Ordinos: A Verifiable Tally-Hiding E-Voting System

Ralf Küsters* Julian Liedtke* Johannes Müller† Daniel Rausch* Andreas Vogt‡

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

† *SnT/University of Luxembourg, Luxembourg*
johannes.mueller@uni.lu

‡ *FHNW, Switzerland*
andreas.vogt@fhnw.ch

***Abstract*—Modern electronic voting systems (e-voting systems) are designed to provide not only vote privacy but also (end-to-end) verifiability. Several verifiable e-voting systems have been proposed in the literature, with Helios being one of the most prominent ones.**

Almost all such systems, however, reveal not just the voting result but also the full tally, consisting of the exact number of votes per candidate or even all single votes. There are several situations where this is undesirable. For example, in elections with only a few voters (e.g., boardroom or jury votings), revealing the complete tally leads to a low privacy level, possibly deterring voters from voting for their actual preference. In other cases, revealing the complete tally might unnecessarily embarrass some candidates. Often, the voting result merely consists of a single winner or a ranking of candidates, so revealing only this information but not the complete tally is sufficient. This property is called tally-hiding and it offers completely new options for e-voting.

In this paper, we propose the first provably secure end-to-end verifiable tally-hiding e-voting system, called Ordinos. We instantiated our system with suitable cryptographic primitives, including an MPC protocol for greater-than tests, implemented the system, and evaluated its performance, demonstrating its practicality. Moreover, our work provides a deeper understanding of tally-hiding in general, in particular in how far tally-hiding affects the levels of privacy and verifiability of e-voting systems.

1. Introduction

E-voting systems are widely used both for national, state-wide, and municipal elections all over the world with several hundred million voters so far. Beyond such high-stake elections, e-voting is also becoming very common for elections within companies, associations, political parties, non-profit organizations, and religious institutions (see, e.g., [1]–[5]). In order to meet this increasing demand for e-voting solutions, many IT companies offer their services [6], including, for example, Microsoft [7].

There are roughly two types of e-voting systems: those where the voter has to go to a polling station in order to cast her vote using a voting machine and those that allow the voter to cast her vote remotely over the Internet (remote e-voting), using her own device. The latter type of e-voting is the main focus of this paper.

Since e-voting systems are complex software and hardware systems, programming errors are unavoidable and deliberate manipulation of such systems is often hard or virtually impossible to detect. Hence, there is a high risk that votes are not counted correctly and that the published election result does not reflect how voters actually voted (see, e.g., [8]–[10]).

Therefore, there has been intensive and ongoing research to design e-voting systems which provide what is called *(end-to-end) verifiability* (see, e.g., [11]–[20]), where voters and external observers are able to check whether votes were actually counted and whether the published election result is correct, even if voting devices and servers have programming errors or are outright malicious. Several of such systems have already been deployed in real binding elections (see, e.g., [15], [16], [21]–[24]), with Helios [15] being one of the most prominent (remote) e-voting systems. In Switzerland and Norway, for example, e-voting systems for national and local elections and referendums are even required to provide verifiability [25], [26].

Tally-hiding e-voting. In the real world, there are numerous voting methods that differ in the form of the final result and the rules to determine it. For example, one of the most widely used ones is plurality voting in which the candidate with the highest number of votes is elected. For some elections, a more refined version of plurality voting with two rounds is used: if a candidate receives the absolute majority of votes in the first round, she wins; otherwise, there is a runoff between the two best candidates of the first round. In both cases, the final result of the voting method merely consists of the winner. Another popular voting method is to allocate k seats of a board or commission to the k candidates with the highest number of votes. In this case, the final result of the voting method merely consists of the set of elected candidates.

Given this variety of voting methods, how do existing (electronic or paper-based) voting protocols realize a given voting method? Essentially all of them first reveal the *full tally*, i.e., the number of votes per candidate/party or even all single votes, as an intermediate step that is then used to compute the *actual election result*, e.g., the winner. For traditional, fully paper-based voting, it seems practically infeasible to follow a different approach without having to sacrifice verifiability.

As demonstrated in this paper, electronic voting can, fortunately, offer completely new options for elections in that beyond the necessary final result no further information is revealed. Strictly realizing the given voting method without revealing any unnecessary information, such as intermediate results, provides several advantages,

for example, in the following situations:

(i) As mentioned above, some elections have several rounds. In particular, they might involve runoff elections. In order to get unbiased voters' opinions, one might not want to reveal intermediate election tallys, except for the information which candidates move on to the runoff elections. For example, if no candidate received the absolute majority, one might want to reveal only the two best candidates (without their vote counts and without revealing their ranking), who then move on to the runoff election.

(ii) Elections are often carried out among a small set of voters (e.g., in boardroom or jury votings). Even in an ideal voting system, revealing the full tally in such a setting leads to a low level of privacy because a single voter's vote is "hidden" behind only a low number of other votes. Therefore, even if, say, it is only interesting who won the election, a voter might not vote for her actual preference knowing that nevertheless the full tally is revealed, and hence, her vote does not really remain private.

(iii) In some elections, for example, within companies, student associations, or in boardroom elections, it might be unnecessarily embarrassing for the losing candidates to publish the (possibly low) number of votes they received. For example, the German Informatics Society as well as the German Research Foundation do not always publish the full tally for this reason.

These examples illustrate that, for some situations, it is desirable to not publish the full tally as part of the tallying procedure but to only publish the pure election result, e.g., only the winner of an election with or without the number of votes he/she has received, only the set of the first k candidates, who win seats, only the set of the last k candidates, which might be excluded from a runoff election, or only a ranking of candidates, without the number of votes each of them has received.

Following [27], we call e-voting systems that compute an election result without revealing any additional information, such as the full tally, *tally-hiding*. So, while tally-hiding e-voting is desirable in many situations, as to the best of our knowledge, only four tally-hiding e-voting systems have been proposed in the literature to date: a quite old one by Benaloh [28], one by Hevia and Kiwi [29], and two more recent ones by Szepieniec and Preneel [27] and by Canard et al. [30]. As further discussed in Section 9, among other shortcomings, none of these systems come with a rigorous cryptographic security proof and only one of these systems has been implemented. As also discussed in Section 9, current generic MPC protocols also do not seem to directly provide suitable solutions for practical tally-hiding and verifiable e-voting.

Hence, it remains an open problem to develop and implement a provably secure end-to-end verifiable tally-hiding e-voting system for (remote) elections. Solving this open problem is the main goal of this paper, where we also want our system to be of practical use. Furthermore, we provide a deeper understanding of tally-hiding in general, in particular how tally-hiding affects verifiability and privacy properties.

By making e-voting systems not only tally-hiding but also verifiable, the feature of tally-hiding will become more attractive in the future: as mentioned above, in classical paper-based elections hiding the full tally, even if only part of it is really needed, seems infeasible without losing trust in the final result. In contrast, as demonstrated in this paper, it is possible to build provably secure verifiable tally-hiding e-voting systems. Such systems allow for detecting manipulation and therefore establish trust in the final result (e.g., the winner or the ranking of candidates) even without publishing the full tally. Altogether, this opens up completely new kinds of elections that were not possible before.

Our Contribution. We present Ordinos, the first provably secure tally-hiding and verifiable (remote) e-voting system.

Conceptually, Ordinos follows the general structure of the Helios remote e-voting system, at least in its first phase, but strictly extends Helios' functionality: Helios always reveals the full tally, no matter what the actual desired election result is. In contrast, Ordinos (see Section 2) supports several election result functions[1] in a tally-hiding fashion. That is, as explained above, beyond the result of the election according to the election result function, Ordinos does not reveal any further information. In particular, Ordinos, unlike Helios, does not reveal the full tally (e.g., the number of votes per candidate) if not required by the result function. Among others, Ordinos supports tally-hiding elections for the following result functions: revealing only the winner of an election, the k best/worst candidates (with or without ranking), or the overall ranking, optionally with or without disclosing the number of votes for some or all candidates.

Compared to Helios, Ordinos uses different (instantiations of) cryptographic primitives and also additional primitives, in particular, a suitable MPC component, in order to obtain a tally-hiding system.

We carry out a detailed cryptographic analysis proving that Ordinos provides privacy (Section 5) and verifiability (Section 4): We show that Ordinos preserves the same level of verifiability as Helios under the same trust assumptions (namely that the verification devices and the bulletin board are not corrupted), independently of the result function considered. More generally, this demonstrates that the common definition of verifiability can be achieved independently of whether a result function is computed in a tally-hiding way. Conversely, the level of privacy Ordinos provides can be much better than for Helios, e.g., if only the winner or the ranking of candidates is to be published.

To study privacy for tally-hiding voting more generally, we derive privacy results for an ideal tally-hiding voting protocol for various result functions in order to compare the privacy levels (Section 6). We then show that the privacy of Ordinos can be reduced to the ideal protocol (Section 5). These general results about properties of tally-hiding voting systems are of independent interest.

We note that Ordinos even provides *accountability*, a security property that is stronger than just verifiability. That is, roughly speaking, Ordinos guarantees that if the published election result does not match how voters

1. Given individual votes of voters or the number of votes per candidate/choice, an *(election) result function* returns the final result of the election, e.g., the winner of the election or the k best candidates (with or without a ranking of these candidates).

actually voted, then this is not only detected (as required for verifiability), but misbehaving parties can even be identified and blamed for their misbehavior. Due to space limitations, our presentation in this paper concentrates on verifiability. However, as discussed in Section 4.2, we actually prove accountability, from which verifiability follows immediately.

Our cryptographic analysis of Ordinos (for privacy and verifiability/ accountability) is based on generic properties of the employed cryptographic primitives, and hence, is quite general and does not rely on specific instantiations. In order to obtain a workable system, we carefully crafted one instantiation (Section 7) using, among others, Paillier encryption [31], an MPC protocol for greater-than by Lipmaa and Toft [32], as well as NIZKPs by Schoenmakers and Veeningen [33]. We provide a proof-of-concept implementation of Ordinos based on this instantiation and evaluate its performance, demonstrating its practicability (Section 8).

