2016, pp. 779–798.
21. B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater, "Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios," in *USENIX/ACCURATE Electronic Voting Technology (EVT 2009)*, 2009.
22. J. Heather, "Implementing STV securely in Prêt à Voter," in *IEEE CSF 2007*, 2007, pp. 157–169.
23. J. Benaloh, T. Moran, L. Naish, K. Ramchen, and V. Teague, "Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting," *IEEE Trans. Information Forensics and Security*, vol. 4, no. 4, pp. 685–698, 2009.
24. R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, "Ordinos: A Verifiable Tally-Hiding Remote E-Voting System," in *IEEE EuroS&P 2020*, 2020.
25. J. D. Benaloh, "Improving Privacy in Cryptographic Elections (technical report)," Technical report, Tech. Rep., 1986.

26. A. Hevia and M. A. Kiwi, "Electronic Jury Voting Protocols," in *LATIN 2002, Proceedings*, 2002, pp. 415–429.
27. A. Szepieniec and B. Preneel, "New Techniques for Electronic Voting," *USENIX Journal of Election Technology and Systems (JETS)*, vol. 3, no. 2, pp. 46 – 69, 2015.
28. S. Canard, D. Pointcheval, Q. Santos, and J. Traoré, "Practical Strategy-Resistant Privacy-Preserving Elections," in *ESORICS 2018*, 2018, pp. 331–349.
29. R. Wen and R. Buckland, "Minimum Disclosure Counting for the Alternative Vote," in *VoteID 2009. Proceedings*, ser. LNCS, vol. 5767. Springer, 2009, pp. 122–140.
30. W. Jamroga, P. B. Rønne, P. Y. A. Ryan, and P. B. Stark, "Risk-Limiting Tallies," in *E-Vote-ID 2019, Proceedings*, ser. LNCS, vol. 11759. Springer, 2019, pp. 183–199.
31. M. R. Clarkson and A. C. Myers, "Coercion-Resistant Remote Voting Using Decryption Mixes," in *In Frontiers in Electronic Elections (FEE 2005)*, 2005.
32. ACM, "Q&A on ACM's Internet Voting," https://www.acm.org/binaries/content/assets/acmelections/acminternetvoting-1.pdf, 2020.
33. Gesellschaft für Informatik (GI), "GI Wahlen," https://gi.de/wahlen, 2019.
34. CrossRef, "Election process and results," https://www.crossref.org/board-and-governance/elections/, 2019.
35. Society for Industrial and Applied Mathematics (SIAM), "SIAM Announces New 2020 Leadership," https://sinews.siam.org/Details-Page/siam-announces-new-2020-leadership-1, 2019.
36. Deutsche Forschungsgemeinschaft (DFG), "DFG Fachkollegienwahl 2019," https://www.dfg.de/download/pdf/dfg_im_profil/gremien/fachkollegien/fk-wahl2019/fkwahl_2019_wahlergebnis_endgueltig_200218.pdf, 2019.
37. Personal communication (email) with Philip Wright, Technical Director of CES, 2020.
38. J. Groth, "On the Size of Pairing-Based Non-interactive Arguments," in *EUROCRYPT 2016, Proceedings, Part II*, ser. LNCS, vol. 9666. Springer, 2016, pp. 305–326.
39. N. Huber, R. Küsters, T. Krips, J. Liedtke, J. Müller, D. Rausch, P. Reisert, and A. Vogt, "Implementation of Kryvos." 2022, https://github.com/JulianLiedtke/kryvos.
40. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, Transparent, and Post-Quantum Secure Computational Integrity," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 46, 2018.
41. I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster Zero-Knowledge for Boolean Circuits," in *25th USENIX Security Symposium, 2016*. USENIX Association, 2016, pp. 1069–1083.
42. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn, "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings," in *Proceedings of the 2019 ACM CCS*, 2019, pp. 2111–2128.
43. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup," in *Proceedings of the 2017 ACM CCS*, 2017, pp. 2087–2104.
44. A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 953, 2019.
45. A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and Transparent Recursive Proofs from Holography," in *EUROCRYPT 2020, Proceedings, Part I*, ser. LNCS, vol. 12105. Springer, 2020, pp. 769–793.
46. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 315–334.
47. E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent Succinct Arguments for R1CS," in *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*, ser. Lecture Notes in Computer Science, vol. 11476. Springer, 2019, pp. 103–128.
48. S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 2087–2104.
49. scipr-lab, "libsnark," https://github.com/scipr-lab/libsnark, 2017.
50. S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller, "Publicly Auditable MPC-as-a-Service with succinct verification and universal setup," in *IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 2021, pp. 386–411.
51. A. Ozdemir and D. Boneh, "Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets," *IACR Cryptol. ePrint Arch.*, p. 1530, 2021.
52. R. Cramer, M. K. Franklin, B. Schoenmakers, and M. Yung, "Multi-Authority Secret-Ballot Elections with Linear Work," in *EUROCRYPT 1996*, 1996, pp. 72–83.
53. A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, E. Shi *et al.*, "CØCØ: A Framework for Building Composable Zero-Knowledge Proofs," *Cryptology ePrint Archive*, 2015.

54. B. Feng, L. Qin, Z. Zhang, Y. Ding, and S. Chu, "ZEN: Efficient Zero-Knowledge Proofs for Neural Networks," Cryptology ePrint Archive, Tech. Rep. 2021/87, 2021.
55. T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed., vol. 576. Springer, 1991, pp. 129–140. [Online]. Available: https://doi.org/10.1007/3-540-46766-1_9
56. K. Okeya and K. Sakurai, "Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve," in *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2162. Springer, 2001, pp. 126–141.
57. B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013, pp. 238–252.
58. J. Bootle and J. Groth, "Efficient Batch Zero-Knowledge Arguments for Low Degree Polynomials," in *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 10770. Springer, 2018, pp. 561–588.
59. T. Attema, I. Cascudo, R. Cramer, I. B. Damgård, and D. Escudero, "Vector Commitments over Rings and Compressed Σ-Protocols," *Cryptology ePrint Archive*, 2022.
60. C. Culnane and S. A. Schneider, "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems," in *IEEE CSF 2014*, 2014, pp. 169–183.
61. A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias, "On the Security Properties of e-Voting Bulletin Boards," in *SCN 2018, Proceedings*, 2018, pp. 505–523.
62. L. Hirschi, L. Schmid, and D. A. Basin, "Fixing the Achilles Heel of E-Voting: The Bulletin Board," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 109, 2020.
63. J. Benaloh, "Ballot Casting Assurance via Voter-Initiated Poll Station Auditing," in *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007*, 2007.
64. D. Galindo, S. Guasch, and J. Puiggali, "2015 Neuchâtel's Cast-as-Intended Verification Mechanism," in *VoteID 2015, Proceedings*, 2015, pp. 3–18.
65. R. Cramer and V. Shoup, "A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack," in *CRYPTO '98, Proceedings*, ser. LNCS, vol. 1462. Springer, 1998, pp. 13–25.
66. J. Camenisch and V. Shoup, "Practical Verifiable Encryption and Decryption of Discrete Logarithms," in *CRYPTO 2003, Proceedings*, ser. LNCS, vol. 2729. Springer, 2003, pp. 126–144.
67. T. Haines, D. Pattinson, and M. Tiwari, "Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme," in *Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 12031. Springer, 2019, pp. 36–53.
68. R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols," in *Advances in Cryptology - CRYPTO '94, Proceedings*, 1994, pp. 174–187.
69. European Broadcasting Union, "Eurovision Song Contest - How it works," https://eurovision.tv/about/how-it-works, 2020.
70. Republic of Nauru, "Electoral Act No. 15," http://ronlaw.gov.nr/nauru_lpms/files/acts/d83250a1ebdc56c1701fa7aa245af5b1.pdf, 2016.
71. M. Balinski and R. Laraki, "A Theory of Measuring, Electing, and Ranking," *Proceedings of the National Academy of Sciences*, vol. 104, no. 21, pp. 8720–8725, 2007.
72. ——, "Judge: Don't vote!" *Operations Research*, vol. 62, no. 3, pp. 483–511, 2014.
73. Debian Project, "Ubuntu IRC Council Position," https://www.debian.org/vote/, 2012.
74. Electoral Commission NSW, "NSW State Electoin Results 2015," https://pastvtr.elections.nsw.gov.au/SGE2015/la-home.htm, 2015.
75. R. Küsters, T. Truderung, and A. Vogt, "Accountability: Definition and Relationship to Verifiability," in *ACM CCS 2010*, 2010, pp. 526–535.
76. R. Küsters, T. Truderung, and A. Vogt, "Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study," in *IEEE S&P 2011*, 2011, pp. 538–553.
77. R. Küsters and T. Truderung, "Security Analysis of Re-Encryption RPC Mix Nets," in *IEEE EuroS&P 2016*. IEEE Computer Society, 2016, pp. 227–242.
78. R. Küsters, T. Truderung, and A. Vogt, "Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking," in *S&P 2014*, 2014, pp. 343–358.

79. ——, "Clash Attacks on the Verifiability of E-Voting Systems," in *IEEE S&P 2012*, 2012, pp. 395–409.
80. T. Haines and J. Müller, "SoK: Techniques for Verifiable Mix Nets," in *IEEE 33rd Computer Security Foundations Symposium, CSF, 2020*. IEEE Computer Society, 2020.
81. X. Boyen, T. Haines, and J. Müller, "A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing," in *Computer Security - 25th European Symposium on Research in Computer Security, ESORICS 2020*, 2020.
82. D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions," in *IEEE S&P 2015*, 2015, pp. 499–516.
83. D. Mesten, J. Müller, and P. Reisert, "How Efficient are Replay Attacks against Vote Privacy? A Formal Quantitative Analysis," in *IEEE 35rd Computer Security Foundations Symposium, CSF, 2022*, 2022. To appear.
84. F. Hertel, N. Huber, J. Kittelberger, R. Küsters, J. Liedtke, and D. Rausch, "Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting," Cryptology ePrint Archive, Tech. Rep. 2021/1420, 2021.
85. V. Cortier, P. Gaudry, and Q. Yang, "A toolbox for verifiable tally-hiding e-voting systems," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 491, 2021.
86. J. Lee, J. Choi, J. Kim, and H. Oh, "SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1270, 2019.
87. M. Campanelli, D. Fiore, and A. Querol, "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019, pp. 2075–2092.
88. D. J. Bernstein, "Curve25519: New diffie-hellman speed records," in *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*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. Springer, 2006, pp. 207–228. [Online]. Available: https://doi.org/10.1007/11745853_14
89. P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.
90. C. Costello and B. Smith, "Montgomery curves and their arithmetic - The case of large characteristic fields," *J. Cryptogr. Eng.*, vol. 8, no. 3, pp. 227–240, 2018.
91. M. Bellare, G. Fuchsbauer, and A. Scafuro, "NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion," in *ASIACRYPT 2016, Proceedings, Part II*, ser. LNCS, vol. 10032, 2016, pp. 777–804.
92. B. Abdolmaleki, H. Lipmaa, J. Siim, and M. Zajac, "On Subversion-Resistant SNARKs," Cryptology ePrint Archive, Report 2020/668, 2020.
93. E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs," in *IEEE SP 2015*. IEEE Computer Society, 2015, pp. 287–304.
94. S. Bowe, A. Gabizon, and I. Miers, "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 1050, 2017.
95. B. Abdolmaleki, K. Baghery, H. Lipmaa, J. Siim, and M. Zajac, "UC-Secure CRS Generation for SNARKs," in *AFRICACRYPT 2019, Proceedings*, ser. LNCS, vol. 11627. Springer, 2019, pp. 99–117.
96. IBM, "IBM Hyper Protect Virtual Servers," https://www.ibm.com/products/hyper-protect-virtual-servers, 2020.
97. AMD, "AMD Secure Encrypted Virtualization," https://developer.amd.com/sev/, 2020.
98. B. Ward, "Majority Rule and Allocation," *Journal of Conflict Resolution*, vol. 5, no. 4, pp. 379–389, 1961.

