Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting⋆

Fabian Hertel[1], Nicolas Huber[2], Jonas Kittelberger[3], Ralf Küsters[2], Julian Liedtke[2], and Daniel Rausch[2]

[1] University of Stuttgart st151599@stud.uni-stuttgart.de
[2] University of Stuttgart {firstname.lastname}@sec.uni-stuttgart.de
[3] University of Stuttgart jonas.kittelberger@gmail.com

Abstract. Modern electronic voting systems (e-voting systems) are designed to achieve a variety of security properties, such as verifiability, accountability, and vote privacy. Some of these systems aim at so-called *tally-hiding*: they compute the election result, according to some result function, like the winner of the election, without revealing any other information to any party. In particular, if desired, they neither reveal the full tally consisting of all (aggregated or even individual) votes nor parts of it, except for the election result, according to the result function. Tally-hiding systems offer many attractive features, such as strong privacy guarantees both for voters and for candidates, and protection against Italian attacks. The Ordinos system is a recent provably secure framework for accountable tally-hiding e-voting that extends Helios and can be instantiated for various election methods and election result functions. So far, practical instantiations and implementations for only rather simple result functions (e.g., computing the k best candidates) and single/multi-vote elections have been developed for Ordinos.

In this paper, we propose and implement several new Ordinos instantiations in order to support Borda voting, the Hare-Niemeyer method for proportional representation, multiple Condorcet methods, and Instant-Runoff Voting. Our instantiations, which are based on suitable secure multi-party computation (MPC) components, offer the first tally-hiding implementations for these voting methods. To evaluate the practicality of our MPC components and the resulting e-voting systems, we provide extensive benchmarks for all our implementations.

Keywords: E-Voting · Tally-Hiding · MPC · Accountability · Privacy · Implementations · Benchmarks.

1 Introduction

There is a multitude of different voting methods ranging from relatively simple ones, such as plurality/single-choice voting, to more complex ones, such as

⋆ This work was in part funded by the Deutsche Forschungsgemeinschaft (DFG) KU 1434/11-1 and the Center for Integrated Quantum Science and Technology (IQST).

cumulative voting with multiple votes as well as preferential elections and multi-round votings. Also, there are many different result functions used in elections. For example, one might be interested only in the winner of the election (e.g., for presidential elections), the number of seats of parties in a parliament, or the k best or worst candidates (ranked or not ranked), e.g., to fill positions or to decide who moves on to a runoff election.

Tally-Hiding. A desirable and strong security property that several e-voting systems try to achieve is *tally-hiding* [1,2,3,4,5,6,7]. A tally-hiding system computes and publishes the election result, according to some result function, e.g., the winner of an election, without revealing any other information to any party. In particular, if desired, except for the election result itself, they neither reveal the full tally consisting of all (aggregated or even individual) votes nor parts of it, such as the winner of an election round or the number of votes of a candidate. Even internal parties, like trustees, should not learn anything besides the result. In essence, tally-hiding is a strong form of privacy that not just avoids leaking the content of individual ballots but rather avoids leaking any unnecessary information altogether. As discussed, e.g., in [6], tally-hiding is an attractive feature in many situations: it prevents introducing biases in voters during multi-round elections, losing candidates are not unnecessarily embarrassed due to a (potentially very low) number of votes, mandates of winning candidates remain strong even if they won only by a small margin, tally-hiding helps prevent gerrymandering since the exact vote distributions remain hidden, and it also prevents Italian attacks. To retain trust in the overall result, tally-hiding elections, like other elections, have to provide *verifiability*: Each voter must be able to verify that her vote was counted correctly and that the overall result is correct. Moreover, it should not only be possible to verify the result, but, if verification fails, it should be possible to identify misbehaving parties and hold them accountable for the failure. This stronger form of verifiability is called *accountability* [8].

There are also several systems that achieve what we call *partial tally-hiding*, e.g., [9,10,11,12,13,14]. These systems generally focus on solving specific issues, most notably Italian attacks, and achieve this by hiding only those parts of the tally that are critical for the issue at hand, e.g., the individual votes. However, they still reveal certain information besides the election result, e.g., the losers of intermediate election rounds. In this work, we focus on (full) tally-hiding where nothing but the final result is revealed.

Current State. As mentioned, several e-voting systems have been designed to be tally-hiding. These systems generally follow the same underlying idea, namely, using a publicly verifiable secure multi-party computation (MPC) protocol to compute the election result from an encrypted tally. From a theoretical point of view, it is clear that essentially arbitrary functions, and thus election results, can be computed in this way. The main challenge lies in constructing an efficient MPC tallying component. For example, in recent work Cortier et al. [7] tackles, among others, this challenge by proposing tally-hiding MPC components (for single-vote elections, majority judgement, Condorcet-Schulze, and STV) and studying their asymptotic complexity.

So far, there are only very few (fully) tally-hiding protocols that have been implemented, benchmarked, and shown to be viable. Specifically, Canard et. al. [5] proposed and implemented a tally-hiding protocol for majority judgement that is shown to achieve practical performance. In [6], Küsters et. al. proposed the general Ordinos framework for provably secure accountable tally-hiding e-voting. They also designed and implemented several Ordinos instantiations and demonstrated their practicality. Specifically, they considered the following highly relevant but relatively simple result functions for single/multi-vote elections: computing the k candidates with the highest/lowest number of votes, computing all candidates that pass a certain threshold of votes, a combination of both, with or without revealing the ranking among the winners, and with or without revealing the number of votes the candidates in question have obtained.

Our Goal. In this work, we want to extend the state-of-the-art by implementing and benchmarking MPC components for tally-hiding elections also for many other voting methods. To this end, we build on the Ordinos system, since, as mentioned, Ordinos provides a general provably secure framework for accountable (and hence, verifiable) tally-hiding elections, and because we can base our work on the practical instantiations of Ordinos that have been proposed before.

Our Contributions. We propose and implement several new instantiations of Ordinos for complex election types and result functions. Specifically, we propose MPC components for Borda voting, the Hare-Niemeyer method for proportional representation, Instant-Runoff Voting, and multiple versions of Condorcet (plain Condorcet, weak Condorcet, Copeland evaluation, Minimax evaluation, Smith set, and Schulze evaluation). As we explain, our MPC components for tallying satisfy the requirements of the Ordinos framework and therefore yield provably secure e-voting systems, i.e., they inherit the accountability, privacy, and tally-hiding properties of the Ordinos framework.

Our implementations of the MPC components are available at [15]. We accurately assess the performance and scalability of our MPC components for practical applications. While our algorithms do not asymptotically improve over naturally expected baselines (e.g., IRV performs exponentially in the number of candidates), which was not the main goal of this work anyways, we are indeed able to show that the concrete performance is practical for real world elections (in the case of IRV and Schulze only for relatively small numbers of candidates).