Further details are provided in the appendix and all formal proofs can be found in our technical report [34]. The implementation and detailed benchmarks are available upon request.

2. Description of Ordinos

In this section, we present the Ordinos voting protocol on the conceptual level. We start with the cryptographic primitives that we use. Instead of relying on specific primitives, the security of Ordinos can be guaranteed under certain assumptions these primitives have to satisfy. In particular, they can later be instantiated with the most appropriate primitives available.

As already mentioned, Ordinos extends the prominent Helios e-voting protocol. While in Helios the complete election result is published (the number of votes per candidate/choice), Ordinos supports tally-hiding elections. More specifically, the generic version of Ordinos, which we prove secure, supports arbitrary result functions evaluated over the aggregated votes per candidate and computes these result functions in a tally-hiding way. Our concrete instantiation (see Section 7) then realizes many such practically relevant functions.

In a nutshell, Ordinos works as follows: Voters encrypt their votes and send their ciphertexts to a bulletin board. The ciphertexts are homomorphically aggregated to obtain ciphertexts that encrypt the number of votes per candidate. Then, by an MPC protocol, trustees evaluate the desired result function on these ciphertexts and publish the election result.

Cryptographic primitives. For Ordinos, we use the following cryptographic primitives: A homomorphic, IND-CPA-secure $(t, n_{\mathsf{trustees}})$-*threshold public-key encryption scheme* $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$, e.g., exponential ElGamal or Paillier. A *non-interactive zero-knowledge proof* (NIZKP) π^{Enc} for proving knowledge and correctness of a plaintext vector given a ciphertext vector, a public key, and a predicate which specifies valid choices (see below); a NIZKP $\pi^{\mathsf{KeyShareGen}}$ for proving knowledge and correctness of a private key share given a public key share (see Appendix C for details). A *multi-party computation (MPC) protocol* $\mathsf{P}_{\mathsf{MPC}}$ that takes as input a ciphertext vector of encrypted integers (encrypted using $\mathcal{E}$ from above) and securely evaluates a given function f_{tally} over the plain integers and outputs the result on a bulletin board. For example, a function f_{tally} that outputs the index(s) of the ciphertext(s) with the highest integer would be used to determine and publish the winner of an election. The exact security properties $\mathsf{P}_{\mathsf{MPC}}$ has to satisfy to achieve privacy and verifiability for the overall system, Ordinos, are made precise in the following sections. Finally, we use an EUF-CMA-secure *signature scheme* $\mathcal{S}$.

Protocol participants. The Ordinos protocol is run among the following participants: a voting authority Auth, (human) voters $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$, voter supporting devices $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, voter verification devices $\mathsf{VVD}_1, \ldots, \mathsf{VVD}_{n_{\mathsf{voters}}}$, trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$, an authentication server AS, and an append-only bulletin board B.

As further described below, the role of each (untrusted) voter supporting device VSD is to generate and submit the voter's ballot, whereas the (trusted) voter verification device VVD checks that the VSD behaved correctly. The role of the trustees is to tally the voters' ballots. In order to avoid that a single trustee knows how each single voter voted, the secret tallying key is distributed among all of them so that t out of n_{trustees} trustees need to collaborate to tally the ballots.

We assume the existence of the following authenticated channels:[2] First, all parties have unilaterally authenticated channels to the bulletin board B, ensuring that all parties have the same view on the bulletin board. Second, for all VSD_i, an authenticated channel between this device and the authentication server AS, allowing AS to ensure that only eligible voters are able to cast their ballots. Third, authenticated channels between each voter V_i and her VSD_i as well as her VVD_i, modeling direct interaction.

Protocol overview. A protocol run consists of the following phases: In the *setup phase*, parameters and key shares are fixed/generated, and the public key shares are published. In the *voting phase*, voters pick their choices, encrypt them, and then either audit or submit their ballots. Now or later, in the *voter verification phase*, voters verify that their ballots have been published by the authentication server. In the *tallying phase*, the trustees homomorphically aggregate the ballots, run $\mathsf{P}_{\mathsf{MPC}}$ in order to secretly compute, and publish the election result according to the result function f_{tally} so that not even the trustees learn anything beyond the final result (which guarantees tally-hiding). Finally, in the *public verification phase*, everyone can verify that the trustees tallied correctly.

We now explain each phase in more detail.

Setup phase. In this phase, all election parameters are fixed and posted on the bulletin board by the voting authority Auth: the list $\vec{\mathsf{id}}$ of eligible voters, opening and closing times, the election ID $\mathsf{id}_{\mathsf{election}}$, etc. as well as the set $\mathsf{C} \subseteq \{0, \ldots, n_{\mathsf{vpc}}\}^{n_{\mathsf{option}}}$ of valid choices where n_{option} denotes the number of options/candidates, n_{vpc} the number of allowed votes per option/candidate, and $\mathsf{abstain}$ models that a voter abstains from voting. For example,

2. By assuming such authenticated channels, we abstract away from the exact method the voters use to authenticate; in practice, several methods can be used, such as one-time codes, passwords, or external authentication services.

if each voter can vote for at most one candidate, then $n_{\mathsf{vpc}} = 1$ and every vector in C contains at most one 1-entry.

The authentication server AS and each trustee T_k run the key generation algorithm of the digital signature scheme $\mathcal{S}$ to generate their public/private (verification/signing) keys. The verification keys are published on the bulletin board B. In what follows, we implicitly assume that whenever the authentication server AS or a trustee T_k publish information, they sign this data with their signing keys.

Every trustee T_k runs the key share generation algorithm of the public-key encryption scheme $\mathcal{E}$ to generate its public/private (encryption/decryption) key share pair $(\mathsf{pk}_k, \mathsf{sk}_k)$. Additionally, each trustee T_k creates a NIZKP $\pi_k^{\mathsf{KeyShareGen}}$ to prove knowledge of sk_k and validity of $(\mathsf{pk}_k, \mathsf{sk}_k)$. Each trustee T_k then posts $(\mathsf{pk}_k, \pi_k^{\mathsf{KeyShareGen}})$ on the bulletin board B. With $\mathsf{PublicKeyGen}$, everyone can then compute the (overall) public key pk.

Voting phase. In this phase, every voter V_i can decide to abstain from voting or to vote for some choice $\mathsf{ch} \in \mathsf{C} \subseteq \{0, \ldots, n_{\mathsf{vpc}}\}^{n_{\mathsf{option}}}$. In the latter case, the voter inputs ch to her voter supporting device VSD_i which proceeds as follows. First, VSD_i encrypts each entry of ch separately under the public key pk and obtains a ciphertext vector $\vec{\mathsf{c}}_i$. That is, the j-th ciphertext in $\vec{\mathsf{c}}_i$ encrypts the number of votes assigned by voter V_i to candidate/option j. After that, in addition to $\vec{\mathsf{c}}_i$, VSD_i creates a NIZKP π_i^{Enc} in order to prove that it knows which choice ch the vector $\vec{\mathsf{c}}_i$ encrypts and that $\mathsf{ch} \in \mathsf{C}$. Finally, VSD_i sends a message to V_i to indicate that a ballot $\mathsf{b}_i = (\mathsf{id}, \vec{\mathsf{c}}_i, \pi_i^{\mathsf{Enc}})$ is ready for submission, where $\mathsf{id} \in \vec{\mathsf{id}}$ is the voter's identifier. Upon receiving this message, the voter V_i can decide to either audit or submit the ballot b_i (*Benaloh challenge* [35]), as described in what follows.[3]

If V_i wants to audit b_i, V_i inputs an audit command to VSD_i who is supposed to reveal all the random coins that it had used to encrypt V_i's choice and to generate the NIZKPs. After that, V_i forwards this data and her choice ch to her verification device VVD_i which is supposed to check the correctness of the ballot, i.e., whether ch chosen by V_i and the revealed randomness by VSD_i yield b_i. Audited ballots cannot be cast. The voter is therefore asked to vote again.

If V_i wants to submit b_i, V_i inputs a cast command to VSD_i, which is supposed to send b_i to the authentication server AS on an authenticated channel. If AS receives a ballot in the correct format (i.e., $\mathsf{id} \in \vec{\mathsf{id}}$ and id belongs to V_i, and b_i is tagged with the correct election ID $\mathsf{id}_{\mathsf{election}}$) and the NIZKP π_i^{Enc} is valid, then AS responds with an acknowledgement consisting of a signature on the ballot b_i; otherwise, it does not output anything.[4] After that, VSD_i forwards the ballot b_i as well as the acknowledgement to VVD_i for verification purposes later on. If the voter tried to re-vote and AS already sent out an acknowledgement, then AS returns the old acknowledgement only and does not accept the new vote. If VVD_i does not receive a correct acknowledgement from AS via VSD_i, it outputs a message to V_i who then tries to re-vote, and, if this does not succeed, files an authentication complaint on the bulletin board.[5]

When the voting phase is over, AS creates the list of valid ballots $\vec{\mathsf{b}}$ that have been submitted. Then AS removes all ballots from $\vec{\mathsf{b}}$ that are duplicates w.r.t. the pair $(\vec{\mathsf{c}}, \pi^{\mathsf{Enc}})$ only keeping the first one in order to protect against *replay attacks*, which jeopardize vote privacy [37]. Afterwards, AS signs $\vec{\mathsf{b}}$ and publishes it on the bulletin board.