A Designing Circuits for QAPs

A.1 Efficiently Proving Knowledge of the Tally

A SNARK can in principle be used to prove statements for arbitrary relations, however, the resulting performance (in terms of runtime, memory overhead, bandwith, ...) quickly deteriorates and becomes impractical for large circuits such as, e.g., cryptographic algorithms (see, e.g., [53, 54]). Therefore, a main challenge of using SNARKs consists of carefully constructing suitable circuits with minimal numbers of constrains for the intended relations such that the resulting SNARKs are practical. While we discuss the overall circuits for our SNARKs in Section 4, we need as a central building block for all of our SNARKs an efficient circuit for verifying decommitments, which we construct in the following.

Commitment Scheme. We use Pedersen commitments over a group $(\mathcal{G}, \cdot)$ of order q with generator g as introduced in [55]. Recall that to compute a Pedersen commitment, one first takes an auxiliary value h

defined by $h = g^a$ for a uniform $a \xleftarrow{\$} \mathbb{F}_q$ and then commits to a value $v \in \mathbb{F}_q$ with $\mathsf{Com}(v,r) = g^v h^r$ using a uniform $r \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$. Pedersen commitments are a standard solution that have a simple structure and hence also rather small computational complexity. Furthermore, they are additively homomorphic and perfectly hiding.

The choice of the group $\mathcal{G}$ significantly affects efficiency of the corresponding SNARK, with some groups being more favorable than others. For example the first natural choice $\mathcal{G} = \mathbb{F}_q$ for a prime q of secure bit length at least 2,048, is a priori incompatible with the underlying proof system, which supports only values of up to 255 bits. While it is possible to circumvent this problem, e.g., by splitting each value (of at least 2,048 bits) into chunks of at most 255 bits, the resulting circuit would be too inefficient for our use case (cf. the benchmarks for RSA encryption in [53]). We therefore use an elliptic curve $\mathcal{G}$, where sufficient security guarantees are already available at a much lower length q.

We choose specifically Curve25519 which is a Montgomery curve that works over $\mathbb{F}_q$ with $q = 2^{255} - 19$ prime and curve equation $y^2 = x^3 + Ax^2 + x$ for some $A \in \mathbb{F}_q$ with $A^2 \neq 4$. The exact value of A does not affect the efficiency of the SNARK in terms of the number of constraints representing the corresponding arithmetic circuit and can therefore be chosen freely (under the requirement $A^2 \neq 4$ of the curve). We use $A = 486{,}662$ as done in [88]. The coordinates of this curve can be represented by 255 bits and therefore do not require a splitting approach.

Optimizing the circuit. A Pedersen commitment $g^v \cdot h^r$ requires exponentiations and multiplications,[17] with exponentiations being the most expensive operation.

The chosen Montgomery curve allows to compute the exponentiation of a point g of $\mathcal{G}$ purely based on the first (affine) coordinate of $g = (g_x, g_y) \in \mathbb{F}_q \times \mathbb{F}_q$ and therewith we do not require a splitting approach. Furthermore, exponentiation on a Montgomery curve can be realized very efficiently with the ladder algorithm [89], [90] and this efficiency transfers to the SNARK—an observation already made in [53]. To fully use the efficency advantage of the ladder algorithm we have to work with projective coordinates for the exponentiation, since the resulting circuit does not have to seperately check special cases related to ∞ then. For other operation however, e.g. for multiplication, we use affine coordinates. Generally, we chose for each algorithm, the representation that allows the most efficient execution of this specific algorithm.

Unlike previous results, e.g. the C∅C∅-framework which focuses on a Diffie-Hellman key exchange inside a SNARK, Pedersen commitments do not only require exponentiations of points inside the SNARK, but also the multiplication of different curve points g^v and h^r in order to compute $\mathsf{Com}(v,r) = g^v \cdot h^r$. For this multiplication we do however need the y-coordinate of both g^v and h^r, which are not provided by the Montgomery ladder. There is however a standard way to reconstruct a y-coordinate by Okeya and Sakurai [56]. Their algorithm (y-Rec) takes the output of a Montgomery ladder, which is apart from $(g^v)_x$ also $(g^{v+1})_x$, as well as both coordinates of the original input g, and outputs the y-coordinate of $(g^v)_y$.

After we recovered both coordinates, we perform the before mentioned switch from projective to affine coordinates to prepare for the final multiplication of g^v and h^r.[18] The multiplication of g^v and h^r is then done in the standard way for Montgomery curves.

Altogether, we get the circuit illustrated in Figure 5 for the computation a Pedersen commitment.

Table 4 shows micro benchmarks of these operations inside the SNARK. Essentially, the recovery of the y-coordinate, the conversion, and the point multiplication are rather negligible compared to the complexity of the Montgomery ladder. This also shows that a Montgomery curve is a suitable choice in our setup, since the advantage of the efficient exponentiation on this curve easily compensates for the additional Okeya and Sakurai algorithm.

In principle, we have now obtained a way to represent a Pedersen commitment through an arithmetic circuit. However, as Table 4 shows, even with our design choices, representing a single Pedersen commitment still costs 11,701 constraints, which still turns out to be too expensive for our application.

[17] To stay compatible with the usual notation for Pedersen commitments, we denote the group operation on an elliptic curve with $\cdot$ instead of the usual $+$. The wording *addition* and *multiplication* is consequentially changed to *multiplication* and *exponentiation*.

[18] The coordinate transformation costs 15 constraints.

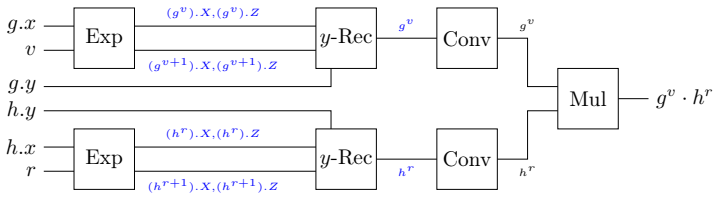

Fig. 5: Circuit of a Pedersen commitment. By $g.x$ resp. $g.y$ we denote the affine x-coordinate and the affine y-coordinate of g, while upper case letters denote the respective projective coordinates. Additionally, projective coordinates are illustrated in blue.

Operation	Constraints
Montgomery Ladder [89] (255 bits Exponent)	5,084
Montgomery Ladder [89] (32 bits Exponent)	624
y-Coordinate Recovery (Okeya-Sakurai Algorithm [56])	39
Point Multiplication	86
Conversion (Projective to Affine Coordinates)	16
Pedersen Commitment (v of 255 bits, r of 255 bits)	11,701
Pedersen Commitment (v of 32 bits, r of 255 bits)	6,549

Table 4: Number of (QAP) constraints for various operations.

Hence, we will now discuss further optimizations.

A.2 Further Performance Improvements

As discussed before, even with the introduced design choices, the resulting arithmetic circuit for computing a commitment is still too large for our purposes. Hence, we will now introduce further necessary optimizations that drastically reduce the number of constraints required for computing the commitments that correspond to $\mathbb{T}$ (recall that in the basic approach one commitment contains the number of votes for one choice, so $\mathbb{T}$ is represented by n_{choices} commitments). From Table 4 we can see that even when using a Montgomery curve and the Montgomery ladder, the main factor for the complexity of computing a Pedersen commitment inside a SNARK are still the exponentiations g^v and h^r. However, recall that in our application the input v is actually the aggregated number of votes for a certain candidate/choice and hence v will typically be rather small and not require the entire 255 bits (for example, in a single-vote election v will not exceed the number of eligible voters). As indicated by Table 4, limiting v to 32 bits already almost halves the constraints required to express the computation of a Pedersen commitment inside the SNARK.[19] Regarding the randomness r, however, we will need an exponentiation with up to 255 bits for security reasons.

Next, recall that as part of our system we want to compute commitments corresponding to the tally $\mathbb{T}$, where $\mathbb{T}$ is a list of the aggregated votes v_i for each candidate/choice i. If one implements this in a straightforward manner via standard Pedersen commitments, then there will be one separate commitment $\mathsf{com}(v_i,r_i)$ per v_i (just as Helios uses one ciphertext per choice). Each of these commitments introduces another exponentiation for some randomness r_i to obtain h^{r_i}, which also needs to be computed in the arithmetic circuit in order to prove knowledge of $\mathbb{T}$. Since each randomness requires the full 255 bits, this can become very costly in elections with multiple choices/candidates (cf. left side of Table 1). We solve this issue by instead using Pedersen vector commitments as described in [58]. Essentially, these commitments allow for committing to a vector $\boldsymbol{v} = (v_1, \ldots, v_N)$ with entries in $\mathbb{F}_q$ by computing

$$\mathsf{Com}(\boldsymbol{v},r) = g_1^{v_1} \cdot \ldots \cdot g_N^{v_N} \cdot h^r$$

[19] Note that the circuit also ensures that the input values are of correct size.

for generators $g_1, \ldots, g_N$ of $\mathcal{G}$ chosen uniformly at random. In this case, we say that N is the *slot size* of the commitment. One observes that Pedersen vector commitments are also additively homomorphic, perfectly hiding, and computationally binding under the discrete logarithm assumption [59]. By setting N to the number of choices, this construction requires just one exponentiation with randomness of 255 bits to commit to the full set of choices. This in turn drastically improves both the computational cost but also the size of the CRS as shown in Table 1. E.g., for committing to 250 values, moving from computing one commitment per value to computing one commitment for all values in a single SNARK, reduces the amount of required constraints by more than 1.4 million, which drastically reduces the proof time as well as the CRS size.

All in all, the various presented optimizations allow for proving knowledge of an opening inside a SNARK efficiently, which we can use in order to prove of knowledge of $\mathbb{T}$ in our approach for publicly tally-hiding e-voting. We can also use this technique to allow voters to prove validity of their ballot. As we will see in the Section 3, we will need both of these applications in Kryvos.