Structure. In Section 2 we recall the Ordinos framework. We then, in Section 3, present and construct important building blocks used in subsequent sections. In Sections 4 to 7, we present our instantiations, implementations, and evaluations for the various voting methods we consider. We conclude in Section 8.

2 The Ordinos Framework

We need the following notation and terminology. We write $[n]$ to denote the set $\{0, \ldots, n-1\}$. Let n_c be the number of candidates/choices on a ballot and let n_v be the (maximal) number of voters. The format of a plain ballot is defined via a

finite *choice space* $\mathsf{C} \subseteq \mathbb{N}^{n_c}$, i.e., a ballot assigns each candidate/choice a number subject to constrains defined by C. For example, a single vote election where a plain ballot contains one vote for a single candidate/choice can be modeled via the choice space $\mathsf{C}_{\text{single}} := \{(b_0, \ldots, b_{n_c-1}) \in \{0,1\}^{n_c} \mid \sum_i b_i = 1\}$. For voter j we denote her plain ballot by $v^j := (v_i^j)_{i \in [n_c]} \in \mathsf{C}$. Ordinos uses an additively homomorphic t-out-of-n_t threshold[4] public key encryption scheme $\mathcal{E} = (E, D)$ with $E_{\mathsf{pk}}(a)$ denoting a ciphertext obtained as an encryption of plaintext a under the public key pk of the election.

Given this terminology, Ordinos [6] works roughly as follows. The protocol is run among a voting authority, the voters, n_t trustees, an authentication server, and an append-only bulletin board (BB). In the *setup phase*, parameters of the election are generated, including a public key and corresponding secret key shares for $\mathcal{E}$, one for each trustee, along with a NIZKP $\pi^{\mathsf{KeyShareGen}}$ from each trustee to prove knowledge of their key share. Additionally, C and the result function f_{res} of the election (see below) are fixed and published. In the *voting phase*, the voters first encrypt their ballots and then publish them on the BB, authenticating themselves as eligible voters with the help of the authentication server. An encrypted ballot of voter j has the form $(E_{\mathsf{pk}}(v_i^j))_{i \in [n_c]}$, i.e., each component of the plain ballot is encrypted separately. The encrypted ballot also contains a NIZKP π^{Enc} that proves validity of the plain ballot, i.e., $v^j = (v_i^j)_{i \in [n_c]} \in \mathsf{C}$. The published encrypted ballots can then be (publicly) homomorphically aggregated to obtain the encrypted and aggregated full tally, i.e., one obtains ciphertexts for $v_i := \sum_{j \in [n_v]} v_i^j$ where v_i is the total number of votes/points that candidate/choice i obtained in the election. In the *tallying phase*, the trustees run a publicly accountable MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ to compute f_{res}. This protocol takes as (secret) inputs the secret key shares of the trustees and the (public) encrypted aggregated tally and outputs $f_{\mathsf{res}}(v_0, \ldots, v_{n_c-1})$. This result, along with any material that is needed to verify the MPC computation, is published by the trustees on the BB. Finally, in the *verification phase*, voters can check that their ballots appear on the BB and everyone can verify the result by checking all NIZKPs as well as the (accountable) MPC computation.

Security of Ordinos (privacy and accountability) was shown independently of specific instantiations of the mentioned primitives, and hence, security is guaranteed by any instantiation fulfilling the necessary requirements. In what follows, we briefly recall the two generic security results of Ordinos (including the requirements for the underlying primitives), which have been formalized and proven in [6]. The first result states accountability of Ordinos, where accountability was formalized using the *KTV framework* [8].

Theorem 1 (Accountability [6], informal). *Let $\mathcal{E}$ be a correct additively homomorphic threshold public-key encryption scheme $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} be secure NIZKPs for $\mathcal{E}$, and $\mathsf{P}_{\mathsf{MPC}}$ be a publicly accountable MPC protocol, i.e., if the result does not correspond to the input, then this can be detected and at least one misbehaving trustee can be identified; this must hold true even if all*

[4] I.e., there are n_t secret key shares with $t \leq n_t$ secret shares being necessary for successful decryption.

trustees running the MPC protocol are malicious. Then (the resulting instance of) Ordinos is accountable.[5]

Importantly, Ordinos provides accountability (and hence, by results in [8] also verifiability) even if all trustees are malicious.

The following theorem (that was formalized and proven in [6]) states privacy of Ordinos, i.e., the tally-hiding property that no information besides the final result, according to the result function, is revealed to anyone, including the trustees. It was proven using the privacy definition given in [16].

Theorem 2 (Privacy/Tally Hiding [6], informal). *Let $\mathcal{E}$ be an additively homomorphic IND-CPA-secure t-out-of-n_t threshold public-key encryption scheme, $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} be secure NIZKPs for $\mathcal{E}$, and let $\mathsf{P_{MPC}}$ be an MPC protocol that securely realizes (in the sense of UC [17,18]) an ideal MPC functionality which essentially takes as input a vector of ciphertexts and returns f_{tally} evaluated on the corresponding plaintexts without leaking any other information if at most $t-1$ trustees are malicious. Then (the resulting instance of) Ordinos provides privacy/is tally-hiding in presence of up to $t-1$ malicious trustees.*

Instantiations of Ordinos. As mentioned in the introduction, for practical instantiations of Ordinos the main challenge lies in finding efficient and suitable instantiations of the primitives, including the MPC component, that work well and efficiently together. For certain kinds of elections and result functions this has been achieved by Küsters et al. in [6]. These instantiations use a threshold variant of the Paillier encryption scheme [19] to implement $\mathcal{E}$. To design their MPC protocols $\mathsf{P_{MPC}}$ for their result functions, Küsters et al. make use of and combine NIZKPs and publicly accountable MPC protocols from the literature that implement the following basic operations:

- $E_{\mathsf{pk}}(c) = f_{\mathsf{add}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a + b$, directly from the additive homomorphic property of Paillier encryption; for brevity we write $E_{\mathsf{pk}}(a) + E_{\mathsf{pk}}(b)$. Similarly, $E_{\mathsf{pk}}(c) = f_{\mathsf{mul}}(E_{\mathsf{pk}}(a), b)$ s.t. $c = a \cdot b$; for brevity we write $E_{\mathsf{pk}}(a) \cdot b$.
- $E_{\mathsf{pk}}(c) = f_{\mathsf{mul}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a \cdot b$, using a publicly accountable MPC protocol for multiplication [19]; for brevity we write $E_{\mathsf{pk}}(a) \cdot E_{\mathsf{pk}}(b)$.
- $E_{\mathsf{pk}}(c) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ iff $a \geq b$ and 0 otherwise, using a publicly accountable MPC protocol for the greater-than test [20].
- $E_{\mathsf{pk}}(c) = f_{\mathsf{eq}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ iff $a = b$ and 0 otherwise, using a publicly accountable MPC protocol for equality tests from [20].
- $c = f_{\mathsf{dec}}(E_{\mathsf{pk}}(a))$ s.t. $E_{\mathsf{pk}}(a)$ is an encryption of c, using publicly accountable distributed Paillier decryption [19].