Voter verification phase. After the list of ballots $\vec{\mathsf{b}}$ has been published, each voter V_i can use her VVD_i to check whether (i) her ballot b_i appears in $\vec{\mathsf{b}}$ in the case she voted (if not, V_i can publish the acknowledgement she received from AS on the bulletin board which serves as binding evidence that AS misbehaved), or (ii) none of the ballots in $\vec{\mathsf{b}}$ contain her id in the case she abstained. In the latter case, the dispute cannot be resolved without further means: Did V_i vote but claims that she did not or did V_i not vote but AS used her id dishonestly?[6]

In both cases, however, it is well-known that, realistically, not all voters are motivated enough to perform these verification procedures, and even if they are, they often fail to do so (see, e.g., [38]). In our security analysis of Ordinos, we therefore assume that voters perform the verification procedures with a certain probability. In order to increase verification rates, fully automated verification, as deployed in the sElect voting system [12] turned out to be helpful and could be implemented in Ordinos as well.

Tallying phase. The list of ballots $\vec{\mathsf{b}}$ posted by AS is the input to the tallying phase, which works as follows.

1. Homomorphic aggregation. Each trustee T_k reads $\vec{\mathsf{b}}$ from the bulletin board B and verifies its correctness (i.e., whether duplicates and invalid ballots were removed). If this check fails, T_k aborts since AS should guarantee this. Otherwise, T_k homomorphically aggregates all vectors $\vec{\mathsf{c}}_i$ in $\vec{\mathsf{b}}$ entrywise and obtains a ciphertext vector $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ with n_{option} entries each of which encrypts the number of votes of the respective candidate/option.

2. Secure function evaluation. The trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ run the MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ with input $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ to securely evaluate the result function f_{tally}. They then output the election result according to f_{tally}, together with a NIZKP of correct evaluation π^{MPC}.[7]

Public verification phase. In this phase, every participant, including the voters or external observers, can verify

3. We note that, beyond Benaloh challenges, there exist several techniques in the literature (e.g., verification codes) to enable human voters to verify whether their ballots were cast by the voting devices as intended. Since these techniques are (typically) independent from whether the full tally is revealed or hidden, Ordinos could also be equipped with a different cast-as-intended mechanism than Benaloh challenges.

4. Just as for Helios, variants of the protocol are conceivable where the voter's ID is not part of the ballot and not put on the bulletin board or at least not next to her ballot (see, e.g., [36]).

5. If such a complaint is posted, it is in general impossible to resolve the dispute and decide exactly who is to be blamed: (i) AS who might not have replied as expected (but claims, for instance, that the ballot was not cast), or (ii) VSD_i who might not have submitted a ballot or forwarded the (correct) acknowledgement to VVD_i, or (iii) the voter who might not have cast a ballot but nevertheless claims that she has. Note that this is a very general problem which applies to virtually any remote voting protocol. In practice, the voter could ask the voting authority Auth to resolve the problem.

6. Variants of the protocol are conceivable where a voter is supposed to sign her ballot and the authentication server presents such a signature in the case of a dispute (see, e.g., [18]). This also helps in preventing so-called ballot stuffing.

7. π^{MPC} will typically consist of several NIZKPs, e.g., NIZKPs of correct decryption, etc. See also our instantiation in Section 7.

the correctness of the tallying procedure, in particular, the correctness of all NIZKPs.

Instantiation and implementation. As already mentioned, in Section 7 we show how to efficiently instantiate Ordinos with concrete primitives. In particular, we provide an efficient realization of a relevant class of result functions which can be computed in a tally-hiding fashion, e.g., for publishing only the winner of an election or certain subsets or rankings of candidates. In Section 8, we describe our implementation and provide benchmarks. Our model and security analysis of Ordinos, presented in the following sections are, however, more general and apply to arbitrary instantiations of Ordinos as long as certain assumptions are met.

3. Model

In this section, we formally model the Ordinos voting protocol. This model is the basis for our security analysis of Ordinos carried out in the following sections. Our model of Ordinos is based on a general computational model proposed in [39]. This model introduces the notions of processes, protocols, instances, and properties, which we recall in Appendix A.

Modeling of Ordinos. The Ordinos voting protocol can be modeled in a straightforward way as a protocol $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, f_{\mathsf{tally}})$ in the above sense, as described next (see the technical report [34] for full technical details). By n_{voters} we denote the number of voters V_i and by n_{trustees} the number of trustees T_k. By μ we denote a probability distribution on the set of choices C, including abstention. An honest voter makes her choice according to this distribution.[8] This choice is called the *actual choice* of the voter. By $p_{\mathsf{verify}} \in [0,1]$ we denote the probability that an honest voter V_i performs the checks described in Section 2, voter verification phase. We denote the probability that a given (arbitrary) ballot is audited by an honest voter by $p_{\mathsf{audit}} \in [0,1]$.[9] As before, with f_{tally} we denote the result function.

In our model of Ordinos, 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. We assume S and the bulletin board B to be honest, i.e., they are never corrupted. While S is merely a virtual entity, in reality, B should be implemented in a distributed way (see, e.g., [40], [41]). As usual, we also require that the verification devices, VVD_i, are honest.

4. End-to-End Verifiability

In this section, we formally establish the level of (end-to-end) verifiability provided by Ordinos. We show that Ordinos inherits the level of verifiability from the original Helios voting protocol. In particular, this implies that this level is independent of f_{tally}, and hence, the degree to which f_{tally} hides the tally. This might be a bit surprising at first since less information being published might mean that a system provides less verifiability.

Our analysis of Ordinos in terms of verifiability uses the generic verifiability framework by Küsters et al. (the *KTV framework*, originally presented in [42] with some refinements in [39]). We briefly recall this framework in Section 4.1 along with some instantiation needed for our analysis of Ordinos. Beyond its expressiveness, the KTV framework is particularly suitable to analyze Ordinos because (i) it does not make any specific assumptions on the result function of the voting protocol, and (ii) it can, as illustrated here, also be applied to MPC protocols. The results of our verifiability analysis of Ordinos are then presented in Section 4.2.

4.1. Verifiability Framework

In a nutshell, an e-voting system provides verifiability if the probability that the published result is not correct but no one complains and no checks fail, i.e., no misbehavior is observed, is small (bounded by some small $\delta \in [0,1]$).

Judge. More specifically, the KTV verifiability definition assumes a *judge* J whose role is to accept or reject a protocol run by writing accept or reject on a dedicated channel $\mathsf{decision}_{\mathsf{J}}$. To make a decision, the judge runs a so-called *judging procedure*, which performs certain checks (depending on the protocol specification), such as the verification of all zero-knowledge proofs in Ordinos and taking voter complaints into account. Intuitively, J accepts a run if the protocol run looks as expected. The input to the judge is solely public information, including all information and complaints (e.g., by voters) posted on the bulletin board. Therefore, the judge can be thought of as a "virtual" entity: the judging procedure can be carried out by any party, including external observers and even voters themselves. The specification of the judging procedure for Ordinos follows quite easily from the description in Section 2 and it is provided with full details in Appendix D.

Goal. The KTV verifiability definition is centered around the notion of a *goal* of a protocol P, such as Ordinos or an MPC protocol. Formally, a goal γ is simply a set of protocol runs. The goal γ specifies those runs which are "correct" in some protocol-specific sense. For e-voting, the goal would contain those runs where the announced election result corresponds to the actual choices of the voters.

In what follows, we describe the goal $\gamma(k,\varphi)$ that we use to analyze end-to-end verifiability of Ordinos. This goal has already been applied in [36] to the original Helios protocol, where here we use a slightly improved version suggested in [39]. The parameter φ is a Boolean formula that describes which protocol participants are assumed to be honest in a run, i.e., not corrupted by the adversary. For Ordinos, we set $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B}) \wedge \bigwedge_{i=1}^{n_{\mathsf{voters}}} \mathsf{hon}(\mathsf{VVD}_i)$, i.e., the scheduler S, the judge J, the bulletin board B, and all of the voters' verification devices VVD are assumed to be honest. On a high level, the parameter k denotes the maximum number

8. This in particular models that adversaries know this distribution. In reality, the adversary might not know this distribution precisely. This, however, makes our security results only stronger.

9. Following [36], one could as well consider a sequence of audit probabilities to model that the probability of the voter auditing a ballot decreases with the number of audits she has performed. Furthermore, one could assign different verification probabilities for different voters. However, both refinements would result in a fairly complicated analysis that is not specific to Ordinos and its tally-hiding property.

of choices made by honest voters the adversary is allowed to manipulate. So, roughly speaking, altogether the goal $\gamma(k,\varphi)$ contains all those runs of a (voting or MPC) protocol P where either (i) at least one of the parties S, J, B, or some VVD_i have been corrupted (i.e., φ is false) or (ii) where none of them have been corrupted (i.e., φ holds true) and where the adversary has manipulated at most k votes/inputs of honest voters/input parties. We formally define the goal $\gamma(k,\varphi)$ in Appendix B.

Verifiability. Now, the idea behind the verifiability definition is very simple. The judge J should accept a run only if the goal γ is met: as discussed, if $\gamma = \gamma(k,\varphi)$, then this essentially means that the published election result corresponds to the actual choices of the voters up to k votes of honest voters. More precisely, the definition requires that the probability (over the set of all protocol runs) that the goal γ is not satisfied but the judge nevertheless accepts the run is δ-bounded: 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 $\ell > \ell_0$. Although $\delta = 0$ is desirable, this would be too strong for almost all e-voting protocols. For example, typically not all voters check whether their ballot appears on the bulletin board, giving an adversary A the opportunity to manipulate or drop some ballots without being detected. Therefore, $\delta = 0$ cannot be achieved in general in e-voting protocols. The parameter δ is called the *verifiability tolerance* of the protocol.

By $\Pr[\pi^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ we denote the probability that π, with security parameter 1^ℓ, produces a run which is not in γ but nevertheless accepted by J.