B Non-Interactive Zero-Knowledge Arguments

The following definition of non-interactive zero-knowledge arguments of knowledge follows [38].

Let $\mathcal{R}$ be a relation generator that given a security parameter λ in unary returns a polynomial time decidable binary relation R. For pairs $(x,w) \in R$, we call x the (public) statement and w the (secret) witness. We define $\mathcal{R}_\lambda$ to be the set of possible relations R that the relation generator may output given 1^λ. We will in the following assume that λ can be deduced from the description of R. The relation generator may also output some side information, an auxiliary input z, which will be given to the adversary. An *efficient prover publicly verifiable non-interactive argument* for $\mathcal{R}$ is a tuple of ppt algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ such that:

- $(\mathsf{CRS}, \tau) \leftarrow \mathsf{Setup}(R)$: The setup produces a common reference string (CRS) CRS and a simulation trapdoor τ for the relation R.
- $\pi \leftarrow \mathsf{Prove}(R, \mathsf{CRS}, x, w)$: The prover algorithm takes as input a common reference string CRS and $(x, w) \in R$ and returns an argument π.
- $0/1 \leftarrow \mathsf{Verify}(R, \mathsf{CRS}, x, \pi)$: The verification algorithm takes as input a common reference string CRS, a statement x, and an argument π, and returns 0 (reject) or 1 (accept).
- $\pi \leftarrow \mathsf{Sim}(R, \tau, x)$: The simulator takes as input a simulation trapdoor τ and statement x, and returns an argument π.

Definition 3. *We say that* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *is a perfect non-interactive zero-knowledge argument of knowledge for* $\mathcal{R}$ *if it has perfect completeness, perfect zero-knowledge and computational knowledge soundness as described below.*

In what follows, we describe the main idea of the respective properties and refer to [38] for the formal definitions:

- Perfect zero-knowledge: We say that an argument is *zero-knowledge* if it does not leak any information besides the truth of the statement.
- Computational soundness: We say that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ is *sound* if it is not possible to prove a false statement.
- Computational knowledge soundness: We say that $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ is an argument of knowledge if there is an extractor that can compute a witness whenever the adversary produces a valid argument.

B.1 Zero-Knowledge Proof Relations

In this section, we precisely define the relations for the NIZKPs that are used in Kryvos. We refer to Section 3 for the parameters.

Well-formed ballot. Let $m = (m^1, \ldots, m^{n_{\mathsf{choices}}})$ be the choice of the voter she wants to commit to. As specified by Auth, depending on the voting function, this choice is split into n_{tuples} blocks, each of size N. Thus, the actual committed value is $\tilde{m} := (m_1, \ldots, m_{n_{\mathsf{tuples}}})$. Let $c := (c^1, \ldots, c^{n_{\mathsf{tuples}}})$ be these commitments with randomness vector $r := (r_1, \ldots, r_{n_{\mathsf{tuples}}})$

The following relation describes that a vector c is a (vector) commitment to a valid choice $m \in C$ with randomness vector r.

$$R^C_{\mathsf{ballot}} = \{(c, (\tilde{m}, r)) \colon c = \mathsf{Com}(\tilde{m}; r) \wedge m \in C\}$$

We denote the SNARK for this relation by Π_{ballot}.

Correct final result. Let $x = (c^{\perp,1}, \ldots, c^{\perp,n_{\mathsf{tuples}}}, \mathrm{res})$ and $w = (\mathbb{T}, r^{\perp,1}, \ldots, r^{\perp,n_{\mathsf{tuples}}})$. By $t^{\perp,i}$ we denote the n_{tuples} tuples that represent the tally for the commitments.

The following relation describes that the final result res is correctly computed w.r.t. the commitments $(c^{\perp,1}, \ldots, c^{\perp,n_{\mathsf{tuples}}})$ and result function f_{res}.

$$\begin{aligned} R_{f_{\mathrm{res}}} = \{((c^{\perp,1}, \ldots, c^{\perp,n_{\mathsf{tuples}}}, \mathrm{res}), (\mathbb{T}, r^{\perp,1}, \ldots, r^{\perp,n_{\mathsf{tuples}}})) \colon \\ (t^{\perp,0}, \ldots, t^{\perp,n_{\mathsf{tuples}}}) = \mathbb{T}, \mathrm{res} = f_{\mathrm{res}}(\mathbb{T}), \\ \forall i \in \{1, \ldots, n_{\mathsf{tuples}}\} \colon c^{\perp,i} = \mathsf{Com}(t^{\perp,i}; r^{\perp,i})\} \end{aligned}$$

We denote the SNARK for this relation by $\Pi_{f_{\mathrm{res}}}$.

Correct decryption. The following relation describes that e is an encryption of message m under public key pk:

$$\begin{aligned} R_{\mathsf{dec}} = \{((e, m, \mathsf{pk}), \mathsf{sk}) \colon m = \mathsf{Dec}(e, \mathsf{pk}) \wedge \\ (\exists r \colon (\mathsf{pk}, \mathsf{sk}) = \mathsf{KeyGen}(r))\} \end{aligned}$$

We denote the NIZKP for this relation by Π_{dec}.

C CRS Generation

If the CRSs for the Groth16 SNARKs used in Kryvos are not generated honestly by the voting authority Auth but were rather generated maliciously (*subverted*), then the soundness and/or the zero-knowledge properties of the SNARKs break down.

This trust assumption can be mitigated using standard techniques: To allow for verifying that a CRS provides the zero-knowledge property, one only has to add some additional elements to the CRS during its generation, as detailed in [91],[92]. We note that these additional elements do not have to be downloaded in order to compute a proof, i.e., this mechanism does not add to the overall size that a voter needs to download in order to submit a ballot. To additionally retain the soundness property, one can let the talliers use MPC to generate the CRS in a distributed fashion [93], [94], [95]. Under the assumption that at least one tallier is honest, which we have to make for privacy anyways, the resulting CRS provides soundness.

We note that distributed CRS generation is indeed practical for the parameters used in the Kryvos instantiations from Section 4: The benchmarks of Bowe et al. [94] for a system with 2^{21} constraints show that generating and checking the CRS took approximately an hour. The SNARKs used in our system are quite similar to those benchmarked in [94] (in particular we use approximately 5 million constraints). We therefore estimate our case to be in the same order of magnitude. Since CRS generation can be performed far ahead of time, and needs to be performed only exactly once for each voting method, distributed CRS generation is a viable option.

Alternatively, in cases where trusting a hardware manufacturer is an option (e.g., small low stake elections such as boardroom voting), one can use trusted execution environments (TEEs) such as [96] or [97] to generate a CRS in an honest manner. The TEE should be chosen suitably, in particular it should have access to true randomness and a sufficient amount of memory.

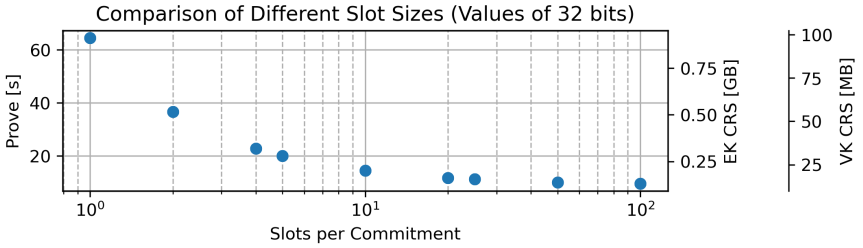

Fig. 6: Benchmarks of SNARKs that compute commitment(s) with different slot sizes to 100 values. Each point represents a single SNARK that uses commitments with the given number of slots to commit to 100 values.

D The Optimal Number of Slots per Commitment

As discussed in Section 4.1, the slot size of the Pedersen vector commitments used in Kryvos mainly influences the circuit size of the SNARKs that prove the correct opening of the commitments. These SNARKs appear at two parts of Kryvos: Firstly, the SNARKs for proving ballot well-formedness in the voting phase. Such SNARK proofs are created by each voter. Secondly, the SNARK in the evaluation phase used to prove the correct computation of the result function on the aggregated tally. This proof is created by the designated tallier. The SNARK for the evaluation always takes as input the complete aggregated tally and computes the election result based on the values of the aggregated tally. One natural question is to determine the slot size N used by the vector commitments for optimizing the computation of these SNARKs. Figure 6 shows benchmarks of monolithic SNARKs (i.e., one single SNARK proof is used to prove the correctness of the openings of $n_{\mathsf{choices}} = 100$ values) in dependency of the slot size N per commitment (which is equal to $100/\#commitments$ in this case). For $N < n_{\mathsf{choices}}$, such a monolithic SNARK needs to be computed by the talliers in order to prove the correct computation of the result function on the aggregated tally. The voters, however, can create an individual SNARK proof for each commitment.[20]

As explained in Section 2.3, each additional commitment introduces the computation of another exponentiation with some randomness inside the SNARK, which becomes very costly for large slot sizes. While in Table 1 we only considered the extreme cases of $N = 1$ and $N = n_{\mathsf{choices}}$, we see that the benchmarks in Figure 6 confirm the intuition that for proving the correct opening using a monolithic SNARK it is in general more efficient to use less commitments of higher slot size than using many commitments of smaller slot size for the same number of values. Thus, for the tallying SNARK (which is always a monolithic SNARK), we want to use the maximum slot size $N = n_{\mathsf{choices}}$ such that all choices fit into a single commitment.

On the other hand, it is crucial that the ballots can be created efficiently. For this, we require that the creation of the SNARK proof of the ballot's well-formedness is practical for the voters. As the benchmarks in Section 4 show, using one Pedersen vector commitment of maximal slot size containing the ballot inside a monolithic SNARK proof is usually also efficient enough for proving the ballot's well-formedness. However, one exception to this is Instant-Runoff Voting (IRV), where n_{choices} can become particularly high and hence the size of $\mathsf{CRS}_{\mathsf{EK}}$ as well as the time necessary to create a proof using a monolithic SNARK can become inpractical. In these cases we opt for letting voters use multiple commitments of smaller slot size as well as multiple SNARK proofs even though these multiple commitments only allow for less efficient tallying SNARKs. However, as already described in Section 4.3, e.g. using a slot size of $N = n_{\mathsf{choices}}/10$ allows for computing the SNARK proofs of ballot well-formedness as well as the monolithic tallying SNARK efficiently.

[20] Note that using commitments of different slot sizes would require the voters to use a different $\mathsf{CRS}_{\mathsf{EK}}$ for each commitment of different size. For this reason, if we want to use multiple commitments and multiple SNARKs, we always use commitments of equal slot size and adapt the relation to be proved in a way such that the voters need exactly one CRS for all SNARK proofs (see also Section 4.3 for more details on this relation).