The above components have been chosen not only because they meet the necessary security requirements but also due to their efficiency, which facilitates constructing practical instantiations. That is, f_{add} and multiplication with a

[5] We note that the security proof for accountability (and also for privacy) makes certain standard assumptions, such as honesty of the BB. We refer interested readers to [6] for full details. We also note that if $\mathsf{P_{MPC}}$ provides only public verifiability, instead of public accountability, then Ordinos provides verifiability.

publicly known value can be computed locally for the Paillier scheme. Furthermore, both f_{gt} and f_{eq} as proposed by [20] run in sublinear time independently of the actual plaintext space of the encryption scheme if plaintexts contained within the ciphertexts are upper bounded by some bound b_{ct}. Ordinos indeed has this property, where the bound generally depends on n_v and C. Furthermore, both f_{gt} and f_{eq} and Paillier synergize rather well. As discussed in [6], while f_{gt} and f_{eq} can in principle also be used with exponential ElGamal, both functions use decryption for a (upper-bounded but still) relatively large plaintext space, and hence, would perform poorly with exponential ElGamal.

We note that the above components have a useful property, namely, they can be combined to compute more complex functions such that the resulting protocol is still a secure publicly accountable MPC protocol. In other words, they allow for building protocols $\mathsf{P}_{\mathsf{MPC}}$ for Ordinos that meet the requirements of Theorems 1 and 2.

Our Instantiations and Parameters. In this work, we use Paillier encryption and the above basic building blocks. The main challenge and indeed a core contribution of our paper is to show and empirically demonstrate that these components are not just suitable for constructing protocols $\mathsf{P}_{\mathsf{MPC}}$ for simple result functions (e.g., revealing the candidate with the most votes in a single-vote election), but also for much more complex voting methods and result functions. To benchmark our implementations, we use the parameters as [6]. That is, we use a Paillier key of size 2048 bits and for the greater-than and equality protocols we use the range $[2^{16}]$, i.e., $b_{\mathsf{ct}} = 2^{16}$, for the (encrypted) plaintext inputs. This range can be increased if needed, i.e., to account for cases where aggregated ciphertexts might contain plaintexts outside of $[2^{16}]$. Note that, except for requiring a suitable upper bound b_{ct}, the performance of our MPC protocols is otherwise independent of the exact number of voters n_v due to aggregation of the ballots. The setup for our benchmarks consists of three trustees communicating over a local network. Each trustee ran on an ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM). As in [6], the benchmarks of our MPC protocols start with an already aggregated tally. Küsters et al. [6] showed for their MPC protocols that the number of trustees does not influence the benchmarks in a noticeable way and that, due to the sublinear communication complexity of the comparison protocols, there is no significant difference between a local network and the Internet. Both results also hold for our MPC constructions which are based on the same primitives. Hence, our benchmarks focus on the number of candidates which is the main factor for the performance of our protocols.

3 Building Blocks

In this section, we describe three MPC building blocks that can be obtained using the basic operations described in Section 2 and which we use to construct $\mathsf{P}_{\mathsf{MPC}}$ for our Ordinos instances, where the first building block is from [6].

Minimum k and Maximum k Values. Often, we have a vector $(E_{\mathsf{pk}}(a_i))_{i \in [n]}$ and want to compute ciphertexts $(E_{\mathsf{pk}}(b_i))_{i \in [n]}$ of a vector $(b_i)_{i \in [n]}$ such that

Floor Division

Input: $E_{\mathsf{pk}}(a), b, n$
Result: $E_{\mathsf{pk}}(i)$ with $i \in [n]$ such that $i \cdot b \leq a$ and $(i+1) \cdot b > a$
1 **for** $j \in [n+1]$ **do**
2 | $E_{\mathsf{pk}}(r_j) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(j \cdot b))$
3 **for** $j \in [n]$ **do**
4 | $E_{\mathsf{pk}}(\hat{r}_j) = E_{\mathsf{pk}}(r_j) - E_{\mathsf{pk}}(r_{j+1})$
5 $E_{\mathsf{pk}}(i) = \sum_{j \in [n]} E_{\mathsf{pk}}(j) \cdot E_{\mathsf{pk}}(\hat{r}_j)$
6 **return** $E_{\mathsf{pk}}(i)$

Fig. 1: Algorithm for Floor Division.

$b_i = 1$ if a_i is one of the k largest (resp. smallest) values in $(a_i)_{i \in [n]}$ and $b_i = 0$ otherwise. We do so as described in [6]. That is, we first construct the lower halve of the comparison matrix M such that $M_{i,j<i} := f_{\mathsf{gt}}(E_{\mathsf{pk}}(a_i), E_{\mathsf{pk}}(a_j))$. From this matrix, which consists of ciphertexts containing 0 or 1, one can compute a ciphertext for each a_i that contains the number of comparisons that i has won, i.e., where $a_i \geq a_j$ for some $j \neq i$. We can then use f_{gt} to compare this ciphertext (containing the results for a_i) with a ciphertext on the number $n - k - 1$ and obtain $E_{\mathsf{pk}}(b_i)$.[6] One can proceed similarly in order to find the smallest k values. Note that this algorithm can also be applied if k is not publicly known but rather only available as a ciphertext; in this situation, k is also not revealed by the algorithm. We make use of this property in the context of the Hare-Niemeyer method, see Section 4. We denote these algorithms for computing the vectors $E_{\mathsf{pk}}(b_i)$ by GetBest(), resp. GetWorst(). These algorithms have runtime $\mathcal{O}(n^2)$.

Maximum. If we are just interested in obtaining a ciphertext $E_{\mathsf{pk}}(a_i)$ of the maximum value a_i in the vector $(E_{\mathsf{pk}}(a_i))_{i \in [n]}$, we can do so more efficiently in linear runtime. That is, we start with the possible maximum $m = E_{\mathsf{pk}}(a_0)$ and iterate through all a_i's. For each a_i we test whether it is greater than the current maximum with $g = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a_i), m)$ and adapt the maximum accordingly with $m = g \cdot E_{\mathsf{pk}}(a_i) + (E_{\mathsf{pk}}(1) - g) \cdot m$. The minimum can be computed accordingly. We denote these algorithms by GetMax() resp. GetMin(). If we are interested in the indices of the values that are the maximum resp. minimum, we can first compute the encrypted maximum m and then compute for each index the encrypted indicator $E_{\mathsf{pk}}(b_i) := f_{\mathsf{eq}}(E_{\mathsf{pk}}(a_i), m)$, $b_i \in \{0, 1\}$. We denote these algorithms for obtaining the tuple of encrypted indicators with GetMaxIdx() and GetMinIdx().