Definition 1 (Verifiability[10])**.** *Let P be a protocol with the set of agents Σ. Let $\delta \in [0,1]$ be the tolerance, $\mathsf{J} \in \Sigma$ be the judge, and γ be a goal. Then, we say that* the protocol P is (γ,δ)-verifiable by the judge J *if for all adversaries π_A and $\pi = (\hat{\pi}_\mathsf{P} \| \pi_\mathsf{A})$, the probability $\Pr[\pi^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ is δ-bounded as a function of ℓ.*

4.2. End-to-End Verifiability of Ordinos

We are now able to precisely state and prove the verifiability level offered by Ordinos according to Definition 1. The level depends on the voter verification rate p_verify and the voter auditing rate p_audit, as described in Section 3.

Assumptions. We prove the verifiability result for Ordinos for the goal $\gamma(k,\varphi)$, with $\gamma(k,\varphi)$ as defined in Section 4.1, and under the following assumptions:

(V1) The public-key encryption scheme $\mathcal{E}$ is correct (for verifiability, IND-CPA-security is not needed), $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} are NIZKPs, and the signature scheme $\mathcal{S}$ is EUF-CMA-secure.

(V2) The scheduler S, the bulletin board B, the judge J, and all voter verification devices VVD_i are honest, i.e., $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B}) \wedge \bigwedge_{i=1}^{n_\mathsf{voters}} \mathsf{hon}(\mathsf{VVD}_i)$.

(V3) The MPC protocol P_MPC is $(\gamma(0,\varphi),0)$-verifiable, meaning that if the output of P_MPC does not correspond to its input, then this can always be detected publicly.

Verifiability. The judging procedure performed by J essentially involves checking signatures and NIZKPs. If one of these checks fails, the judge rejects the protocol run and hence the result. Also, J takes care of voter complaints as discussed in Section 2.

Intuitively, the following theorem states that the probability that in a run of Ordinos more than k votes of honest voters have been manipulated but the judge J nevertheless accepts the run is bounded by $\delta_k(p_\mathsf{verify}, p_\mathsf{audit})$.

Theorem 1 (Verifiability)**.** *Under the assumptions (V1) to (V3) stated above, the protocol $\mathsf{P}_\mathsf{Ordinos}(n_\mathsf{voters}, n_\mathsf{trustees}, \mu, p_\mathsf{verify}, p_\mathsf{audit}, f_\mathsf{tally})$ is $(\gamma(k,\varphi), \delta_k(p_\mathsf{verify}, p_\mathsf{audit}))$-verifiable by the judge J where*

$$\delta_k(p_\mathsf{verify}, p_\mathsf{audit}) = \max\left(1 - p_\mathsf{verify}, 1 - p_\mathsf{audit}\right)^{\lceil \frac{k+1}{2} \rceil}.$$

The intuition and reasoning behind this theorem is as follows: In order to break $\gamma(k,\varphi)$, the adversary has to manipulate more than k votes of honest voters (actually less, see below). Due to the NIZKPs and signatures employed, we can show that such a manipulation is not detected only if none of the affected honest voters perform their auditing or verification procedure. The probability for this is $\max\left(1 - p_\mathsf{verify}, 1 - p_\mathsf{audit}\right)^{\lceil \frac{k+1}{2} \rceil}$: the exponent is not $k+1$, as one might expect, but $\lceil \frac{k+1}{2} \rceil$ because, according to the formal definition of $\gamma(k,\varphi)$, if the adversary changes one vote of an honest voter from one choice to another, the distance between the actual result and the manipulated one increases by two.

Observe that assumption (V2), in particular that all voter verification devices are honest, is necessary for the verifiability result of Ordinos. In fact, if for a given voter V_i both her VSD_i and her VVD_i colluded, then VSD_i could replace the chosen candidate by a different one and VVD_i could incorrectly claim that VSD_i behaved honestly. This is a general property of Benaloh challenges which applies, for example, to Helios, too.

Note that, possibly surprisingly, our results show that the level of verifiability provided by Ordinos is independent of the result function f_tally, and hence, independent of how much of the full tally is hidden by the desired output of f_tally: less information might give the adversary more opportunities to manipulate the result without being detected. Roughly speaking, the reason is that the goal $\gamma(k,\varphi)$ is concerned with the actual *input* to the voting protocol (as provided by the voters) rather than its output (e.g., the complete result or only the winner).

The correctness of Theorem 1 follows immediately from an even stronger result. In fact, Ordinos even provides *accountability* which is a stronger form of verifiability as demonstrated in [42]. For verifiability, one requires only that, if some goal of the protocol is not achieved (e.g., the election outcome does not correspond to how the voters actually voted), then the judge does not accept such a run (more precisely, he accepts it with a small probability only). The judge, however, is not required to blame misbehaving parties. Conversely, accountability requires that misbehaving parties are blamed, an important property in practice as misbehavior should be identifiable and have consequences: accountability serves as a deterrent. Now,

10. We note that the original definition in [42] also captures soundness/fairness: if the protocol runs with a benign adversary, which, in particular, would not corrupt parties, then the judge accepts all runs. This kind of fairness/soundness can be considered to be a sanity check of the protocol, including the judging procedure, and is typically easy to check. For brevity of presentation, we omit this condition here.

analogously to the verifiability result presented above, Ordinos has the same level of accountability as Helios. Due to space limitations, we formally state accountability of Ordinos in Appendix E and provide a complete proof in our technical report [34].

5. Privacy

In this section, we carry out a rigorous analysis of the vote privacy of Ordinos. We show that the privacy level of Ordinos is essentially ideal assuming the strongest possible class of adversaries, as explained next.

Observe that if the adversary controls the authentication server, say, and does not care at all about being caught cheating, then he could drop all ballots except for one. Hence, the final result would only contain a single choice so that the respective voter's privacy is completely broken. This privacy attack applies to virtually all remote e-voting systems, including Helios, as already observed in [12], and later further investigated in [43].

Therefore, in general, a voting protocol can only provide vote privacy if an adversary does not drop or replace "too many" ballots prior to the tallying phase; this is necessary to ensure that a single voter's choice is "hidden" behind sufficiently many other votes. This class of adversaries is the strongest one for which privacy can still be guaranteed under realistic assumptions.[11] Now, for this class of adversaries, we show that the privacy level of Ordinos coincides with the privacy level of an ideal voting protocol, where merely the election result according to the result function considered is published.

To better understand the relationship between the privacy level of a voting protocol and the result function used, in Section 6 we study the level of privacy of the ideal voting protocol in depth parameterized by the result function, which then also precisely captures the level of privacy of Ordinos.

We first introduce the class of adversaries as sketched above, and present the privacy definition we use. We then state the privacy result for Ordinos.

5.1. Risk-Avoiding Adversaries

The privacy definition we use (see Section 5.2) requires that, except with a small probability, the adversary should not be able to distinguish whether some voter (called the *voter under observation*) voted for ch_0 or ch_1 when she runs her honest program. Now, an adversary who controls the authentication server, say, could drop or replace all ballots except for the one of the voter under observation. The final result would then contain only the vote of the voter under observation, and hence, the adversary could easily tell how this voter voted, which breaks privacy.

However, such an attack is extremely risky: recall that the probability of being caught grows exponentially in the number k of honest votes that are dropped or changed (see Section 4). Thus, in the above attack where k is big, the probability of the adversary to get caught would be very close to 1. In the context of e-voting, where misbehaving parties that are caught have to face severe penalties or loss of reputation, this attack seems completely unreasonable.

A more reasonable adversary would possibly consider dropping some small number of votes, for which the risk of being caught is not too big, in order to weaken privacy to some degree. To analyze this trade-off, we use the notion of *k-risk-avoiding adversaries* that was originally introduced in [12] and adjust it to our setting. (In [12], such adversaries are called k-semi-honest. However, this term is misleading since these adversaries do not have to follow the protocol.)

Intuitively, a k-risk-avoiding adversary would not manipulate too many votes of honest voters. More specifically, he would produce runs in which the goal $\gamma(k,\varphi)$ holds true. From the (proof of the) verifiability result obtained in Section 4, we know that whenever an adversary decides to break $\gamma(k,\varphi)$ his risk of being caught is at least $1-\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}})$: Consider a run in which $\gamma(k,\varphi)$ does not hold true and in which all random coins are fixed except for the ones that determine which honest voters perform their verification procedure. Then, the probability taken over these random coins that the adversary gets caught is at least $1-\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}})$. That is, such an adversary knows upfront that he will be caught with a probability of at least $1-\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}})$ which converges exponentially fast to 1 in k. Therefore, an adversary not willing to take a risk of being caught higher than $1-\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}})$ would never cause $\gamma(k,\varphi)$ to be violated, and hence, manipulate too many votes.

This motivates the following definition: an adversary is *k-risk-avoiding in a run of a protocol* P if the goal $\gamma(k,\varphi)$ is satisfied in this run. An adversary (of an instance π of P) is *k-risk-avoiding* if he is k-risk-avoiding with overwhelming probability (over the set of all runs of π).

5.2. Definition of Privacy

For our privacy analysis of Ordinos, we use the privacy definition for e-voting protocols proposed in [45]. This definition allows us to *measure* the level of privacy a protocol provides, unlike other definitions (see, e.g., [46]).

As briefly mentioned above, privacy of an e-voting protocol is formalized 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 ch_0 or ch_1.

To define this notion formally, we first introduce the following notation. Let P be an e-voting protocol (in the sense of Section 3 with voters, authorities, result function, etc.). Given a voter $\mathsf{V}_{\mathsf{obs}}$ and $\mathsf{ch} \in \mathsf{C}$, we now consider instances of P of the form $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}) \| \pi^* \| \pi_{\mathsf{A}})$ where $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})$ is the honest program of the voter $\mathsf{V}_{\mathsf{obs}}$ under observation who takes ch 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 Ordinos, π^* would include the scheduler, the bulletin board, the authentication server, all other voters, and all trustees.