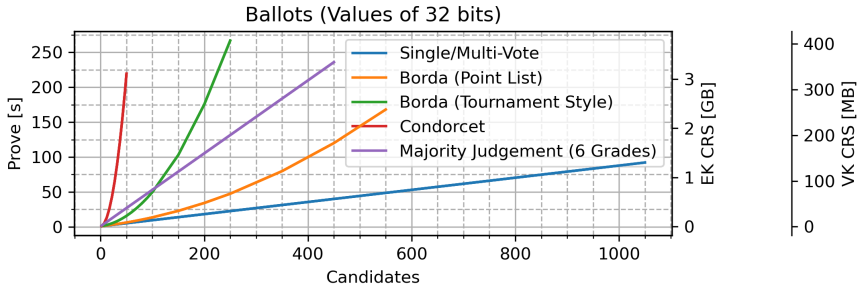

Fig. 7: Extended Benchmarks of SNARKs for proving Ballot validity.

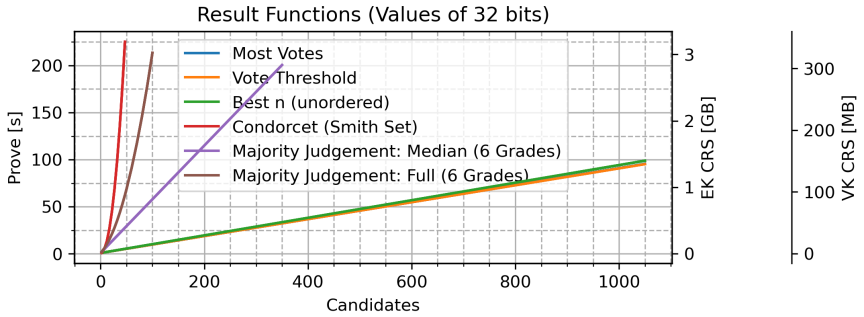

Fig. 8: Extended Benchmarks of Different Election Result Functions.

E Instantiation for Common Voting Methods

In this section we provide detailed descriptions, including the respective choice spaces, of the voting methods for which we instantiated Kryvos and provided and discussed benchmarks in Section 4. We provide extended benchmarks in Figures 7 and 8.

Single-Vote and Multi-Vote. The most straightforward type of elections are single vote elections, where every voter can give a single vote to one candidate/choice. The votes are then aggregated and used to compute the final result based on some result function f_{res}, e.g., the candidate with the most votes. This is easy to capture in Kryvos by using f_{res} along with the choice space C_{single} as defined in Section 3.1, where each choice corresponds to a single candidate.

A generalization of this election type is called multi-vote, where every voter may distribute n_{votes} votes among all candidates, potentially even allowing multiple votes for the same candidate. This variant can be captured by Kryvos using the choice space

$$C_{\mathsf{multi}} := \left\{ (x_1, \ldots, x_{n_{\mathsf{choices}}}) \mid x_i \in [0,b], \sum_{i=1}^{n_{\mathsf{choices}}} x_i = n_{\mathsf{votes}} \right\},$$

where $[0,b]$ denotes the set $\{0,1,\ldots,b\}$, b is the maximal number of votes allowed for each candidate and each choice corresponds to one candidate.

As can be seen from the general discussion in Section 4 (see also Figure 2 and Figure 3 and Figure 7 and Figure 8), Kryvos is quite efficient for both single and multi-vote elections using essentially arbitrary result functions f_{res}. For example, using an arbitrary number of talliers, $n_v = 500{,}000$ voters, and up to $n_{\mathsf{choices}} = 1{,}000$ candidates, then each voter needs under 100 seconds to cast a vote and the tallying phase also takes less than 100 seconds irrespective of f_{res}. In comparison, Ordinos [24] takes more than 90 minutes to evaluate the same result functions with three trustees and 40 candidates. A more in-depth comparison of Kryvos and Ordinos is given in Section 6.

Borda voting is a prominent ranked voting method, which is famously used for determining the winner of the grand final in the Eurovision Song Contest [69], but also for national elections, e.g., for parliamentary elections in the Republic of Nauru [70]. In Borda voting, a voter ranks the candidates according to her preferences and each candidate receives a number of points that depends on the position in the ranking. There are many ways how these points are determined. For Kryvos, we implemented and benchmarked two of them - nevertheless, other variants are also viable and can be realized with Kryvos. We implemented a generalization of the method that is used for the Eurovision Song Contest and for the Republic of Nauru (we call this variant *point list*), and the *tournament style* variant. We allowed arbitrary ties in both variants (Ties are typically only allowed for the last place in the ranking of Borda elections). All Borda variants aggregate the points for each candidate, an arbitrary result function can be applied to determine the winner of the election (for example, the candidate with the most points wins).

The *point list* ballot format works as follows. Each rank corresponds to a unique number of points defined via a list $\mathcal{P} \in \mathbb{N}^{n_{\mathsf{points}}}$, where in the standard case, $n_{\mathsf{points}} = n_{\mathsf{choices}}$ is the number of candidates and $\mathcal{P}[1]$ is the number of points assigned to the candidate that is ranked first and so on. The standard version of Borda requires candidates to be assigned a unique rank (i.e., unique number of points), advanced variants allow for assigning multiple candidates the same rank, i.e., the same number of points. In that case, if i candidates are assigned the same rank r, then the next lower candidate is required to start at rank $r + i$ (instead of $r + 1$, as is the case when ranks are assigned uniquely).

This is captured by the following choice space, where we interpret $\mathcal{P}$ both as a list and a set. Note that for simplicity we assume that $n_{\mathsf{points}} \leq n_{\mathsf{choices}}$.

$$
\begin{aligned}
C_{\mathsf{BordaPointList}}(\mathcal{P}) = \Big\{ (x_1, \ldots, x_{n_{\mathsf{choices}}}) \;\Big|\; & \forall i : x_i \in \mathcal{P} \wedge \\
& \forall r \in [1, n_{\mathsf{points}}], \forall i \in [2, \sigma(r)] : \\
& \nexists j \in [1, n_{\mathsf{choices}}] : x_j = \mathcal{P}[r + i - 1] \\
& \wedge (r + \sigma(r) \leq n_{\mathsf{points}} \\
& \Rightarrow \exists j \in [1, n_{\mathsf{choices}}] : x_j = \mathcal{P}[r + \sigma(r)]) \Big\},
\end{aligned}
$$

where $\sigma(r) \in [1, n_{\mathsf{choices}}]$ denotes the number of occurrences of $\mathcal{P}[r]$ in $(x_1, \ldots, x_{n_{\mathsf{choices}}})$. Using $C_{\mathsf{BordaPointList}}(\mathcal{P})$, standard Borda can be derived by additionally requiring $\sigma(r) = 1$ for all values of r. Further straightforward variations include allowing the voter to submit only a partial ranking.

The *tournament style* ballot format works as follows. Based on the ranking of the candidates by a voter, each candidate receives two points per candidate that is ranked lower, and one point for each other candidate with the same rank. This is captured by the following choice space:

$$
\begin{aligned}
C_{\mathsf{BordaTournamentStyle}} = \Big\{ (x_1, \ldots, x_{n_{\mathsf{choices}}}) \;\Big|\; & \forall i : \\
& x_i = 2 \cdot |\{j \in [1, n_{\mathsf{choices}}] : x_j < x_i\}| \\
& + |\{j \in [1, n_{\mathsf{choices}}] \setminus \{i\} : x_j = x_i| \Big\}.
\end{aligned}
$$

As shown in Figures 3 and 7, voters in Kryvos can efficiently prove that their ballots are well-formed. For example, for up to $n_{\mathsf{choices}} = 400$ candidates, a prover requires a total runtime of 100 seconds. The performance of the tallying phase is analogous to single/multi vote.

Condorcet methods are voting methods that can be used for single-seat elections. Among others, Condorcet methods are used for internal elections by the Debian project [73]. All Condorcet methods have the goal of determining a so-called *Condorcet winner*, that is a candidate that would win against each other candidate in direct comparison. However, there is no guarantee that a Condorcet winner even exits, as it is easily possible that no candidate is preferred by a majority of voters over each other candidate. Therefore several methods have been proposed in order to loosen the restriction for the election result to only consisting of the Condorcet winner whenever a Condorcet winner exists, while still yielding an election result if no Condorcet winner exists.

The most simple idea, as described in [98], is to declare multiple winners. A basic method for this is to declare a set of winners - the so-called *Smith set* - as the smallest set of candidates that win against each candidate outside of the set. Clearly, if an election has a Condorcet winner, then the Smith set will consist of exactly this candidate.

Kryvos supports elections based on the Condorcet method with Smith set. We allow each voter to indicate her preferred candidate for each pair of candidates, i.e. a ballot is of the form of an $(n_{\mathsf{cand}} \times n_{\mathsf{cand}})$-matrix A with entries 0 or 1 where n_{cand} denotes the number of candidates, i.e., we have $n_{\mathsf{choices}} = n_{\mathsf{cand}} \cdot n_{\mathsf{cand}}$. A 1 at position (i,j) indicates that the voter prefers candidate c_i over c_j and a 0 indicates that she prefers c_j over c_i. In the standard case, we do not allow two candidates to be tied. Furthermore, for such a ballot to be valid, we require it to be transitive, i.e. if a voter prefers a candidate c_1 over c_2 and c_2 over c_3, then we expect her to prefer candidate c_1 over c_3.

The corresponding choicespace C is given by

$$C_{\mathsf{Condorcet}} = \Big\{ A \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{cand}}} \Big| \forall i,j,k \in [1, n_{\mathsf{cand}}] : \\ i \neq j \Longrightarrow A_{ij} + A_{ji} = 1 \ \ \wedge \ \ A_{ij} = A_{jk} = 1 \Rightarrow A_{ik} = 1 \Big\}$$

As indicated in Figures 3 and 7, computing the Ballot SNARKs for elections with above choice space in Kryvos is comparable to the computation of the other presented Ballot SNARKs and thus feasible. We note that - without the use of SNARKs - proving well-formedness of the ballots for Condorcet-voting in zero-knowledge is complex (see, for example, [67]).

Ballots are then aggregated by adding all n_v matrices. The designated tallier then computes and publishes the Smith set. It is used as public input for the SNARK. The generated SNARK proof allows for verifying that no candidate outside of the Smith set wins against any candidate inside the Smith set and that the set of winners is minimal. Interestingly, the naive approach, using the Smith set as input and simply checking that no candidate outside the Smith set wins against any candidate inside the Smith set, is not secure. The talliers can increase the input set by additional candidates that win against all other candidates outside the Smith set. The checks will still pass, but the input set is not minimal, and thus by definition not the Smith set. Instead, we carefully designed another algorithm for the computation of the Smith set as an arithmetic circuit. We iteratively build the Smith set over multiple rounds such that in each round, candidates that win against any candidate in the current set, is also added to the possible Smith set. For details about the algorithm, see [39]. As indicated by the Benchmarks given in Figures 2 and 8, the evaluation is performed efficiently in Kryvos.

Majority Judgment is a voting system in which the voter grades each candidate independently from a predefined set of grades [71]. There are two main variants for the evaluation, First, one can output for each candidate her median grade (called "Median Grade" in the benchmarks in Section 4). Second, if one wants to get one winning candidate, one can find the candidate with the best median grade. If no such unique candidate exists, this evaluation method (called "Full" in the benchmarks in Section 4) is carried out over multiple rounds. In each round, the best obtained median grade is computed. All candidates that currently have a worse median grade are eliminated. Then, one vote to this median grade is removed for every remaining candidate. For the next round, the median grades are updated accordingly to the removal of the single vote. This procedure is repeated until only one candidate is left, who is declared as winner of the election.