Floor Division. Given a ciphertext $E_{\mathsf{pk}}(a)$ of some $a \in \mathbb{N}$ and a plain value $b \in \mathbb{N}_{>1}$, this algorithm, described in Figure 1, is used to compute a ciphertext $E_{\mathsf{pk}}(i)$ with $i = \lfloor \frac{a}{b} \rfloor$. The algorithm also requires a value $n \in \mathbb{N}$, s.t. $n \cdot b$ does not exceed the plaintext space size and $i \in [n]$. The algorithm compares all possible values. The sequence $(r_j)_{j \in [n+1]}$ consists of a sequence of zeros followed by a sequence of ones, where $r_j = 0$ if $a < jb$ and $r_j = 1$ otherwise. We are interested in the index i such that $r_i = 1$ and $r_{i+1} = 0$. We obtain this index by computing for each j the value $\hat{r}_j := r_j - r_{j+1}$. Then, we can use these $\hat{r}_j$ as indicators to obtain the correct division result.

[6] If there are multiple a_i with the same value, there might be more than k b_i that are 1. In cases where always exactly k such values are required, one can use a tie breaker mechanism such as the one described in [7].

4 Hare-Niemeyer Method

The Hare-Niemeyer method is an evaluation method for proportional allocation of seats that is used for example in Ukraine and Italy, but has also been used for German federal elections until 2005. The Hare-Niemeyer method is used for situations where a fixed number of seats needs to be assigned to candidates from different parties, where a voter typically votes only for the party and not the candidates themselves. Often, this type of proportional voting is also combined with some form of plurality or majority voting, such as first-pass-the-post-voting for electing single representatives for electoral districts, in so-called *mixed electoral systems*. Such mixed systems are also used for elections in many state parliaments in Germany, elections for the Scottish and Welsh parliaments and elections for the New Zealand House of Representatives. More specifically, the Hare-Niemeyer method for proportional voting works as follows: Assume that there are n_s seats to be assigned among n_c parties. Then, if there are a total of n_v valid votes and party c_i has received v_i votes, the number of seats that c_i is awarded is computed using the "ideal quota" given by $q_i := \frac{v_i \cdot n_s}{n_v}$. Initially, the number of seats awarded to each c_i is set to be $s'_i := \lfloor q_i \rfloor$. However, since these s'_i usually do not add up to n_s, the remaining $n_r \in [n_c]$ seats are distributed in the order of the highest remainders of $\frac{v_i \cdot n_s}{n_v}$. That is, the n_r parties c_i with the highest remainders $d_i = q_i - s'_i$ receive one additional seat each. Note that it could happen that multiple parties have the same remainders d_i, and thus, more than n_r additional seats are assigned. If this is not desired, then one would use a tie-breaking algorithm (cf. Section 5 and Footnote 6). There are many possible ways to vote in proportional elections. Our algorithm can handle every possible ballot format, as long as the ballots can be aggregated such that we obtain one ciphertext per party containing the total number of votes for the party. In the simplest case, one can use $\mathsf{C}_{\text{single}}$ as choice space with ballot format NIZKPs π^{Enc} from, for example, [21] and [19].

Our MPC algorithm for computing the Hare-Niemeyer method is presented in Figure 2. On a high-level, the algorithm follows the above description, i.e., it first computes the seat distribution without taking the remainder seats into account. Next, for each party, the remainder of the division (see above) is computed and the remainder seats are distributed among the parties with the highest remainder values. Importantly, this is achieved without revealing the total number of remainder seats or the set of parties that have received an additional seat.

We present benchmarks for our MPC tallying protocol in Figure 3. The runtime of the algorithm is linear in $n_c \cdot n_s$. As the figure shows, evaluating the Hare-Niemeyer method is highly efficient for a practical number of seats (1000) and (up to) 4 parties. Due to the linear growth, this should still be the case even if there are more parties than the maximum of 4 that we benchmarked. Also, recall from Section 2 that these benchmarks are essentially independent of the number of voters and trustees. In terms of security for our Ordinos instantiation, we obtain the following.

Theorem 3 (Security of Hare-Niemeyer method with Ordinos). *Let $\mathcal{E}$ be an additively homomorphic IND-CPA-secure t-out-of-n_t threshold public-key*

Tally-Hiding Hare-Niemeyer Evaluation

```
Input: Encrypted aggregated votes per party: {E_pk(v_i)}_{i∈[n_c]}
       Number of seats in total n_s and number of total votes n_v
Result: Vector s such that s_i is the number of seats of party i.
1  for i ∈ [n_c] do
2  |   m_i = E_pk(v_i) · n_s
3  |   E_pk(s'_i) = FloorDivision(m_i, n_v, n_s)
4  E_pk(n_r) = E_pk(n_s) − Σ_{i∈[n_c]} E_pk(s'_i)
5  for i ∈ [n_c] do
6  |   E_pk(d_i) = E_pk(v_i) · n_s − n_v · E_pk(s'_i).
7  (E_pk(d_i^best))_{i∈[n_c]} = GetBest((E_pk(d_0), ..., E_pk(d_{n_c−1})), E_pk(n_r))
8  for i ∈ [n_c] do
9  |   E_pk(s_i) = E_pk(s'_i) + E_pk(d_i^best)
10 |   s_i = f_dec(E_pk(s_i))
11 return s
```

Fig. 2: Tally-Hiding Hare-Niemeyer Evaluation

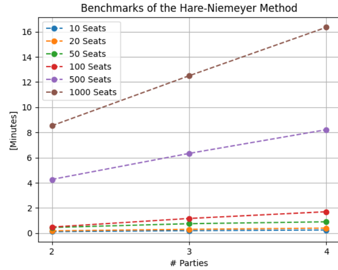
encryption scheme and $\pi^{\mathsf{KeyShareGen}}$ *be a secure NIZKP for* $\mathcal{E}$ *such as, e.g., the primitives used in [6]. Let* π^{Enc} *be the ballot format NIZKP from above, and let* $\mathsf{P}_{\mathsf{MPC}}$ *be our MPC component for the Hare-Niemeyer method as defined above. Then, the Ordinos instance using these primitives is an accountable and private (and hence tally-hiding) voting system for the Hare-Niemeyer method.*

Proof Sketch. This theorem is a direct corollary of Theorems 1 and 2 which were proven in [6]. Observe that the primitives $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$, and π^{Enc} already fulfill the requirements of Theorems 1 and 2. The only thing left to show for Theorems 1 and 2 is that our new tallying protocol $\mathsf{P}_{\mathsf{MPC}}$ is secure. That is, we have to show that $\mathsf{P}_{\mathsf{MPC}}$ is a private and publicly accountable implementation of the Hare-Niemeyer method.