Let $\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1]$ denote the probability that the adversary writes the output 1 on some

11. We note that there are privacy results where the class of adversaries considered is not restricted (see, e.g., [44]), but these results essentially assume that manipulations are not possible or manipulations are abstracted away in the modeling of the protocols (see also the discussions in [12], [43]).

dedicated channel in a run of $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})\|\pi^*\|\pi_{\mathsf{A}})$ with security parameter ℓ and some $\mathsf{ch} \in \mathsf{C}$, where the probability is taken over the random coins used by the parties in $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})\|\pi^*\|\pi_{\mathsf{A}})$.

Now, similarly to [45], we can define vote privacy. The definition is w.r.t. a mapping $\mathcal{A}$ which maps an instance π of a protocol (excluding the adversary) to a set of admissible adversaries; for Ordinos, for example, only k-risk-avoiding adversaries are admissible.

Definition 2 (Privacy)**.** *Let* P *be a voting protocol,* $\mathsf{V}_{\mathsf{obs}}$ *be the voter under observation,* $\mathcal{A}$ *be a mapping as explained above, and* $\delta \in [0,1]$*. Then,* P achieves δ-privacy (w.r.t. $\mathcal{A}$)*, if the difference between the probabilities* $\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_0)\|\pi^*\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1]$ *and* $\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_1)\|\pi^*\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1]$ *is* δ*-bounded as a function of the security parameter* 1^ℓ*, for* π^* *as defined above, for all choices* $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C} \setminus \{\mathsf{abstain}\}$ *and adversaries* π_{A} *that are admissible for* $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})\|\pi^*$ *for all possible choices* $\mathsf{ch} \in \mathsf{C}$*, i.e.,* $\pi_{\mathsf{A}} \in \bigcap_{\mathsf{ch}\in\mathsf{C}} \mathcal{A}(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})\|\pi^*)$.

The requirement $\mathsf{ch}_0, \mathsf{ch}_1 \neq \mathsf{abstain}$ says that we allow the adversary to distinguish whether or not a voter voted at all.

Since δ often depends on the number $n_{\mathsf{voters}}^{\mathsf{honest}}$ of honest voters, privacy is typically formulated w.r.t. this number: the bigger the number of honest voters, the smaller δ should be, i.e., the higher the level of privacy. Note that even for an ideal e-voting protocol, where voters privately enter their votes and the adversary sees only the election outcome, consisting of the number of votes per candidate say, δ cannot be 0: there may, for example, 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 clearly see how the voter under observation voted. Hence, it is important to also take into account the probability distribution used by the honest voters to determine their choices; as already mentioned in Section 3, we denote this distribution by μ. Moreover, the level of privacy, also of an ideal voting protocol, will depend on the result function, i.e., the information contained in the published result, as further investigated in Section 6.

5.3. Privacy of Ordinos

We now prove that Ordinos provides a high level of privacy w.r.t. k-risk-avoiding adversaries and in the case that at most $t-1$ trustees are dishonest, where t is the decryption threshold of the underlying encryption scheme: clearly, if t trustees were dishonest, privacy cannot be guaranteed because an adversary could simply decrypt every ciphertext in the list of ballots. By "high level of privacy" we mean that Ordinos provides δ-privacy for a δ that is very close to the ideal one.

More specifically, the formal privacy result for Ordinos is formulated w.r.t. an ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$. In this protocol, honest voters pick their choices according to the distribution μ. In every run, there are $n_{\mathsf{voters}}^{\mathsf{honest}}$ many honest voters and n_{voters} 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 result function f_{res}. In Section 6, we analyze the privacy level $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)}(f_{\mathsf{res}})$ this ideal protocol has depending on the given parameters.

Assumptions. To prove that the privacy level of Ordinos is essentially the ideal one, we make the following assumptions about the primitives we use (see also Section 2):

(P1) The public-key encryption scheme $\mathcal{E}$ is IND-CPA-secure, the signatures are EUF-CMA-secure, and $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} are NIZKPs.

(P2) The MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ realizes (in the sense of universal composability [47], [48]) an ideal MPC protocol which essentially takes as input a vector of ciphertexts and returns f_{tally} evaluated on the corresponding plaintexts (see Appendix F).

The level of privacy of Ordinos clearly depends on the number of ballots cast by honest voters. In our analysis, to have a guaranteed number of votes by honest voters, we assume that honest voters do not abstain from voting. Note that the adversary would anyway know which voters abstained and which did not. Technically, we define:

(P3) The probability of abstention is 0 in μ.

(P4) For each instance π of $\mathsf{P}_{\mathsf{Ordinos}}$, the set $\mathcal{A}(\pi)$ of admissible adversaries for π is defined as follows. An adversary π_{A} belongs to $\mathcal{A}(\pi)$ iff it satisfies the following conditions: (i) π_{A} is k-risk-avoiding for π, (ii) the probability that π_{A} corrupts more than $t-1$ trustees in a run of $\pi\|\pi_{\mathsf{A}}$ is negligible, (iii) the probability that π_{A} corrupts more than $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters in a run of $\pi\|\pi_{\mathsf{A}}$ is negligible, and (iv) the probability that π_{A} corrupts an honest voter's supporting or verification device is negligible.[12]

Now, the privacy theorem for Ordinos says that the level of privacy of Ordinos for this class of adversaries is the same as the one for the ideal protocol with $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ honest voters.

Theorem 2 (Privacy)**.** *Under the assumptions (P1) to (P4) stated above and with the mapping* $\mathcal{A}$ *as defined above, the voting protocol* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, f_{\mathsf{tally}})$ *achieves a privacy level of* $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k, \mu)}(f_{\mathsf{res}})$ *w.r.t.* $\mathcal{A}$ *where* f_{res} *first counts the number of votes per candidate and then evaluates* f_{tally}*.*[13]

The proof is provided in our technical report [34], where we reduce the privacy game for Ordinos with $n_{\mathsf{voters}}^{\mathsf{honest}}$ honest voters, as specified in Definition 2, to the privacy game for the ideal voting protocol with $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ voters, by a sequence of games.

As discussed, since the risk of being caught cheating increases exponentially with k, the number of changed votes k will be rather small in practice. But then the privacy theorem tells us that manipulating just a few votes of honest voters does not buy the adversary much in terms

12. We note that, in principle, the following privacy result also holds true if the honest voters' verification devices are dishonest. This is because, in our model, an honest voter chooses a "fresh" candidate according to μ each time after having audited her ballot. If, however, an honest voter always used her actual candidate when auditing her ballot, then the verification device needs to be honest for vote privacy, too.

13. Recall that in Ordinos, the tallying function f_{tally} is evaluated over the homomorphically aggregated votes, i.e., the vector that encrypts the total number of votes for each candidate. Conversely, the more general result function f_{res} of the ideal voting protocol receives the voters' choices as input. Hence, f_{res} needs to first aggregate the votes and then apply f_{tally}.

of weakening privacy. In fact, as illustrated in Section 6, even with only 15 honest voters the level of privacy does not decrease much when the adversary changes the honest votes by only a few. Conversely, the result function can very well have a big effect on the level of privacy of a tally-hiding system: whether only the winner of an election is announced or the complete result typically has a big effect on privacy.

6. Privacy of the Ideal Protocol

As discussed in Section 5.2, the level δ of privacy is bigger than zero for virtually every voting protocol, as some information is always leaked by the result of the election. In order to have a lower bound on δ for all tallying-hiding voting protocols (where the results are of the form considered below), including Ordinos, we now determine the optimal value of δ for the ideal (tally-hiding) voting protocol.

We have already sketched the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ in Section 5. We now formally analyze how the privacy level $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)}(f_{\mathsf{res}})$ of the ideal voting protocol depends on the specific result function f_{res} in relation to the number of voters n_{voters}, the number of honest voters $n_{\mathsf{voters}}^{\mathsf{honest}}$, and the probability distribution μ according to which the honest voters select their choices.

We developed a formula for the optimal level of privacy $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(f_{\mathsf{res}})$ for the the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$. The following theorem shows that the level $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(f_{\mathsf{res}})$ is indeed optimal (see Appendix I for the precise formula and [34] for the proof of the theorem as well as further results on privacy).

Theorem 3. *The ideal protocol* $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ *achieves a privacy level of* $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(f_{\mathsf{res}})$. *Moreover, it does not achieve* δ'*-privacy for any* $\delta' < \delta^{\mathsf{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(f_{\mathsf{res}})$.

Impact of hiding the tally. In the following, we compare the levels of privacy of the ideal protocol for some practically relevant result functions (computed in a tally-hiding fashion), namely f_{rank} where the ranking of all candidates is published (but not the number of votes per candidate), f_{win} where only the winner of the election is published (again, no number of votes), and f_{complete} where the whole result of the election is published, i.e., the number of votes per candidate (as in almost all verifiable e-voting systems, including, e.g., Helios). We denote the corresponding privacy levels by $\delta^{\mathsf{ideal}}_{\mathsf{rank}}$, $\delta^{\mathsf{ideal}}_{\mathsf{win}}$, and $\delta^{\mathsf{ideal}}_{\mathsf{complete}}$, respectively.

In general, more information means less privacy. Depending on the distribution on the candidates, in general $\delta^{\mathsf{ideal}}_{\mathsf{complete}}$ is bigger than $\delta^{\mathsf{ideal}}_{\mathsf{rank}}$ which in turn is bigger than $\delta^{\mathsf{ideal}}_{\mathsf{win}}$; see Figure 1 for an example.

Revealing the complete result can lead to much worse privacy. This is demonstrated already by Figure 1.