Using this description, the algorithm is highly inefficient due to the fact that the number of rounds is based on the number of voters (since in each round, only one vote is removed). One can, however, as described in [71], use an alternative algorithm that only scales in the number of candidates and grades, and not in the number of voters. The algorithm still computes the same winner as the algorithm above does. Even more, this algorithm allows for aggregation of the votes. For this, the ballots are a matrix of the candidates and grades, where entry A_{ij} is set to one if candidates i receives grade j, and zero otherwise. This is captured by the following choicespace:

$$C_{\mathsf{MajorityJudgement}} = \Big\{ A \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{grades}}} \Big| \\ \forall i \in [1, n_{\mathsf{cand}}], j \in [1, n_{\mathsf{grades}}] : \\ A_{ij} \in \{0,1\} = 1 \\ \wedge \forall i \in [1, n_{\mathsf{cand}}] : \sum_{j \in [1, n_{\mathsf{grades}}]} A_{ij} = 1 \Big\}$$

A comparison between Kryvos and [28] is given in Section 6.

F Optimizations and Insights for IRV

In this section we give some more insights on our implementation of NSW-IRV for Kryvos and the benchmarks we obtained.

Tie Breaking in NSW-IRV. The version of IRV that is used for the New South Wales Legislative Assembly election [2] (NSW-IRV), includes a sophisticated tie-breaking mechanism: If, in any evaluation round r, two (or more) candidates are tied for elimination, then both candidates are compared based on the ballots from round $r-1$. The candidate that was ranked first by the least number of votes in that round is then eliminated. If both candidates are still tied in that round, then the process is repeated with round $r-2$, and so on. In rare cases were candidates are tied in all previous rounds, a lot decides who is eliminated. We model these rare cases where the ties are broken by lot in the deterministic function for the evaluation by using additional input: For each round, each candidate receives a public value, which is then used to break ties.

Optimizing the CRS for IRV. If implemented naively, each of the statements (a), (b), and (c) from Section 4.3 requires its own Groth16 SNARK instance and hence own CRS. Note, however, that all of these Groth16 SNARK instances are for quite similar functions: they first check that the secret input w is a valid opening for the public commitment c by re-computing that commitment, and then also show that w has specific properties. Re-computing the commitment actually accounts for almost the full size of the CRS, whereas performing any additional checks on w barely affects the size of the CRS. Hence, we can use the following optimization: we consider a Groth16 SNARK for a function that takes as public input not just the commitment c but also an additional flag $\mathit{flag} \in \mathbb{N}$. This Groth16 SNARK instance allows for creating a single proof that shows that, on the one hand, the secret input w is an opening for c, and on the other hand, depending on flag, that w fulfills various additional properties. If n_{choices} is not a multiple of N, we could also let the last commitment have a smaller slot size than the previous ones. However, then each voter would need an additional $\mathsf{CRS}_{\mathsf{EK}}$ to create a proof for (b). Thus, if n_{choices} is not a multiple of N, we artificially increase the slot size of the last commitment in order to be able to use a single CRS for the ballot SNARK.

Comparison of our Setup with the Setup from [1] We have given a comparison of our benchmarks for the evaluation of NSW-IRV with the benchmarks from [1] in Table 3. We now briefly discuss the setup that [1] used for obtaining their benchmarks. The benchmarks of [1] are taken directly from [1] which uses a setup comparable to the one we used for Kryvos, consisting of an Intel i7-6770HQ, 2.6 GHz, with 4 cores (8 threads) and 32 GB RAM. While the benchmarks of Kryvos are obtained using a single core, [1] does not state whether a single core or multiple cores were used. Note that the timings given for [1] are actually lower bounds since they exclude the runtime required for computing some of the NIZKPs during the tallying phase.

Specialized ZKPs for Ballot Well-Formedness for IRV. As described in Section 4, for Instant-Runoff Voting, we make use of the choice space for single-vote. This allows for using a specialized ZKP [68] for showing the ballot well-formedness instead on relying on SNARK proofs. In this section, we want to explore this possibility.

Specialized ZKPs for C_{single} usually work as follows. The ZKP consists of various sub zero-knowledge proofs that allow for testing of one specific plaintext value of a ciphertext (or commitment):

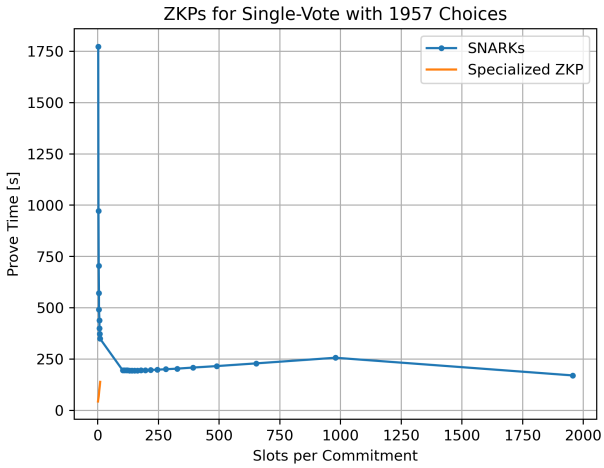

Fig. 9: Comparison between specialized ZKPs and SNARK proofs for IRV and C_{single} with $n_{\mathsf{cand}} = 6$ and thus $n_{\mathsf{choices}} = 1{,}957$. The benchmarks show the runtime to create a complete proof of ballot well-formedness without the use of parallelization.

$$R^{\text{value}}_{x,\mathsf{pk}} = \{(c, r) \mid c = \mathsf{Com}_{\mathsf{pk}}(x, r)\}$$

These sub ZKPs are then combined via an OR protocol. In the typical setting with $N = 1$, i.e., one commitment contains one value, the ZKP is constructed as follows:

- For each commitment $c_i, i \in [1, n_{\mathsf{cand}}]$: $OR(c_i \in L_{R^{\text{value}}_{0,\mathsf{pk}}}, c_i \in L_{R^{\text{value}}_{1,\mathsf{pk}}})$
- For the aggregated commitment $c := \sum_{i \in [1, n_{\mathsf{cand}}]} : OR(c \in L_{R^{\text{value}}_{0,\mathsf{pk}}}, c \in L_{R^{\text{value}}_{1,\mathsf{pk}}})$

This construction can be extended to support Pedersen vector commitments. For this, the OR protocols have to test each possible combination of the values in the commitment. Therefore, the number of statements that are expressed inside each OR protocol increases, while the overall number of OR protocols decreases. Additionally, increasing the slot size increase the cost handling a commitment, since it contains multiple values and for each an exponentiation and a multiplication needs to be performed.

Figure 9 shows a comparison of the benchmarks for the specialized ZKPs for C_{single} as described above and SNARK proofs for the same relation for different slot sizes. The benchmarks show that the specialized ZKPs are more efficient than SNARK proofs, as long as we use small slot sizes. After very few slots per commitment, the SNARK proofs are more efficient. Thus, one interesting question is to investigate how efficient the SNARK for the evaluation of the result function is. As shown in Figure 6, using commitments with more slots is more efficient than using many commitments with less slots. Figure 10 shows benchmarks for the evaluation of the NSW-IRV result function for 6 candidates using different slot sizes. The benchmarks show that using a low slot size requires much more computation power for the tallying SNARK. Therefore, we recommend using SNARKs to show the ballot well-formedness in order to allow for an efficient tallying SNARK.

G Non-Uniform Distribution for IRV Privacy

The non-uniform distribution in Figure 4 is as follows: the probabilities for the first rank are $(0.3, 0.2, 0.2, 0.12, 0.18)$. If the first rank is candidate 1, the probability that the second rank is candidate 2 is 0.95. Analogously for

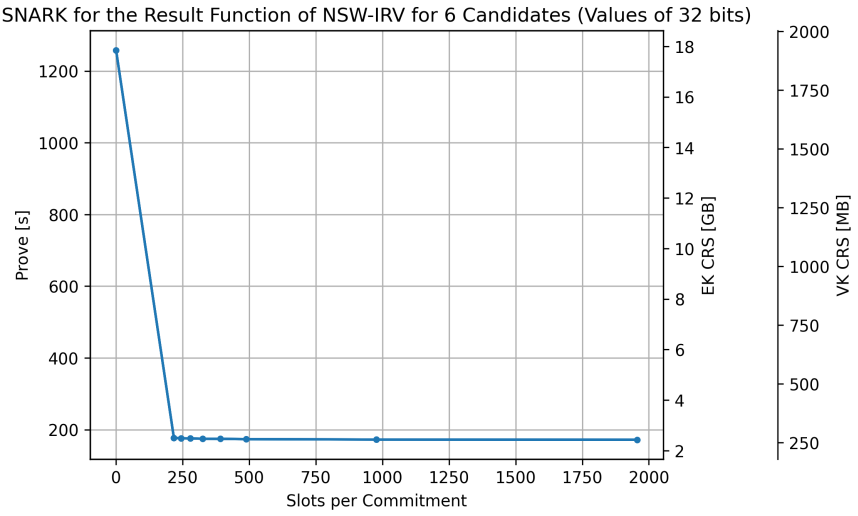

Fig. 10: Benchmarks of SNARK evaluating NSW-IRV.

the pairs of first and second rank (2,1),(4,5),(5,4). All remaining choices are taken uniformly at random. More precisely:

$$\begin{aligned}
\mathsf{Pr}(1{,}2,*,*,*) &= 0.3 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(2{,}1,*,*,*) &= 0.2 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(1{,}3/4/5,*,*,*) &= 0.3 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(2{,}3/4/5,*,*,*) &= 0.2 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(3,*,*,*,*) &= 0.2 \cdot 4^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(4{,}1/2/3,*,*,*) &= 0.12 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(4{,}5,*,*,*) &= 0.12 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(5{,}1/2/3,*,*,*) &= 0.18 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(5{,}4,*,*,*) &= 0.18 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1}.
\end{aligned}$$

H General Computational Model

In this section, we explain the computational model for our security analysis (Section 5) in more detail.

Process. A *process* is a set of probabilistic polynomial-time interactive Turing machines (ITMs, also called *programs*) which are connected via named tapes (also called *channels*). Two programs with a channel of the same name but opposite directions (input/output) are connected by this channel. A process may have external input/output channels, those that are not connected internally. At any time of a process run, one program is active only. The active program may send a message to another program via a channel. This program then becomes active and after some computation can send a message to another program, and so on. Each process contains a *master program*, which is the first program to be activated and which is activated if the active program did not produce output (and hence, did not activate another program). If the master program is active but does not produce output, a run stops.