Both properties follow because our MPC protocol is built from combinations of the basic components presented in Section 2. As mentioned in that section, each of these basic components guarantees privacy and public accountability. As for the connections of these components, the respective inputs and outputs are all encrypted (except for the final decryption of the election result) and published on the BB. Due to the encryption, these intermediate results do not leak any additional information, neither to internal parties nor to external observers. Also, since the intermediate results are published, external observers can check that the output of one step is used correctly as the input to the next step. Thus, if some trustee tries to use a different input, she can be held accountable. □

5 Instant Runoff Voting (IRV)

Instant-runoff-voting (IRV) is a ranked voting method which can be used in single-seat elections. It is often used, e.g., in Australia, India, the UK and the US. In IRV, if a candidate has been ranked first by an absolute majority of voters, this candidate is the winner of the election. Otherwise, the candidate ranked first least often is eliminated, i.e., removed from the pool of candidates. Then, all ballots are adjusted accordingly, i.e., the eliminated candidate is removed and other (lower-ranked) candidates are moved up a rank. This process

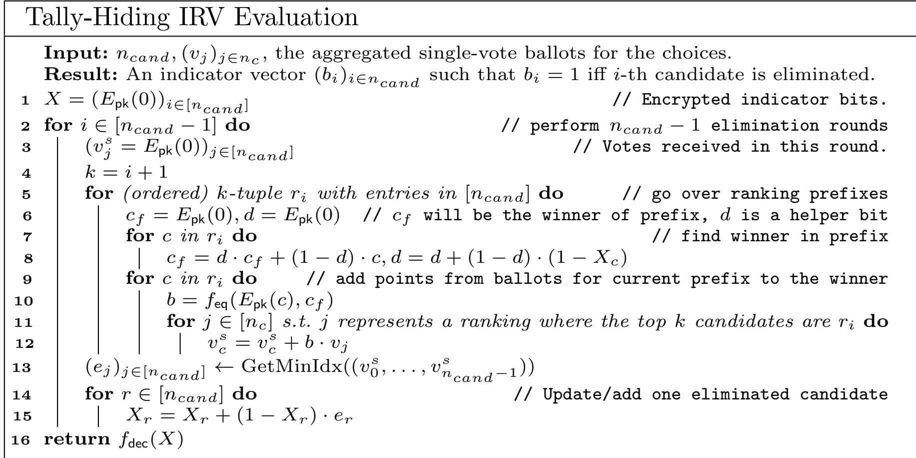


# Candidates	Runtime
3	6min 0s
4	18min 0s
5	327min 30s

Fig. 3: Benchmarks for the Hare-Niemeyer method (left) and IRV (right).

is repeated until one of the remaining candidates has received the absolute majority of votes and thus wins the election. An algorithm for evaluating IRV in a fully tally-hiding way has already been proposed in [3]. However, this algorithm does not support aggregation and therefore scales with the number of ballots/voters. Hence, instead of building on and providing the first implementations and benchmarks of this algorithm, we rather follow a different approach: we propose an algorithm that is compatible with the aggregation approach of Ordinos. By supporting aggregation, the performance of our solution remains essentially independent of the number of voters. For our instantiation, we use $\mathsf{C}_{\text{single}}$ but interpret each choice as a ranking of candidates. For example, for $n_{cand} = 5$, we have $n_c = n_{cand}! = 120$ choices, where each choice represents a permutation of the set of candidates. Observe that this encoding indeed allows for aggregating IRV ballots to obtain the full (encrypted) tally as usually done in Ordinos. NIZKPs π^{Enc} for showing the well-formedness of such a ballot are given in [21] and [19]. Note that the size of this choice space (and thus the runtime of our algorithm) scales exponentially in the number of candidates. However, we are able to show that this approach is still practical for a small amount of candidates (≤ 5) as they have occurred in practice (see benchmarks presented in Figure 3 and the discussion below).

We present our algorithm to evaluate an IRV election with Ordinos in Figure 4. The idea of our algorithm is that in round i, i.e. after i candidates have been eliminated, we have to consider the first $k = i + 1$ candidates of each ballot to find a candidate that has not been eliminated. We can then look at each possible ordering r_i of k candidates and check how many votes every permutation that starts with r_i received. These votes are then assigned to the respective first non-eliminated candidate in that permutation and the candidate with the least votes is eliminated. Note that it can happen that two candidates are assigned the same (lowest) number of votes in a round. Typically, IRV does not eliminate multiple candidates in the same round, hence in these situations some kind of tie-breaking algorithm is required. Often, this is done by lot - for example, this is the default method for IRV elections in Maine [22]. We address this issue, by letting GetMinIdx() output only the first candidate (i.e., the lower index) with the least amount of votes. To obtain randomized tie-breaking, one starts with



```
Tally-Hiding IRV Evaluation
  Input: n_cand, (v_j)_{j∈n_c}, the aggregated single-vote ballots for the choices.
  Result: An indicator vector (b_i)_{i∈n_cand} such that b_i = 1 iff i-th candidate is eliminated.
1  X = (E_pk(0))_{i∈[n_cand]}                                   // Encrypted indicator bits.
2  for i ∈ [n_cand − 1] do                            // perform n_cand − 1 elimination rounds
3  |  (v_j^s = E_pk(0))_{j∈[n_cand]}                            // Votes received in this round.
4  |  k = i + 1
5  |  for (ordered) k-tuple r_i with entries in [n_cand] do      // go over ranking prefixes
6  |  |  c_f = E_pk(0), d = E_pk(0)   // c_f will be the winner of prefix, d is a helper bit
7  |  |  for c in r_i do                                          // find winner in prefix
8  |  |  |  c_f = d · c_f + (1 − d) · c, d = d + (1 − d) · (1 − X_c)
9  |  |  for c in r_i do     // add points from ballots for current prefix to the winner
10 |  |  |  b = f_eq(E_pk(c), c_f)
11 |  |  |  for j ∈ [n_c] s.t. j represents a ranking where the top k candidates are r_i do
12 |  |  |  |  v_c^s = v_c^s + b · v_j
13 |  (e_j)_{j∈[n_cand]} ← GetMinIdx((v_0^s, ..., v_{n_cand−1}^s))
14 |  for r ∈ [n_cand] do                              // Update/add one eliminated candidate
15 |  |  X_r = X_r + (1 − X_r) · e_r
16 return f_dec(X)
```

Fig. 4: Tally-Hiding IRV Evaluation.

a uniformly randomly ordered list of candidates. It is interesting future work to explore implementations of more sophisticated tie-breaking algorithms.