The balancing attack. As just mentioned, the difference between $\delta^{\mathsf{ideal}}_{\mathsf{win}}$ and $\delta^{\mathsf{ideal}}_{\mathsf{complete}}$ can be very big if one choice has a bigger probability. We now illustrate that sufficiently many dishonest voters can help to cancel out the advantage of tally-hiding functions in terms of the

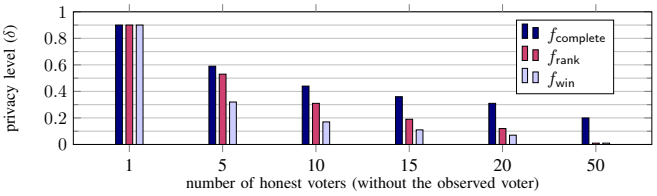

Figure 1: Level of privacy (δ) for the ideal protocol with three candidates, $p_1 = 0.6$, $p_2 = 0.3$, $p_3 = 0.1$ and no dishonest voters.

privacy of single voters. We call this the *balancing attack*. More specifically, the adversary can use dishonest voters to balance the probabilities for candidates. For illustration purposes consider the case of ten honest voters and two candidates, where the first candidate has a probability of 0.9. Now, if eight dishonest voters are instructed to vote for the second candidate, the expected total number of votes for each candidate is nine. Hence, the choice of the voter under observation is indeed relatively often crucial for the outcome of f_{win}, given this distribution. As the number of dishonest voters is typically small in comparison to the number of honest voters, this balancing attack is not effective for big elections, but it might be in small elections, with a few voters and a few candidates.

Sometimes ranking is not better than the complete result. If candidates are distributed uniformly, it is easy to show that $\delta^{\mathsf{ideal}}_{\mathsf{complete}} = \delta^{\mathsf{ideal}}_{\mathsf{rank}}$. The reason is that the best strategy for the adversary to decide whether the observed voter voted for i or j is to choose i if i gets more votes than j, and this strategy is applicable even if only the ranking is published. We note that f_{win} is still better, i.e., $\delta^{\mathsf{ideal}}_{\mathsf{win}} < \delta^{\mathsf{ideal}}_{\mathsf{complete}} = \delta^{\mathsf{ideal}}_{\mathsf{rank}}$.

We finally note that due to Theorem 2, these results directly carry over to Ordinos. Also, they yield a lower bound for privacy of tally-hiding systems in general.

7. Instantiation of Ordinos

In this section, we provide an instantiation of the generic Ordinos protocol with concrete cryptographic primitives; in Section 8, we then describe our implementation of this instantiation of Ordinos and provide several benchmarks, demonstrating its practicability.

Result functions supported by our instantiation. Our instantiation can be used to realize many different practically relevant result functions which are computed in a tally-hiding way. They all have in common that they reveal chosen parts of the final candidates' ranking (with or without the number of votes a candidate received), for example, the complete ranking, only the winner of the election, the ranking or the set of the best/worst k candidates, only the winner under the condition that she received at least, say, fifty percent of the votes, etc. We describe how to realize these different variants below.

Cryptographic primitives. For our instantiation we use the standard threshold variant of the Paillier public-key encryption scheme [31] as the $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E}$. The main reason for choosing Paillier instead of exponential ElGamal [49] (as in the original Helios protocol) is that for the MPC protocol below the decryption algorithm Dec of $\mathcal{E}$ needs to be

efficient. This is not the case for exponential ElGamal, where decryption requires some brute forcing in order to obtain the plaintext.

The NIZKP π^{Enc} that the voters have to provide for proving knowledge and well-formedness of the chosen $\mathsf{ch} \in \mathsf{C}$ can be based on a standard proof of plaintext knowledge for homomorphic encryption schemes, as described in [33], in combination with [50].

The NIZKP $\pi^{\mathsf{KeyShareGen}}$ depends on the way public/private keys are shared among the trustees. One could, for example, employ the protocol by Algesheimer et al. [51], which includes a NIZKP $\pi^{\mathsf{KeyShareGen}}$. Also, solutions based on trusted hardware are conceivable. Note that setting up key shares for the trustees is done offline, before the election starts, and hence, this part is less time critical. For simplicity, in our implementation (see Section 8), we generate key shares centrally for the trustees, essentially playing the role of a trusted party in this respect.

As for the signature scheme $\mathcal{S}$, any EUF-CMA-secure can be used.

The most challenging part of the instantiating of Ordinos is to construct an efficient MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ for evaluating practically relevant result functions in a tally-hiding way, which at the same time satisfies the conditions for verifiability (see Section 4.2) as well as privacy (see Section 5.3). We now describe such a construction.

Overview of $\mathsf{P}_{\mathsf{MPC}}$. The cornerstone of our instantiation of $\mathsf{P}_{\mathsf{MPC}}$ is a secure MPC protocol $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ that takes as input two secret integers x, y and outputs a secret bit b that determines whether $x \geq y$, i.e., $b = (x \geq y)$.

We instantiate $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ with the "greater-than" MPC protocol by Lipmaa and Toft [32] which has been proposed for an arbitrary arithmetic blackbox (ABB), which in turn we instantiate with the Paillier public-key encryption scheme, equipped with NIZKPs from [33]. Lipmaa and Toft demonstrated that their protocol is secure in the malicious setting. Due to the NIZKPs this protocol employs, it provides verifiability in our specific instantiation, i.e., if the outcome of the protocol is incorrect, this is detected.[14] Importantly, the protocol by Lipmaa and Toft comes with sublinear online complexity which is superior to all other "greater-than" MPC protocols to the best of our knowledge. This is confirmed by our benchmarks which show that the communication overhead is quite small (see Section 8). Similarly, we also use the secure MPC protocol $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ by Lipmaa and Toft [32] which secretly evaluates equality of two secret integers.

Now, $\mathsf{P}_{\mathsf{MPC}}$ is carried out in two phases in Ordinos. In the first phase, given the vector $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ of the encrypted number of votes per candidate (see Section 2), the trustees collaboratively run several instances of the greater-than-test $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ in order to obtain a ciphertext vector $\vec{\mathsf{c}}_{\mathsf{rank}}$ which encrypts the overall ranking of the candidates. In the second phase, the resulting ciphertext vector (plus possibly $\vec{\mathsf{c}}_{\mathsf{unsorted}}$) is used to realize the desired result function in a tally-hiding way.

These two phases are described in more detail in what follows.

First phase: Computing the secret ranking. Recall that in Ordinos each ballot b is a tuple $(\mathsf{id}, \vec{\mathsf{c}}, \pi^{\mathsf{Enc}})$, where id is the voter's id, $\vec{\mathsf{c}} = (\mathsf{c}[1], \ldots, \mathsf{c}[n_{\mathsf{option}}])$ is a ciphertext vector that encrypts the voter's choice, and π^{Enc} is a NIZKP for proving knowledge of the choice/plaintexts and well-formedness of the ciphertext vector (e.g., for proving that exactly a single $\mathsf{c}[i] \in \vec{\mathsf{c}}$ encrypts 1, while all other ciphertexts in $\vec{\mathsf{c}}$ encrypt 0, if a voter can give only one vote to one candidate/option). The input to the tallying phase consists of the ballots with valid NIZKPs π^{Enc}.

In the first step of the tallying phase, the ciphertext vectors $\vec{\mathsf{c}}$ of all valid ballots are homomorphically summed up to obtain a ciphertext vector $\vec{\mathsf{c}}_{\mathsf{unsorted}} = (\vec{\mathsf{c}}_{\mathsf{unsorted}}[1], \ldots, \vec{\mathsf{c}}_{\mathsf{unsorted}}[n_{\mathsf{option}}])$ where $\vec{\mathsf{c}}_{\mathsf{unsorted}}[i]$ encrypts the total number of votes for the ith candidate.

In the second step, we essentially apply the direct sorting algorithm [52] to $\vec{\mathsf{c}}_{\mathsf{unsorted}}$. More precisely, in what follows we denote by $\mathsf{Dec}(\mathsf{c})$ the distributed decryption of a ciphertext c by the trustees. Now, for each pair of candidates/options, say i and j, the trustees run the equality test $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ with input $(\vec{\mathsf{c}}_{\mathsf{unsorted}}[i], \vec{\mathsf{c}}_{\mathsf{unsorted}}[j])$ and output $\mathsf{c}_{\mathsf{eq}}[i,j]$ which decrypts to 1 if $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{unsorted}}[i]) = \mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{unsorted}}[j])$ and to 0 otherwise. Clearly, the trustees need to run the protocol $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ only $\frac{(n_{\mathsf{option}}-1)n_{\mathsf{option}}}{2}$ many times because $\mathsf{c}_{\mathsf{eq}}[i,i]$ always decrypts to 1 and $\mathsf{c}_{\mathsf{eq}}[i,j] = \mathsf{c}_{\mathsf{eq}}[j,i]$. In fact, this step (which comes with almost no communicational and computational overhead) will be used to speed up the following step.

For each pair of candidates/options, say i and j, the trustees now run the greater-than protocol $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ with input $(\vec{\mathsf{c}}_{\mathsf{unsorted}}[i], \vec{\mathsf{c}}_{\mathsf{unsorted}}[j])$ and output $\mathsf{c}_{\mathsf{gt}}[i,j]$ which decrypts to 1 if and only if $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{unsorted}}[i]) \geq \mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{unsorted}}[j])$ and to 0 otherwise. Thanks to the previous step, the trustees need to run the $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ protocol only $\frac{(n_{\mathsf{option}}-1)n_{\mathsf{option}}}{2}$ many times because $\mathsf{c}_{\mathsf{gt}}[i,i]$ always decrypts to 1 and $\mathsf{c}_{\mathsf{gt}}[j,i]$ can easily be computed from $\mathsf{c}_{\mathsf{gt}}[i,j]$ because $\mathsf{c}_{\mathsf{gt}}[j,i] = \mathsf{Enc}(1) - \mathsf{c}_{\mathsf{gt}}[i,j] + \mathsf{c}_{\mathsf{eq}}[i,j]$.