We write a process π as $\pi = p_1 \| \cdots \| p_l$, where $p_1, \ldots, p_l$ are programs. If π_1 and π_2 are processes, then $\pi_1 \| \pi_2$ is a process, provided that the processes are connectible: two processes are *connectible* if common external channels, i.e., channels with the same name, have opposite directions (input/output); internal channels are renamed, if necessary. A process π where all programs are given the security parameter 1^ℓ is denoted

by $\pi^{(\ell)}$. In the processes we consider, the length of a run is always polynomially bounded in ℓ. Clearly, a run is uniquely determined by the random coins used by the programs in π.

Protocol. A protocol P is modeled via a process, where different participants and components are represented via one ITM each. Typically, a protocol contains a *scheduler* S as one of its participants which acts as the master program of the protocol process (see below). The task of the scheduler is to trigger the protocol participants and the adversary in the appropriate order. For example, in the context of e-voting, the scheduler would trigger protocol participants according to the phases of an election.

The honest programs of the agents of P are typically specified in such a way that the adversary A can corrupt the programs by sending the special message c. Upon receiving such a message, the agent reveals all or some of its internal state to the adversary and from then on is controlled by the adversary. Some agents, such as the scheduler, will typically not be corruptible, i.e., they would ignore c messages. Also, agents might only accept c messages upon initialization, modeling static corruption. This is the case for our security analysis of Kryvos.

We say that an agent a is *honest in a protocol run* r if the agent has not been corrupted in this run, i.e., has not accepted a c message throughout the run. We say that an agent a is *honest* if for all adversarial programs π_{A} the agent is honest in all runs of $\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}}$, i.e., a always ignores all c messages.

Property. A *property* γ *of* P is a subset of the set of all runs of P.[21] By $\neg\gamma$ we denote the complement of γ.

I Privacy Proof

In this section, we prove Theorem 2 which establishes the public and internal privacy levels of Kryvos. These levels can be expressed using the privacy level of the voting protocol with ideal privacy (see Figure 11).

I.1 Overview

Recall that, in order to prove the theorem for the protocol Kryvos with n_v voters, n_t talliers, voting method (C, f_{res}), voting distribution μ, and voter under observation $\mathsf{V}_{\mathsf{obs}}$, we have to show the following two conditions:

- *Internal privacy*: For all $m_0, m_1 \in C$, for all programs π^* of the remaining parties such that at least n_v^h voters and *at least one* tallier are honest in π^* (excluding the voter under observation $\mathsf{V}_{\mathsf{obs}}$), we have that

$$|\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_0) \| \pi^*) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_1) \| \pi^*) \mapsto 1]|$$

  is $\delta^{\text{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\text{complete}})$-bounded as a function of the security parameter ℓ, where f_{complete} is the function that returns the full/complete tally.
- *Public privacy*: For all $m_0, m_1 \in C$, for all programs π^* of the remaining parties such that at least n_v^h voters and *all* talliers are honest in π^* (excluding the voter under observation $\mathsf{V}_{\mathsf{obs}}$), we have that

$$|\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_0) \| \pi^*) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m_1) \| \pi^*) \mapsto 1]|$$

  is $\delta^{\text{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\text{res}})$-bounded as a function of the security parameter ℓ, where f_{res} is the actual (tally-hiding) result function.

We can split up the composition π^* in its honest and its (potentially) dishonest part. Let HV be the set of all honest voters (without the voter under observation) and $\hat{\pi}_{\mathsf{HV}}$ be the composition of their honest programs. Recall that the judge J, the scheduler S, the bulletin board B, and at least one (in case of internal privacy) out of or all (in case of public privacy) n_t talliers are honest, respectively (w.l.o.g., we assume that

[21] Recall that the description of a run r of P contains the description of the process $\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}}$ (and hence, in particular the adversary) from which r originates. Therefore, γ can be formulated independently of a specific adversary.

$\mathcal{I}_{\mathsf{voting}}(n_v, n_v^h, C, f, \mu)$

Parameters:

- Function $f\colon \{0,1\}^* \to \{0,1\}^*$
- Probability distribution μ over C
- Numbers of voters n_v and honest voters n_v^h
- $I \leftarrow \emptyset$ (initially)

On $(\mathsf{init}, \mathsf{honest})$ from S do:

1. $\forall i \in \{1, \ldots, n_v^h\}$: store $m_i \xleftarrow{\mu} C$
2. $I \leftarrow I \cup \{1, \ldots, n_v^h\}$
3. Return $\mathsf{success}$

On $(\mathsf{setChoice}, i, m)$ from S do:

1. If $i \notin \{n_v^h + 1, \ldots, n_v\}$ or $m \notin C$, return $\perp$.
2. Store $m_i \leftarrow m$
3. $I \leftarrow I \cup \{i\}$
4. Return $\mathsf{success}$

On $(\mathsf{compute}, b)$ from S do:

1. If $b = 0$, return $\perp$.
2. Return res $\leftarrow f((m_i)_{i \in I})$

Fig. 11: Ideal voting protocol.

the first tallier T_1 is honest in both cases). Therefore, we denote the honest part for the internal privacy result by

$$\hat{\pi}_{\mathsf{H}} = \hat{\pi}_{\mathsf{J}} \| \hat{\pi}_{\mathsf{B}} \| \hat{\pi}_{\mathsf{S}} \| \hat{\pi}_{\mathsf{T}_1} \| \hat{\pi}_{\mathsf{HV}},$$

and for the public privacy result by

$$\hat{\pi}_{\mathsf{H}} = \hat{\pi}_{\mathsf{J}} \| \hat{\pi}_{\mathsf{B}} \| \hat{\pi}_{\mathsf{S}} \| \hat{\pi}_{\mathsf{T}_1} \| \ldots \| \hat{\pi}_{\mathsf{T}_{n_t}} \| \hat{\pi}_{\mathsf{HV}}.$$

By $\hat{\pi}_{\mathsf{H}}(m)$ we will denote the composition of all honest programs including the program of the voter under observation $\mathsf{V}_{\mathsf{obs}}$, i.e., $\hat{\pi}_{\mathsf{H}}(m) = \hat{\pi}_{\mathsf{H}} \| \hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m)$. All remaining parties are subsumed by the adversarial process π_{A}. This means that we can write $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(m) \| \pi^*$ as $\hat{\pi}_{\mathsf{H}}(m) \| \pi_{\mathsf{A}}$.

In order to prove the result, we use two sequences of games (one for internal and one for external privacy). We fix $m \in C$ and start with Game 0 which is simply the process $\hat{\pi}_{\mathsf{H}}(m) \| \pi_A$. Step by step, we transform Game 0 into Game x which is the composition $\hat{\pi}^x_{\mathsf{H}}(m) \| \pi_A$ for some process $\hat{\pi}^x_{\mathsf{H}}(m)$ and the same adversarial process π_A. Game x will be proven indistinguishable from Game 0 from the adversary's point of view, which means that

$$\left| \Pr[(\hat{\pi}^0_{\mathsf{H}}(m) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}^x_{\mathsf{H}}(m) \| \pi_A) \mapsto 1] \right|$$

is negligible for a fixed $m \in C$ (as a function of the security parameter).

Furthermore, it will be straightforward to show that in Game x for arbitrary $m_0, m_1 \in C$, the distance

$$|\Pr[(\hat{\pi}^x_{\mathsf{H}}(m_0) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}^x_{\mathsf{H}}(m_1) \| \pi_A) \mapsto 1]|$$

is bounded by $\delta^{\mathrm{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\mathrm{complete}})$ (internal privacy) or $\delta^{\mathrm{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\mathrm{res}})$ (public privacy), respectively. The reason is that $\hat{\pi}^x_{\mathsf{H}}(m_0)$ and $\hat{\pi}^x_{\mathsf{H}}(m_1)$ use the ideal voting protocol for voting method $(C, f_{\mathrm{complete}})$ (internal

privacy) or (C, f_{res}), respectively, with n_v^h honest voters. Using the triangle inequality, we can therefore deduce that

$$|\Pr[(\hat{\pi}_{\mathsf{H}}(m_0)\|\pi_A) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}(m_1)\|\pi_A) \mapsto 1]|$$

is bounded by $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{complete}})$ (internal privacy) or $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{res}})$ (public privacy) for all $m_0, m_1 \in C$ (as a function of the security parameter ℓ).

Notation. We write $\hat{\pi}^1_{\mathsf{H}}(m)$ for $\hat{\pi}_{\mathsf{H}}(m)$ and consider $\hat{\pi}^1_{\mathsf{H}}(m)$ as one atomic process (one program) and not as a composition of processes.[22] Now, the original Kryvos program is the process $\hat{\pi}^1_{\mathsf{H}}(m)\|\pi_A$. For both sequences of games below, we describe how to modify the previous protocol so that any (ppt) adversary is not able to distinguish which of these two protocols he is communicating with. For each Game j), we denote the respective honest part of the protocol by $\hat{\pi}^j_{\mathsf{H}}(m)$. Let I_h be the set of indices of honest voters (excluding the voter under observation).

I.2 Internal privacy

We use a sequence of games to prove that the internal privacy level of Kryvos is $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{complete}})$ under the assumptions (P1) and (P2) (see Section 5.3) and that at least one tallier is honest (w.l.o.g., we assume that T_1 is honest). The ultimate goal of the proof is to employ the ideal voting functionality to compute the honest voters' aggregated choice. We exploit the ZK property of the ZKPs, the semantic security of the PKE scheme, and the hidingness of the commitment to simulate the honest participants in such a way that the honest participants (in particular the voters) do no longer create (and know) the individual honest votes. Instead, the honest participants only use the (aggregated) honest result obtained from the ideal voting functionality. Therefore, even if the adversary is able to control all honest participants, he only gets to know the aggregated output from the ideal voting functionality. At the same time, we show that all modifications are indistinguishable from the adversary's perspective. This proves that the internal privacy level of Kryvos is negligibly close to the one of the ideal voting functionality which is $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{complete}})$. More precisely, our proof works as follows:

Game 0 $\hat{\pi}^0_{\mathsf{H}}(m)$ is the original Kryvos protocol.

Game 1 In $\hat{\pi}^1_{\mathsf{H}}(m)$, we modify the specification of the (honest) trustee T_1 as follows. If the ciphertext e_1^i of an arbitrary honest voter V_i decrypts to an invalid opening, then T_1 aborts. Apart from this modification, the specification of T_1 does not change.

Due to the correctness of the public-key encryption scheme $\mathcal{E}$, Game 0 and Game 1 are perfectly indistinguishable from the adversary's perspective.

Game 2 In $\hat{\pi}^2_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i ($i \in I_h$) and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. Each V_i ($i \in I_h$) and $\mathsf{V}_{\mathsf{obs}}$ encrypt (the dummy message) 0^w under the honest trustee's, i.e., T_1's, public key pk_1; precisely: $e_1^i \leftarrow \mathsf{Enc}(\mathsf{pk}_1, 0^w)$ (where w is the fixed plaintext size). Apart from this modification, the specification of V_i ($i \in I_h$) and $\mathsf{V}_{\mathsf{obs}}$ does not change. Furthermore, we modify the specification of the honest trustee T_1 as follows. Instead of decrypting the honest voters' ciphertexts e_1^i (which now encrypt useless dummy messages), the honest trustee T_1 receives the honest voters' opening values from the honest voters internally (inside of the simulator which subsumes all honest parties).