We provide benchmarks for our IRV algorithm in Figure 3. Due to the encoding of IRV ballots as permutations of $[n_{\mathsf{cand}}]$, the algorithm has runtime $\mathcal{O}(n_{\mathsf{cand}}!)$. But as can be seen in Figure 3, for small numbers of candidates the evaluation is still feasible. Indeed, 5 candidates is already a realistic scenario for real world IRV elections. E.g., in the 2015 New South Wales state election [23], which, however, uses a different IRV instance than we consider here, most electoral districts had 5 or less candidates. Using the properties of our basic building blocks described in Section 2, one can check that our IRV algorithm does not leak information. By the same reasoning as for Theorem 3 we obtain:

Theorem 4 (Security of Instant-Runoff voting with Ordinos). *Let $\mathcal{E}$ and $\pi^{\mathsf{KeyShareGen}}$ be as for Theorem 3. Let π^{Enc} be the NIZKP from above, and let $\mathsf{P}_{\mathsf{MPC}}$ be our MPC component for the Instant-Runoff voting voting as defined in this section. Then, the Ordinos instance using these primitives is an accountable and private (and hence tally-hiding) voting system for Instant-Runoff voting.*

6 Condorcet methods

Condorcet is a ranked voting method that aims to determine a so-called *Condorcet winner*, i.e., a candidate that would beat all other candidates in a pairwise runoff election (we will call these pairwise runoff elections *comparisons*). It might happen that no candidate exists that wins all comparisons. There are several variants of (plain) Condorcet that deal with this, i.e., they output the Condorcet winner if it exists but additionally define mechanisms for obtaining a winner (or a set of winning candidates) also in some cases where no Condorcet winner exists. We discuss certain variants and their applications in practice below. We represent Condorcet ballots (which specify a full ranking of n_{cand} candidates

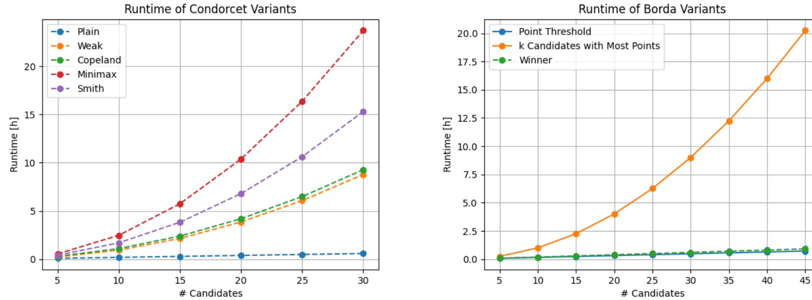


Fig. 5: Benchmarks for Condorcet voting (left) and benchmarks for Borda voting (right). The evaluation of the Schulze method for Condorcet took 135 minutes for 5 candidates and 9 days, 10 hours and 27 minutes for 20 candidates (not included in the figure).

without ties[7]) in Ordinos by interpreting them as a *comparison matrix*, i.e., an $(n_{cand} \times n_{cand})$-matrix M, where $M_{ij} \in \{0,1\}$ and $M_{ij} = 1$ means that a voter V prefers c_i over c_j. In order to obtain a choice space in the sense of definition of Section 2, we encode a comparison matrix as a vector of length $n_c = n_{cand}^2$ as expected way. Combined with some checks that ensure such a matrix indeed encodes a ranking (e.g., comparisons must be transitive), we obtain the choice space:

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

We can use the NIZKP π^{Enc} presented in [9] for showing the well-formedness of such ballots. As usual, Ordinos aggregates all the comparison matrices of all voters, yielding (encryptions of) a matrix containing at entry (i,j) the total number of comparisons that c_i wins versus c_j. This is then used as input for the various Condorcet variants that (try to) compute a winner in different ways.

We have implemented MPC tallying protocols for several such Condorcet variants, with details provided below. The benchmarks of these algorithms are presented in Figure 5. Apart from the Schulze method, the runtime of the MPC components of all Condorcet versions grow quadratically in n_{cand}, as expected due to the nature of pairwise comparisons, but remain practical for reasonable numbers of candidates. (We note that the verification of the NIZKPs given in [9] requires runtime that is asymptotically cubic in the number of candidates but is not included/shown in the benchmarks.) Plain Condorcet in particular exhibits runtime that suggests practicality even for very large numbers of candidates. Also, recall that our benchmarks are essentially independent of n_v and n_{t}. With the same reasoning as for Theorem 3 we obtain:

Theorem 5 (Security of Condorcet voting with Ordinos). *Let $\mathcal{E}$ and $\pi^{\mathsf{KeyShareGen}}$ be as for Theorem 3. Let π^{Enc} be the NIZKP from above, and let*

[7] Often, one allows for ties in Condorcet voting. However, in this work we do not consider this case.

$\mathsf{P_{MPC}}$ *be one of our MPC components for a Condorcet voting method as defined below. Then, the Ordinos instance using these primitives is an accountable and private (and hence tally-hiding) voting system for that Condorcet method.*
Next, we give details of the individual Condorcet variants and our corresponding MPC algorithms.

Plain Condorcet. We denote the vanilla Condorcet method, that outputs the unique Condorcet winner if and only if such a candidate exists, as *Plain Condorcet*. In Figure 6 (for bit $b = 1$), we present an algorithm for Plain Condorcet that is based on the building blocks described in Section 3. Note that, by choosing the bit $b = 0$ in Figure 6, the algorithm instead returns (encrypted) intermediate values, namely N, s^g and s'^g, which can be used for computing other Condorcet methods. Here, N denotes the *strict comparison matrix* that denotes in each entry $N_{i,j} \in \{0, 1\}$ whether c_i has won the majority of comparisons against c_j ($N_{i,j} = 1$) or won the same or less comparisons ($N_{i,j} = 0$). Additionally, for each candidate c_i, s^g_i denotes the number of comparisons that she has won or tied, while s'^g_i only counts the winning comparisons.

Weak Condorcet. In this method all candidates that did not lose any comparisons (but that might be tied with other candidates and thus no Condorcet winners), i.e. all *weak Condorcet winners* are output. This method can be obtained via a straightforward extension of Figure 6 for $b = 0$. That is, for each c_i, compute and check whether $f_{\mathsf{dec}}(f_{\mathsf{eq}}(s^g_i, E_{\mathsf{pk}}(n_{cand} - 1))) = 1$.