All of these ciphertexts are stored in an $n_{\mathsf{option}} \times n_{\mathsf{option}}$ *comparison matrix* M_{rank}:

$$\begin{bmatrix} \mathsf{c}_{\mathsf{gt}}[1,1] & \mathsf{c}_{\mathsf{gt}}[2,1] & \cdots & \mathsf{c}_{\mathsf{gt}}[n_{\mathsf{option}},1] \\ \mathsf{c}_{\mathsf{gt}}[1,2] & \mathsf{c}_{\mathsf{gt}}[2,2] & \cdots & \mathsf{c}_{\mathsf{gt}}[n_{\mathsf{option}},2] \\ \vdots & \vdots & \vdots & \vdots \\ \mathsf{c}_{\mathsf{gt}}[1,n_{\mathsf{option}}] & \mathsf{c}_{\mathsf{gt}}[2,n_{\mathsf{option}}] & \cdots & \mathsf{c}_{\mathsf{gt}}[n_{\mathsf{option}},n_{\mathsf{option}}] \end{bmatrix}$$

Based on this matrix, everyone can compute an encrypted overall ranking of the candidates: for each column i of M_{rank}, the homomorphic sum $\vec{\mathsf{c}}_{\mathsf{rank}}[i] = \sum_{j=1}^{n_{\mathsf{option}}} \mathsf{c}_{\mathsf{gt}}[i,j]$ encrypts the total number of pairwise "wins" of the ith candidate against the other candidates, including i itself. For example, if the ith candidate is the one which has received the fewest votes, then $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{rank}}[i]) = 1$ because $\mathsf{Dec}(\mathsf{c}_{\mathsf{gt}}[i,i]) = 1$, and if it has received the most votes, then $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{rank}}[i]) = n_{\mathsf{option}}$. We collect all of these ciphertexts in a *ranking vector* $\vec{\mathsf{c}}_{\mathsf{rank}} = (\vec{\mathsf{c}}_{\mathsf{rank}}[1], \ldots, \vec{\mathsf{c}}_{\mathsf{rank}}[n_{\mathsf{option}}])$.

Second phase: Calculating the election result. First note that, for example, $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{rank}}) = (6,6,6,3,3,3)$ is a possible plaintext ranking vector, which says that the first three candidates are the winners, they are on position 1. As a result, no one is on position 2 or 3 (following common conventions). The last three candidates are on position 4; no one is on position 5 or 6. Note that, for example, $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{rank}}) = (6,6,6,3,3,2)$ is not a possible plaintext ranking vector.

14. It even provides individual accountability (see Appendix G).

Using $\vec{c}_{\mathsf{rank}}$ and $\vec{c}_{\mathsf{unsorted}}$, we can, for example, realize the following families of result functions in a tally-hiding way and combinations thereof.

Revealing the candidates on the first n positions only. There are three variants:

(i) Without ranking, i.e., the set of these candidates: For all candidates i, the trustees run the greater-than test $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ with input $(\vec{c}_{\mathsf{rank}}[i], \mathsf{Enc}(n_{\mathsf{option}} - n + 1))$ and decrypt the resulting ciphertext. Candidate i belongs to the desired set iff the decryption yields 1. The case $n = 1$ means that only the winner(s) is/are revealed.

(ii) With ranking: For all candidates i, the trustees execute the equality-test $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ with input $(\vec{c}_{\mathsf{rank}}[i], \mathsf{Enc}(n_{\mathsf{option}} - k + 1))$ for all $1 \leq k \leq n$ and decrypt the resulting ciphertext. Then, candidate i is on the k-th position iff for k the test returns 1. If no test returns 1, i is not among the candidates on the first n positions.

(iii) Including the number of votes: The trustees decrypt the ciphertext $\vec{c}_{\mathsf{unsorted}}[i]$ of each candidate i that has been output in the previous variant.

Revealing the candidates on the last n positions. Observe that we can construct a less-than test $\mathsf{P}^{\mathsf{lt}}_{\mathsf{MPC}}$ from the results of the equality tests $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ and the greater-than tests $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ for free: $\mathsf{c}_{\mathsf{lt}}[i,j] = \mathsf{Enc}(1) - \mathsf{c}_{\mathsf{gt}}[i,j] + \mathsf{c}_{\mathsf{eq}}[i,j]$. Now, replace all $\mathsf{c}_{\mathsf{gt}}[i,j]$ in the encrypted comparison matrix M_{rank} with $\mathsf{c}_{\mathsf{lt}}[i,j]$. Then, the same procedures as described for the n best positions above yield the desired variants for the n worst positions.

Threshold tests. For a given threshold τ, the trustees run the greater-than test $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ with input $(\vec{c}_{\mathsf{unsorted}}[i], \mathsf{Enc}(\tau))$ for all candidates i. For example, with τ being half of the number of votes, the trustees can check whether there is a candidate who wins the absolute majority of votes.

Example of a combination. Coming back to an example already mentioned in Section 1, consider an election that is carried out in two rounds. In the first round, there are several candidates. If one of them wins the absolute majority of votes, she is the winner. If not, there is a second round between the candidates on the first two positions. The winner of the second round wins the election. Using our instantiation, no unnecessary information needs to be leaked to anybody in any round of such an election.

In what follows, we denote (tally-hiding) results functions realized as described above by f_{Ordinos}.

Security of our instantiation of Ordinos. We have chosen the cryptographic primitives in our specific instantiation of Ordinos such that all assumptions for the verifiability result of the general Ordinos protocol (Theorem 1) are achieved. In particular, our instantiation inherits the verifiability level of Ordinos, as stated next.

Corollary 1 (Verifiability). *Let* $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B}) \wedge \bigwedge_{i=1}^{n_{\mathsf{voters}}} \mathsf{hon}(\mathsf{VVD}_i)$. *Then, the instantiation of* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, f_{\mathsf{Ordinos}})$ *presented above is* $(\gamma(k,\varphi), \delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}))$*-verifiable by the judge* J *where*

$$\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}) = \max\left(1 - p_{\mathsf{verify}}, 1 - p_{\mathsf{audit}}\right)^{\lceil \frac{k+1}{2} \rceil}.$$

Similarly, our instantiation inherits the privacy level of the general Ordinos protocol (Theorem 2).

Corollary 2 (Privacy). *The above instantiation of the protocol* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, f_{\mathsf{Ordinos}})$ *with* $n^{\mathsf{honest}}_{\mathsf{voters}}$ *honest voters achieves* $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)}(f_{\mathsf{res}})$ *privacy w.r.t. the mapping* $\mathcal{A}$ *to sets of admissible adversaries, with* $\mathcal{A}$ *and* f_{res} *as in Theorem 2.*

We elaborate on why all necessary assumptions for the above results are achieved in Appendix G.

Optimizations. We note that for some specific result functions, the performance of the tallying procedure of the instantiation of Ordinos described above can be improved. For example, if we want to realize a result function that reveals (at least) the full ranking without the number of votes in a tally-hiding way, then we can also use "classical" sorting algorithms (e.g., Quicksort or Insertion Sort). To realize such algorithms, in which intermediate comparison results determine the remaining sorting process, we run $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and immediately decrypt its output: while the decryption reveals information about the ranking of candidates, this information was not supposed to be kept secret. By this, average-case runtime can (asymptotically) be reduced from $\mathcal{O}(n^2_{\mathsf{cand}})$ to $\mathcal{O}(n_{\mathsf{cand}} \cdot \log n_{\mathsf{cand}})$, and in the best case the runtime might even be close to linear. Hence, as demonstrated in Section 8, performance can significantly be improved. Observe, however, that we cannot use such sorting algorithms if we want to reveal less than the complete ranking, e.g., only the winner.

8. Implementation

We provide a proof-of-concept implementation of Ordinos according to Section 7. The main purpose of our implementation was to be able to evaluate the performance of the system in the tallying phase (after the verified ciphertexts were homomorphically aggregated), which is the most time critical part. Our benchmarks therefore concentrate on the tallying phase. In particular, we generated the offline material for $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ in a trusted way (see Section 7 for alternatives).

Recall that the tallying phase consists of two parts. In the first part, the trustees generate $\vec{c}_{\mathsf{rank}}$ for input $\vec{c}_{\mathsf{unsorted}}$. In the second part, the trustees evaluate a specific result function with input $\vec{c}_{\mathsf{rank}}$ (and possibly $\vec{c}_{\mathsf{unsorted}}$). The first part accounts for the vast communication and computation complexity. We provide several benchmarks for running the first part depending on the number of voters, trustees, and candidates for the scenarios where the trustees (i) run on one machine, (ii) communicate over a local network, or (iii) over the Internet. We also demonstrate that the second part (evaluating a specific result function) is negligible in terms of runtime.

Our implementation is written in Python, extended with the gmpy2 module to accelerate modular arithmetic operations. The key length of the underlying Paillier encryption scheme is 2048 bits. We ran the protocol on standard Desktop computers and laptops.[15]

We first note that the length of encrypted integers to be compared by $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ determines the number of recursive calls of $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ from [32]. This protocol, in a nutshell,

15. ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM), Intel Pentium G4500 (64bit, 2x3.5 GHz Dualcore, 8 GB RAM), and Intel Core i7-6600U (CPU @ 2.60GHz, 2801 Mhz, 2 Cores 8 GB RAM).

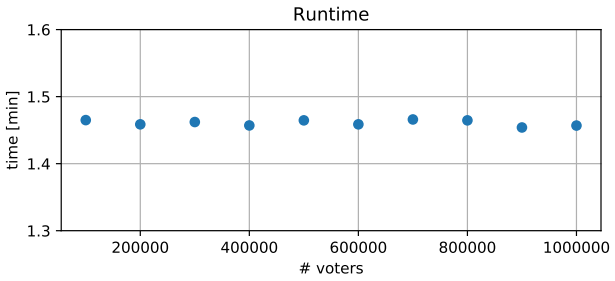

Figure 2: Three trustees on a local network and five candidates; 32-bit integers for vote counts.