Due to the IND-CCA-security of the public-key encryption scheme $\mathcal{E}$ and the removal of ciphertext duplicates, Game 1 and Game 2 are computationally indistinguishable from the adversary's perspective.

Game 3 In $\hat{\pi}^3_{\mathsf{H}}(m)$, we modify the specification of the (honest) voting authority Auth as follows. Recall that the voting authority runs the setup algorithm Setup of the SNARK Π_{ballot}, which is used to prove well-formedness of the voters' commitments, to obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{\mathsf{ballot}}, \tau_{\mathsf{ballot}})$. Now, instead of keeping the trapdoor secret, the authority internally (inside of the simulator) forwards the trapdoor τ_{ballot} to the voter under observation $\mathsf{V}_{\mathsf{obs}}$ and all honest voters V_i

[22] This is w.l.o.g. since every (sub-)process can be simulated by a single program.

$(i \in I_h)$.

It is clear that Game 2 and Game 3 are perfectly indistinguishable from the adversary's perspective.

Game 4 In $\hat{\pi}^4_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. Instead of using the common reference string $\mathsf{CRS}_{\mathsf{ballot}}$ to run the prover algorithm $\mathsf{Prove}(R_{\mathsf{ballot}}, \mathsf{CRS}_{\mathsf{ballot}}, x, w)$, the honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ use the trapdoor τ_{ballot} (received by Auth in the previous game) to run the simulator algorithm $\mathsf{Sim}(R_{\mathsf{ballot}}, \tau_{\mathsf{ballot}}, x)$ of the SNARK Π_{ballot}, where (x, w) denotes the respective voter's pair of public input and witness. Apart from these modifications, the specification of V_i $(i \in I_h)$ and $\mathsf{V}_{\mathsf{obs}}$ does not change.

Game 3 and Game 4 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK Π_{ballot}.

Game 5 In $\hat{\pi}^5_{\mathsf{H}}(m)$, we modify the specification of the honest trustee T_1 as follows. The trustee T_1 simulates the NIZKP of correct decryption Π_{dec} (if any).

Due to the perfect zero-knowledge property of the NIZKP Π_{dec}, Game 4 and Game 5 are perfectly indistinguishable.

Game 6 In $\hat{\pi}^6_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. The voter under observation $\mathsf{V}_{\mathsf{obs}}$ computes $m^i \xleftarrow{\mu} C$ for all honest voters $i \in I_h$, sets $\hat{m}_h \leftarrow \sum_{i \in I_h} m^i$, and runs the voting algorithm with "choice" $\hat{m} \leftarrow \hat{m}_h + m$ instead of her choice m. This means that the "choice" contained in $\mathsf{V}_{\mathsf{obs}}$'s ballot is an aggregation of all honest voters' choices plus the one of the voter under observation. At the same time, all honest voters V_i $(i \in I_h)$ run the voting algorithm with dummy "choice" 0.[23]

Because the commitment scheme is unconditionally hiding, ciphertext duplicates are removed, and full-threshold secret sharing is employed, Game 5 and Game 6 are computationally indistinguishable from the adversary's perspective.

Game 7 In $\hat{\pi}^7_{\mathsf{H}}(m)$, we modify the specification of the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. The voter under observation $\mathsf{V}_{\mathsf{obs}}$ invokes the ideal voting functionality $\mathcal{I}_{\mathsf{voting}}$ (see Figure 11) to obtain the honest voters' aggregated choice $\hat{m} \leftarrow \mathcal{I}_{\mathsf{voting}}(n^h_v + 1, n^h_v + 1, C, f_{\mathrm{complete}}, \mu')$, where the voting distribution μ' always returns m for the first (honest) voter and equals μ for the remaining n^h_v (honest) voters.

Because the ideal voting functionality creates $\hat{m}$ in the same way as $\mathsf{V}_{\mathsf{obs}}$ did in the previous step, Game 6 and Game 7 are perfectly indistinguishable from the adversary's perspective.

In Game 7, we have that for arbitrary $m_0, m_1 \in C$, the distance

$$\left|\Pr[(\hat{\pi}^7_{\mathsf{H}}(m_0) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}^7_{\mathsf{H}}(m_1) \| \pi_A) \mapsto 1]\right|$$

is bounded by $\delta^{\mathrm{ideal}}_{(n_v, n^h_v, \mu)}(C, f_{\mathrm{complete}})$. Since we have proved above that Game 1 (the original Kryvos protocol) and Game 7 are computationally indistinguishable (under the assumptions made, in particular that T_1 is honest), we can conclude that

$$|\Pr[(\hat{\pi}_{\mathsf{H}}(m_0) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}(m_1) \| \pi_A) \mapsto 1]|$$

is bounded by $\delta^{\mathrm{ideal}}_{(n_v, n^h_v, \mu)}(C, f_{\mathrm{complete}})$ for all $m_0, m_1 \in C$ (as a function of the security parameter ℓ).

I.3 Public privacy

We use a sequence of games to prove that the external privacy level of Kryvos is $\delta^{\mathrm{ideal}}_{(n_v, n^h_v, \mu)}(C, f_{\mathrm{res}})$ under the assumptions (P1) and (P2) (see Section 5.3) and that all talliers are honest (w.l.o.g., we assume that there exists a single T and that this trustee is honest). The ultimate goal of the proof is to employ the ideal voting functionality which takes as input the dishonest voters' choices to compute the (tally-hiding) result

[23] Recall that all voters' choices are homomorphically aggregated in the tallying phase. Therefore, it is inconsequential how the honest voters' aggregated choice is split among the homomorphic commitments.

function f_{res}. We exploit the ZK property of the ZKPs, the semantic security of the PKE scheme, and the hidingness of the commitment to simulate the honest participants in such a way that the honest participants (in particular the voters) do no longer create (and know) the individual honest votes. Instead, the honest participants only use the tally-hiding result obtained from the ideal voting functionality. Therefore, even if the adversary is able to control all honest participants, he only gets to know the tally-hiding output from the ideal voting functionality. At the same time, we show that all modifications are indistinguishable from the adversary's perspective. This proves that the external privacy level of Kryvos is negligibly close to the one of the ideal voting functionality which is $\delta^{\text{ideal}}_{(n_v,n_v^h,\mu)}(C, f_{\text{res}})$. More precisely, our proof works as follows:

Game 0 $\hat{\pi}^0_{\mathsf{H}}(m)$ is the original Kryvos protocol.

Game 1 In $\hat{\pi}^1_{\mathsf{H}}(m)$, we modify the specification of the (honest) trustee T as follows. If the ciphertext e^i of an arbitrary honest voter V_i decrypts to an invalid opening, then T aborts. Apart from this modification, the specification of T does not change.

Due to the correctness of the public-key encryption scheme $\mathcal{E}$, Game 0 and Game 1 are perfectly indistinguishable from the adversary's perspective.

Game 2 In $\hat{\pi}^2_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. Each V_i $(i \in I_h)$ and $\mathsf{V}_{\mathsf{obs}}$ encrypt (the dummy message) 0^w under the trustee's public key pk_1; precisely: $e^i \leftarrow \mathsf{Enc}(\mathsf{pk}, 0^w)$ (where w is the fixed plaintext size). Apart from this modification, the specification of V_i $(i \in I_h)$ and $\mathsf{V}_{\mathsf{obs}}$ does not change. Furthermore, we modify the specification of the trustee T as follows. Instead of decrypting the honest voters' ciphertexts e^i (which now encrypt useless dummy messages), the trustee T receives the honest voters' opening values from the honest voters internally (inside of the simulator which subsumes all honest parties).

Due to the IND-CCA-security of the public-key encryption scheme $\mathcal{E}$ and the removal of ciphertext duplicates, Game 1 and Game 2 are computationally indistinguishable from the adversary's perspective.

Game 3 In $\hat{\pi}^3_{\mathsf{H}}(m)$, we modify the specification of the honest trustee T as follows. The trustee T simulates the NIZKP of correct decryption $\mathit{\Pi}_{\mathsf{dec}}$ (if any).

Due to the perfect zero-knowledge property of the NIZKP $\mathit{\Pi}_{\mathsf{dec}}$, Game 2 and Game 3 are perfectly indistinguishable.

Game 4 In $\hat{\pi}^4_{\mathsf{H}}(m)$, we modify the specification of the (honest) voting authority Auth as follows. Recall that the voting authority runs the setup algorithm Setup of the SNARK $\mathit{\Pi}_{\mathsf{f}_{\text{res}}}$ which is used to prove correctness of the final (tally-hiding) result to obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{\mathsf{f}_{\text{res}}}, \tau_{\mathsf{f}_{\text{res}}})$. Now, instead of keeping the trapdoor secret, the authority internally (inside of the simulator) forwards the trapdoor $\tau_{\mathsf{f}_{\text{res}}}$ to the trustee T.

It is clear that Game 3 and Game 4 are perfectly indistinguishable from the adversary's perspective.

Game 5 In $\hat{\pi}^5_{\mathsf{H}}(m)$, we modify the specification of the trustee T as follows. Instead of using the common reference string $\mathsf{CRS}_{\mathsf{f}_{\text{res}}}$ to run the prover algorithm $\mathsf{Prove}(R_{\mathsf{f}_{\text{res}}}, \mathsf{CRS}_{\mathsf{f}_{\text{res}}}, x, w)$, the trustee uses the trapdoor $\tau_{\mathsf{f}_{\text{res}}}$ (received by Auth in the previous game) to run the simulator algorithms $\mathsf{Sim}(R_{\mathsf{f}_{\text{res}}}, \tau_{\mathsf{f}_{\text{res}}}, x)$, where (x, w) denotes the trustee's pair of public input and witness. Apart from these modifications, the specification of T does not change.

Game 4 and Game 5 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK $\mathit{\Pi}_{\mathsf{f}_{\text{res}}}$.

Game 6 In $\hat{\pi}^6_{\mathsf{H}}(m)$, we modify the specification of the (honest) voting authority Auth as follows. Recall that the voting authority runs the setup algorithm Setup of the SNARK $\mathit{\Pi}_{\mathsf{ballot}}$, which is used to prove well-formedness of the voters' commitments, to obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{\mathsf{ballot}}, \tau_{\mathsf{ballot}})$. Now, instead of keeping the trapdoor secret, the authority internally (inside of the simulator) forwards the trapdoor τ_{ballot} to the voter under observation $\mathsf{V}_{\mathsf{obs}}$ and all honest voters V_i $(i \in I_h)$.

It is clear that Game 5 and Game 6 are perfectly indistinguishable from the adversary's perspective.