Copeland. This method, as opposed to the previous two methods, is guaranteed to output some winning candidate(s). To do so, it considers the wins and losses of each candidate in their comparisons and outputs all candidates with the most *Copeland points*, that is the highest difference between wins and losses. For $b = 0$, Figure 6 can be extended to first obtain the Copeland points of a candidate c_i via $E_{\mathsf{pk}}(p_i) := E_{\mathsf{pk}}(s'^g_i + s^g_i)$. We then compute the candidate with the most Copeland points with the GetMaxIdx() discussed in Section 3 and applying $f_{\mathsf{dec}}()$.

Schulze Method. This method is more complicated than the previous ones and is very commonly used in practice (e.g., [24]). This method defines the score of candidate c_i's comparison versus c_j to be the difference of the number of comparisons that c_i wins versus c_j minus the number of comparisons that c_j wins versus c_i. The candidates and the comparisons between them are considered as a directed weighted graph Γ, where the nodes of Γ represent the candidates and an arrow $c_i \to c_j$ is weighted with the score of c_i's comparison versus c_j. Now, for any path p in Γ, we define the *value* of p as the lowest weight among the arrows involved in p. We then consider the *path value matrix* $\mathsf{PathMatrix}$, an $(n_{cand} \times n_{cand})$-matrix with entry $\mathsf{PathMatrix}_{ij}$ being the highest path value among paths from c_i to c_j. The Schulze method then outputs all candidates c_i such that $\mathsf{PathMatrix}_{ij} \geq \mathsf{PathMatrix}_{ji}$ for each $j \in [n_{cand}]$. Note that the Schulze method is guaranteed to output some candidate(s). And if a unique Condorcet winner exists, then it will be returned by the Schulze method. The intuitive and probably most natural way to implement the Schulze method is to simply compute the standard algorithm while using MPC building blocks to implement all operations, which, for example, is also done in [7]. The main challenge lies

Condorcet Evaluation

Input: Encrypted aggregated comparison matrix: $A := E_{\mathsf{pk}}(M)$
$b \in \{0,1\}$: indicator whether plain Condorcet should be evaluated.

1 $N = 0_{n_{cand} \times n_{cand}}, s^g = 0_{n_{cand}}, s'^g = 0_{n_{cand}}$
2 **for** $i \in [n_{cand}]$ **do**
3   **for** $j \in [i+1, n_{cand}]$ **do**
4         $g = f_{\mathsf{gt}}(A_{i,j}, A_{j,i}), e = f_{\mathsf{eq}}(A_{i,j}, A_{j,i}), g' = g - e$
5         $N_{i,j} = g', N_{j,i} = E_{\mathsf{pk}}(1) - g$
6         $s^g_i = s^g_i + g, s^g_j = s^g_j + E_{\mathsf{pk}}(1) - g', s'^g_i = s'^g_i + g', s'^g_j = s'^g_j + E_{\mathsf{pk}}(1) - g$
7   **if** $b = 1$ **then**
8   **if** $f_{\mathsf{dec}}(f_{\mathsf{eq}}(s'^g_i, E_{\mathsf{pk}}(n_{cand}) - 1))$ **then**
9   **return** i
10 **return** N, s^g, s'^g

Fig. 6: Condorcet Evaluation.

Condorcet: Schulze Evaluation

Input: Encrypted aggregated comparison matrix: M
Result: Vector $(b_i)_{i \in [n_{cand}]}$ such that $b_i = 1$ if c_i is a Schulze winner and $b_i = 0$ otherwise.

1 $\mathsf{PathMatrix} = (E_{\mathsf{pk}}(0))_{n_{cand} \times n_{cand}}$
2 **for** $i \in [n_{cand}], j \in [n_{cand}] \setminus \{i\}$ **do**
3     $\mathsf{PathMatrix}_{i,j} = M_{i,j} - M_{j,i}$
4 **for** $i \in [n_{cand}], j \in [n_{cand}] \setminus \{i\}, k \in [n_{cand}] \setminus \{i,j\}$ **do**
5     $m = \mathrm{GetMin}(\mathsf{PathMatrix}_{j,i}, \mathsf{PathMatrix}_{i,k})$
6     $\mathsf{PathMatrix}_{j,k} = \mathrm{GetMax}(M_{j,k}, m)$
7 $M^{\mathsf{Schulze}} = (E_{\mathsf{pk}}(0))_{[n_{cand}] \times [n_{cand}]}$
8 **for** $i \in [n_{cand}], j \in [i]$ **do**
9     $g = f_{\mathsf{gt}}(\mathsf{PathMatrix}_{i,j}, \mathsf{PathMatrix}_{j,i})$
10     $e = f_{\mathsf{eq}}(\mathsf{PathMatrix}_{i,j}, \mathsf{PathMatrix}_{j,i})$
11     $M^{\mathsf{Schulze}}_{i,j} = g, M^{\mathsf{Schulze}}_{j,i} = E_{\mathsf{pk}}(1) - g + e$
12 $b = (E_{\mathsf{pk}}(0))_{n_{cand}}$
13 **for** $i \in [n_{cand}]$ **do**
14     $w = \sum_{j \in [n_{cand}] \setminus \{i\}} M^{\mathsf{Schulze}}_{i,j}$
15     $b_i = f_{\mathsf{dec}}(f_{\mathsf{eq}}(w, E_{\mathsf{pk}}(n_{cand} - 1)))$
16 **return** $(b_i)_{i \in [n_{cand}]}$

Fig. 7: Condorcet: Schulze Evaluation.

in choosing suitable MPC building blocks such that the resulting tally-hiding Schulze algorithm performs well. Here we use the sublinear comparison protocols from Section 2, with the resulting algorithm presented in Figure 7.

Further Condorcet methods: We have also implemented and benchmarked the so-called *Smith set* and *Minmax* Condorcet methods. Intuitively, the smith set outputs a set of candidates such that each candidate from this set wins the comparisons against every candidate outside of the set. Minmax intuitively considers the "worst" comparison of each candidate and then output all candidates that have the "best" of these worst comparisons. Our algorithms for these Condorcet methods are constructed using the same techniques and building blocks as for the previous methods. Due to space constraints, we do not present our algorithms in detail her but rather refer the reader to our implementation [15].

7 Borda

Borda count is a ranked voting method where each assignable rank is associated with a pre-defined number of points that the corresponding candidate receives. The winner typically is the candidate who has received the most points in total (summed over all ballots). A famous application of Borda count is the election

of the winner of the grand final in the Eurovision Song Contest, but it is also used for national elections, for example in the Republic of Nauru.