Game 7 In $\hat{\pi}^7_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. Instead of using the common reference string $\mathsf{CRS}_{\mathsf{ballot}}$ to run the prover algorithm $\mathsf{Prove}(R_{\mathsf{ballot}}, \mathsf{CRS}_{\mathsf{ballot}}, x, w)$, the honest voters V_i $(i \in I_h)$ and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ use the trapdoor τ_{ballot} (received by Auth in the previous game) to run the simulator algorithm

$\mathsf{Sim}(R_{\mathsf{ballot}}, \tau_{\mathsf{ballot}}, x)$ of the SNARKΠ_{ballot}, where (x, w) denotes the respective voter's pair of public input and witness. Apart from these modifications, the specification of V_i ($i \in I_h$) and $\mathsf{V}_{\mathsf{obs}}$ does not change.

Game 6 and Game 7 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK Π_{ballot}.

Game 8 In $\hat{\pi}^8_{\mathsf{H}}(m)$, we modify the specification of the honest trustee T as follows. The trustee T sets m_d to be the (aggregated) dishonest choices (observe that T can decrypt/open all dishonest voters' valid choices). For all $i \in I_h$, the trustee computes $\hat{m}^i \xleftarrow{\mu} C$. Then, T sets $\hat{m}_h \leftarrow \sum_{i \in I_h} \hat{m}^i$ and $\hat{m} \leftarrow \hat{m}_h + m$, and returns the final result $\mathsf{res} \leftarrow f_{\mathrm{res}}(\hat{m} \cup m_d)$.

Because the commitment scheme is unconditionally hiding, Game 7 and Game 8 are perfectly indistinguishable from the adversary's perspective.

Game 9 In $\hat{\pi}^9_{\mathsf{H}}(m)$, we modify the specification of all honest voters V_i ($i \in I_h$) and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ as follows. All honest voters V_i ($i \in I_h$) and the voter under observation $\mathsf{V}_{\mathsf{obs}}$ run the voting algorithm with dummy "choice" 0.

Because the commitment scheme is unconditionally hiding, Game 8 and Game 9 are perfectly indistinguishable from the adversary's perspective.

Game 10 In $\hat{\pi}^1_{\mathsf{H}}0(m)$, we modify the specification of the trustee T as follows. Trustee T invokes the ideal voting functionality to obtain $\mathsf{res} \leftarrow \mathcal{I}_{\mathsf{voting}}(n_v, n_v^h + 1, C, f_{\mathrm{res}}, \mu')$, where the voting distribution μ' always returns m for the first (honest) voter and equals μ for the remaining n_v^h (honest) voters, and where the dishonest choices are extracted from m_d.

Because the ideal voting functionality creates res in the same way as T did in the previous step, Game 9 and Game 10 are perfectly indistinguishable from the adversary's perspective.

In Game 10, we have that for arbitrary $m_0, m_1 \in C$, the distance

$$\left|\Pr[(\hat{\pi}^{10}_{\mathsf{H}}(m_0) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}^{10}_{\mathsf{H}}(m_1) \| \pi_A) \mapsto 1]\right|$$

is bounded by $\delta^{\mathrm{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\mathrm{res}})$. Since we have proved above that Game 1 (the original Kryvos protocol) and Game 10 are indistinguishable (under the assumptions made, in particular that all talliers are honest), we can conclude that

$$|\Pr[(\hat{\pi}_{\mathsf{H}}(m_0) \| \pi_A) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}(m_1) \| \pi_A) \mapsto 1]|$$

is information-theoretically bounded by $\delta^{\mathrm{ideal}}_{(n_v, n_v^h, \mu)}(C, f_{\mathrm{res}})$ for all $m_0, m_1 \in C$.

J Verifiability

In this section, we recall the formal definition of the verifiability notion from [75] and of the goal $\gamma(\varphi)$ from [20] which we use to analyze verifiability of Kryvos (see Section 5.2).

Verifiability definition. In the following definition of verifiability (which is a bit shortened for brevity of presentation), we denote by $\Pr[(\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}})^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ the probability that a run of the protocol along with an adversary π_{A} (and a security parameter ℓ) produces a run which is not in γ but in which J (nevertheless) returns accept. This probability should be negligible.

Definition 4 (Verifiability [75]). *We say that* a goal γ is verifiable by the judge J in a protocol P *if for all adversaries* π_{A}*, the probability* $\Pr[(\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}})^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ *is negligible as a function of* ℓ*.*

Goal $\gamma(\varphi)$. The goal $\gamma(\varphi)$ is defined as follows for an arbitrary voting method (C, f_{res}). The parameter φ is a Boolean formula to describe which protocol participants are assumed honest. Let I_h and I_d denote the set of honest and dishonest voters, respectively, in a given protocol run. Then, $\gamma(\varphi)$ consists of all those runs of a voting protocol P where either

- φ is false (e.g., the adversary corrupted a voter that is assumed to be honest), or
- φ holds true and there exist (valid) dishonest choices $(m_i)_{i \in I_d}$ such that the election result equals $f_{\mathrm{res}}((m_i)_{i \in I_h \cup I_d})$, where $(m_i)_{i \in I_h}$ are the honest voters' choices.

K Verifiability Proof

In this section, we prove Theorem 1 (Verifiability) from Section 5.2. The judging procedure of Kryvos is very simple. The judge reads all data from the bulletin board B and accepts the run if and only if all NIZKPs on B are valid (and only eligible voters voted, at most once): the voters' NIZKPs Π^i_{ballot} to prove well-formedness of ballots, the talliers' NIZKP $\Pi_{f_{\mathrm{res}}}$ to prove correctness of the final result, and (if any) the talliers' NIZKP Π_{dec} to prove that a voters' ciphertext does not decrypt to a valid opening of her commitments.

Recall that, at a high level, we need to prove the following property. If the final result res does not equal $f_{\mathrm{res}}((m^i)_{i\in I})$, where $(m^i)_{i\in I}$ consists of all honest voters' choices and at most one (valid) choice per dishonest voter, then the judge J rejects the result.

Essentially, Theorem 1 follows from the NIZKPs employed in Kryvos (recall Appendix B.1 for the precise definition of the respective relations).

In what follows, we denote the set of indices of honest voters by I_h and the set of indices by dishonest voters by I_d. Let $(b_i)_{i\in I_{\mathsf{valid}}}$ be the set of ballots that have not been discarded prior to the homomorphic aggregation at the end of voting phase.

The first Lemma states that honest ballots cannot be discarded prior to the homomorphic aggregation and that there exists at most one ballot per dishonest voter in the remaining (non-discarded) ballots.

Lemma 1. *Let I_{valid} be defined as above. Then, we have that $I_h \subseteq I_{\mathsf{valid}}$ and $I_d \supseteq (I_{\mathsf{valid}} \setminus I_h)$ holds true (with overwhelming probability in the security parameter ℓ).*

Proof. Observe that $i \in I_{\mathsf{valid}}$ holds true for a given ballot b_i if and only if the test executed by the bulletin board B for incoming ballots (specified at the end of the voting phase, cf. Sec. 3.1) was successful *and* no trustee T_k published a valid NIZKP proving that e^i_k decrypts to an invalid opening.

Now, to see that $I_h \subseteq I_{\mathsf{valid}}$ holds true, observe that the voter's program defined in Sec. 3.1, which each honest voter runs, ensures that the relation of well-formedness R^C_{ballot} is satisfied. Therefore, each honest voter's proof Π_{ballot} is valid. Furthermore, due to the IND-CCA2-security of the PKE encryption scheme (and because the number of voters is polynomially bounded), the probability that an honest voter creates a partially identical ballot to a different voter's one is negligible (in the security parameter). This ensures that each honest voter's ballot b_i is appended by the bulletin board (which we assume to be honest, see assumption (V2) in Sec. 5.2) to the list of ballots $\mathbf{b}$. Since the NIZKP of correct decryption Π_{dec} is sound and the (honest) voter's program guarantees that each ciphertext e^i_k encrypts a correct opening for the respective commitments, a (possibly dishonest) trustee T_k is not able to create a valid NIZKP for claiming that an honest voter's ciphertext e^i_k decrypts to an invalid opening. Hence, $I_h \subseteq I_{\mathsf{valid}}$ follows (with overwhelming probability).

The fact that $I_d \supseteq (I_{\mathsf{valid}} \setminus I_h)$ holds true follows from the checks (precisely, the eligibility and re-voting check) executed by the bulletin board B before it adds a ballot to $\mathbf{b}$.

The following Lemma states that all ballots which are not discarded prior to the homomorpic aggregation are well-formed (i.e., "contain" a valid choice from C) and contain commitments with sufficiently small randomness.

Lemma 2. *Let I_{valid} be defined as above. Then (with overwhelming probability in the security parameter ℓ), for all $i \in I_{\mathsf{valid}}$, there exists $(r^{i,l}_k)_{k\in[n_t],l\in[n_{\mathsf{tuples}}]}$ and $m^i = (m^{i,j})_{j\in[n_{\mathsf{choices}}]} \in C$ such that for all $j \in [n_{\mathsf{choices}}]$, there exists $(m^{i,j}_k)_{k\in[n_t]}$ such that*

1. $\forall j \in [n_{\mathsf{choices}}]\colon \sum_{k=1}^{n_t} m^{i,j}_k \bmod q = m^{i,j}$, *and*
2. $\forall k \in [n_t]\ \forall l \in [n_{\mathsf{tuples}}]\colon$
   $c^{i,l}_k = \mathsf{Com}((m^{i,(l-1)\cdot N+1}_k, \ldots, m^{i,l\cdot N}_k); r^{i,l}_k)$.

Proof. Recall that the bulletin board (which we assume to be honest) verifies whether each incoming ballot b_i contains valid NIZKP Π_{ballot}, and rejects b_i if this is not the case. Therefore, if $i \in I_{\mathsf{valid}}$ for a given b_i, it follows that b_i contains valid ZKP Π_{ballot}. Hence, fact (1) and (2) follow from the computational soundness of Π_{ballot}.

The following Lemma states that the homomorpically aggregated commitments correctly open to the total numbers of votes per choice, as determined by I_{valid}.

Lemma 3. *Let I_{valid} be defined as above. Let $(c_k^{\perp,l})_{k\in[n_t],l\in[n_{\mathsf{tuples}}]}$ be the input commitments to the tallying phase. Then (with overwhelming probability in the security parameter ℓ), for all $l \in [n_{\mathsf{tuples}}]$, there exists $(m^{i,l})_{i\in I_{\mathsf{valid}}}$ and $r^{\perp,l}$ such that*

$$c^{\perp,l} = \mathsf{Com}(\sum_{i\in I_{\mathsf{valid}}} m^{i,l}; r^{\perp,l}).$$

Proof. This fact follows from Lemma 2 and the homomorpic property of the commitment scheme.

Since the talliers' SNARK $\Pi_{f_{\mathrm{res}}}$ (correct result) is computationally sound, we can conclude from Lemma 3 that if the final result does not equal $f_{\mathrm{res}}((m^i)_{i\in I_{\mathsf{valid}}})$, then the judge J rejects the result (with overwhelming probability). Hence, Theorem 1 follows from Lemma 1.