The following choice space can be used to capture Borda, where we interpret $\mathcal{P}$ both as a list and a set: $\mathsf{C}_{\mathsf{Borda}}(\mathcal{P}) = \{(x_1, \ldots, x_{n_c}) \mid \forall i : x_i \in \mathcal{P} \ \wedge \forall i \in \mathcal{P} \exists j : x_j = i\}$. A NIZKP π^{Enc} for the well-formedness of ballots for this choice space is presented in [21]. By definition of Ordinos, the encrypted aggregated tally $(E_{\mathsf{pk}}(p_i))_{i \in [n_c]}$ then consists of encryptions of the sum of points p_i that candidate c_i received. In principle, one can now use the same MPC tallying protocols presented in [6] for single-/multi-vote to *(i)* output the candidate with the highest points, *(ii)* output the k candidates with the most points, or *(iii)* output all candidates that cleared a certain threshold of points. However, for the standard case *(i)* we propose a more efficient way that is not quadratic but linear in the number of candidates: We use the algorithm GetMaxIdx() (cf. Section 3) and then apply f_{dec}; the winner is the candidate for whom decryption yields 1.[8]

The benchmarks of these algorithms are presented in Figure 5, where the result functions *(ii)* and *(iii)* are implemented using the algorithms by [6]. As the benchmarks show, our algorithm for *(i)* and the algorithm for *(iii)* can be computed highly efficiently. Due to the linear growth, this should still be the case even if there are much more candidates than the maximum of 40 that we benchmarked. Result function *(ii)* shows, as expected, a quadratic growth in the number of candidates. However, the runtime for ≤ 40 candidates remains in a range that is often still reasonable for practical elections. Also, recall that our benchmarks are essentially independent from n_v and n_{t}. With the same reasoning as for Theorem 3 we obtain:

Theorem 6 (Security of Borda voting with Ordinos). *Let $\mathcal{E}$ and $\pi^{\mathsf{KeyShareGen}}$ be as for Theorem 3. Let π^{Enc} be the NIZKP from above, and let $\mathsf{P}_{\mathsf{MPC}}$ be one of our MPC components for (one of the result functions for) Borda voting as defined in this section. Then, the Ordinos instance using these primitives is an accountable and private (and hence tally-hiding) voting system for Borda (using that result function).*

8 Conclusion

We have proposed, implemented, and benchmarked several new accountable tally-hiding MPC components for Ordinos. These are the first tally-hiding implementations for the Hare-Niemeyer method, IRV, multiple variants of Condorcet, and Borda. The performance of our MPC components is determined by the number of candidates while being essentially independent of the number of trustees and the number of voters, as long as the aggregated ballots still meet the bound b_{ct}. Analogously to [6], due to the comparison protocols with sublinear communication cost, our runtimes are almost independent of the network (local vs. Internet). Our instantiations achieve reasonable runtimes that allow

[8] If always a single winner should be determined, one can use a tie-breaking algorithm after GetMaxIdx(), similarly to what we describe in Section 5 for GetMinIdx(). Note that this adds only a small linear overhead.

for deployment in real-world applications. In future work, it would be interesting to investigate optimizations for our algorithms and to implement further voting methods.

References

1. J. D. Cohen, *Improving Privacy in Cryptographic Elections*. Citeseer, 1986.
2. A. Hevia and M. A. Kiwi, "Electronic jury voting protocols," *TCS*, 2004.
3. R. Wen and R. Buckland, "Minimum Disclosure Counting for the Alternative Vote," in *VoteID, Luxembourg.*, 2009.
4. A. Szepieniec and B. Preneel, "New Techniques for Electronic Voting," ePrint Report 2015/809.
5. S. Canard, D. Pointcheval, Q. Santos, and J. Traoré, "Practical Strategy-Resistant Privacy-Preserving Elections," in *ESORICS 2018*, vol. 11099. Springer, 2018.
6. R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, "Ordinos: A Verifiable Tally-Hiding E-Voting System," in *EuroS&P*. IEEE, 2020, pp. 216–235.
7. V. Cortier, P. Gaudry, and Q. Yang, "A toolbox for verifiable tally-hiding e-voting systems," ePrint Report 2021/491.
8. R. Küsters, T. Truderung, and A. Vogt, "Accountability: Definition and Relationship to Verifiability," in *CCS*, 2010.
9. T. Haines, D. Pattinson, and M. Tiwari, "Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme," in *VSTTE 2019*, 2019.
10. K. Ramchen, C. Culnane, O. Pereira, and V. Teague, "Universally Verifiable MPC and IRV Ballot Counting," in *FC 2019*, ser. LNCS. Springer, 2019.
11. W. Jamroga, P. B. Rønne, P. Y. A. Ryan, and P. B. Stark, "Risk-Limiting Tallies," in *E-Vote-ID 2019*, 2019.
12. A. Juels, D. Catalano, and M. Jakobsson, "Coercion-Resistant Electronic Elections," ePrint Report 2002/165.
13. J. Heather, "Implementing STV securely in Prêt à Voter," in *CSF*, 2007.
14. J. Benaloh, T. Moran, L. Naish, K. Ramchen, and V. Teague, "Shuffle-sum: coercion-resistant verifiable tallying for STV voting," *TIFS*, 2009.
15. F. Hertel, N. Huber, J. Kittelberger, R. Küsters, J. Liedtke, and D. Rausch, "Ordinos Code Repository," https://github.com/JulianLiedtke/ordinos.
16. R. Küsters, T. Truderung, and A. Vogt, "Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study," in *S&P 2011*, 2011.
17. R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *FOCS 2001*. IEEE Computer Society, 2001.
18. R. Küsters, "Simulation-Based Security with Inexhaustible Interactive Turing Machines," in *CSFW-19*, 2006, see [25] for a full and revised version.
19. I. Damgård, M. Jurik, and J. B. Nielsen, "A Generalization of Paillier's Public-Key System with Applications to Electronic Voting," *Int. J. Inf. Sec.*, 2010.
20. H. Lipmaa and T. Toft, "Secure Equality and Greater-Than Tests with Sublinear Online Complexity," in *ICALP 2013*, vol. 7966. Springer, 2013, pp. 645–656.
21. J. Groth, "Non-interactive Zero-Knowledge Arguments for Voting," in *ACNS 2005*.
22. Maine State Legislature, "Ranked Choice Voting in Maine," http://legislature.maine.gov/lawlibrary/ranked-choice-voting-in-maine/9509, 2020.
23. Electoral Commission NSW, "NSW State Election Results 2015," https://pastvtr.elections.nsw.gov.au/SGE2015/la-home.htm, 2021.
24. M. Schulze, "The Schulze Method of Voting," *CoRR*, 2018.
25. R. Küsters, M. Tuengerthal, and D. Rausch, "The IITM Model: A Simple and Expressive Model for Universal Composability," *Journal of Cryptology*, 2020.