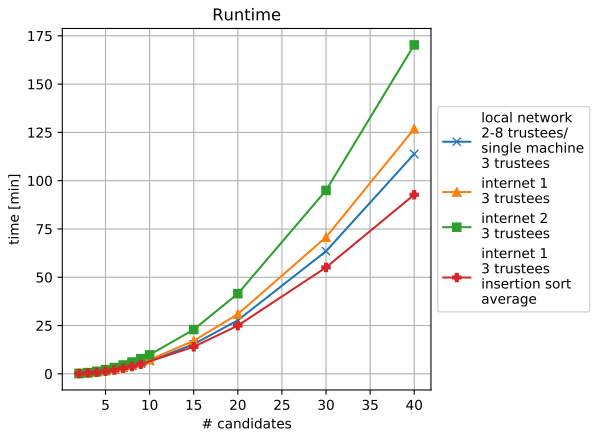

Figure 3: Trustees on a single machine, local network and on the Internet; 16-bit integers for vote counts.

splits the inputs in an upper and lower half and calls itself with one of those halves, depending on the output of the previous comparison. Hence, we use powers of 2 for the bit length of the integers. On a high level, this is also the reason for the logarithmic online complexity of $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$. For our implementation, we assume that each voter has one vote for each candidate. Therefore, we use 2^{16} bit integers for less than 2^{16} voters and 2^{32} bit integers for less than 2^{32} voters.

In summary, the benchmarks illustrate that our implementation is quite practical, even for an essentially unlimited number of voters and several trustees independently of whether the implementation runs over a local network or the Internet. The determining factor in terms of performance is the number of candidates. More specifically, in what follows we first present our benchmarks for the first part of the tallying phase (computing $\vec{c}_{\mathsf{rank}}$) and then briefly discuss the second part.

First phase: computing $\vec{c}_{\mathsf{rank}}$. Figure 2 demonstrates that the running time is independent of any specific number of voters (as long as it is smaller than the maximum number allowed, in this case less than $2^{32} - 1$ voters).

The blue graph (the second from below) in Figure 3 shows that the running time of our implementation is essentially independent of the number of trustees: The time difference for the different numbers of trustees (two to eight on a local network) are less than three seconds, and hence, not visible in this figure. This is due to the logarithmic online complexity of $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$.

Figure 3 also demonstrates that the parameter that determines the running time is the number of candidates, as $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ needs to be invoked $\mathcal{O}(n^2_{\mathsf{cand}})$ times to construct M_{rank} (see Section 7).

Furthermore, Figure 3 shows that the running time is quite independent of the specific networks over which the trustees communicate. In the local network with up to eight trustees, the running time is essentially the same as in the case of running three trustees on three different cores of the same machine (they differ by at most two seconds). In the setting *Internet 1*, we have used the same machines and connected them with a VPN running on a server in a different city so that the trustees effectively communicate over the Internet (via a VPN node in a different city). The setting *Internet 2* is more heterogeneous: we used different machines for the trustees, located in different cities (and partly countries), with two connected to the Internet via Wifi routers in home networks. They were all connected over the Internet to the same VPN as in *Internet 1*. Importantly, the difference between *Internet 1* and *Internet 2* is due to two factors: (i) The slowest machine dictates the overall performance since the other machines have to wait for the messages of this machine. While the fastest machine in our setup performs a greater-than test locally in about 8.5 seconds, the slowest one needs 10.5 seconds. (ii) The Internet connections from the home networks are slower than those in *Internet 1*.

Second phase: computing a specific result function. In order to obtain an upper bound for the runtime of the second phase, we benchmarked the most costly result function in the tally-hiding setting among the functions listed in Section 7, namely the one which reveals the set (without ranking) of the candidates on the first n positions. Note that the runtime of this function does not depend on n. For 40 candidates, we needed about 6.33 minutes for this function, with three trustees and 16-bit integers for vote counts, which is two orders of magnitude less than what is needed for the first part of the tallying phase. Hence, the runtime for the second phase is negligible. Since this part needs a linear number of greater-than operations in the number of candidates and the first part is quadratic, this was to be expected.

Optimizations. As described at the end of Section 7, if we want to reveal the full ranking (without the number of votes), we can improve the overall runtime. To illustrate this, we provide benchmarks of Insertion Sort in the setting *Internet 1* with three trustees (see red line in Figure 3).[16] As the runtime of Insertion Sort depends on the degree to which its input is already sorted, we simulated many different runs of Insertion Sort by distributing votes among the candidates uniformly at random. For example, as can be seen from Figure 3, compared to the general approach (in the same setting), Insertion Sort improves the runtime by more than 15% for elections with 30 candidates and 25% for elections with 40 candidates; clearly, the improvement increases with the number of candidates. Altogether, this demonstrates that for some specific result functions efficiency can be further improved compared to the general approach.

16. We note that we also applied Quicksort for these numbers of candidates ($\leq$ 40) where it was outperformed by Insertion Sort. For elections with many candidates (say $\geq$ 100), Quicksort would, however, be a reasonable alternative due to its asymptotically better average-case runtime.

9. Related Work

In this section, we compare Ordinos with the only four tally-hiding voting protocols [27]–[30] that have been proposed so far, and another voting protocol [53] that employs secure MPC for improving privacy and coercion-resistance, but without being fully tally-hiding. We also discuss general-purpose secure MPC for building a secure tally-hiding e-voting protocols.

Benaloh [28] introduced the idea of tally-hiding e-voting and designed the first protocol for tally-hiding more than thirty years ago. In contrast to modern e-voting systems, in which trust is distributed among a set of trustees, Benaloh's protocol assumes a *single* trusted authority which also learns how each single voter voted. Ordinos, in contrast, distributes trust among a set of trustees. As we have proven, none of the trustees gains more information about a voter's choice than what can be derived from the final published (tally-hiding) result. It seems infeasible to improve Benaloh's protocol in this respect. Additionally, the system lacks a security proof and also has not been implemented.

Hevia and Kiwi [29] designed a tally-hiding Helios-like e-voting system for the case of jury votings, i.e., where 12 voters can either vote yes or no (0 or 1). While this e-voting protocol seems to be a reasonable solution for this specific setting (very few voters, only yes/no votes), its computational complexity crucially depends on the number of voters, and it seems infeasible to generalize it to handle several candidates. Furthermore, Hevia and Kiwi have neither analyzed the security of their e-voting protocol nor implemented it.

Szepieniec and Preneel [27] proposed a tally-hiding voting protocol for which they develop a specific greater-than MPC protocol. Unfortunately, this MPC protocol is insecure, it leaks some information. The authors discuss some mitigations but do not solve the problem (see [27], Appendix A for details). Just as the previous two protocols, this protocol has not been implemented.

Canard et al. [30] have recently proposed a tally-hiding e-voting protocol for a different kind of election than considered here: in their system, the voters rank candidates and the winner of the election is calculated according to specific rules. The focus of their work was on designing and implementing the MPC aspects of the tallying phase. They do not design a complete e-voting protocol (including the voting phase, etc.). In particular, modeling a complete protocol (with e2e-verifiability) and analyzing its security was not in the scope of the paper. In Appendix H, we compare the performance of our implementation with theirs but we note again that Canard et al. tackle a different kind of elections, making a fair comparison hard.

Also very recently, Culnane et al. [53] proposed an instant-runoff voting (IRV) protocol in which the voters encrypt their personal ranking and the trustees run a secure MPC protocol in order to evaluate the winner without decrypting the single voters' encrypted rankings. The focus of this work was on mitigating so-called Italian attacks. We note that the protocol by Culnane et al. has *not* been designed to hide the tally completely: some information about the ranking of candidates always leaks.

In principle, alternatively to the approach taken here, one could also try to employ a generic secure MPC protocol (e.g., [54]) as a tally-hiding e-voting protocol. In order to be useful on the same level as Ordinos, such a protocol should satisfy the following properties in terms of MPC terminology: Firstly, the MPC protocol has to fit to the *client-server model* [55] in which the input parties (i.e., voters) are not required to actively participate in computing the result (i.e., tallying the ballots). Secondly, the correctness of the computation must be *publicly* verifiable [56] so that every external observer, and not only the computing parties, can check that the election outcome is valid. Thirdly, in order to provide accountability, the MPC protocol must provide *identifiable abort* [57], [58]. Fourthly, the previous properties should also be guaranteed even if all computing parties are malicious.[17] While there are solutions for each of these single properties (e.g., [55]–[58]), we are not aware of a (practical) MPC protocol that combines all of these properties. Developing such a protocol will be interesting future work.

10. Conclusion

With Ordinos, we proposed the first provably secure (remote) tally-hiding e-voting system. For the generic version of this protocol, we showed that it provides privacy and verifiability (even accountability). More specifically, we proved that the level of verifiability Ordinos provides does not depend on the result function the system realizes. We also obtained general results for the level of privacy an ideal voting protocol provides parameterized by the result function, and showed that Ordinos enjoys the same level of privacy as the ideal protocol. Besides proving the security of Ordinos, these results also give a deeper understanding of tally-hiding voting systems in general.

We demonstrated how the generic version of Ordinos can be instantiated for computing practically relevant classes of result functions in a tally-hiding way. We implemented Ordinos for these result functions and evaluated its performance, illustrating that our implementation is quite practical, for several trustees and independently of the number of voters.

It would be interesting to enhance the set of result functions that can be computed by (instantiations of) Ordinos in a tally-hiding way and to further improve the efficiency of Ordinos.

Acknowledgements. We thank the anonymous reviewers for their constructive feedback. We also thank Peter B. Rønne and Ben Smyth for helpful remarks. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) through grant KU 1434/11-1 and by the Luxembourg National Research Fund (FNR) through the INTER project SURCVS (Number 11747298).

References

[1] https://www.scytl.com/en/customers/ (accessed 06.08.2019).

[2] https://www.iacr.org/elections/eVoting/ (accessed 06.08.2019).

17. In particular, protocols which mainly aim at verifiable computation (e.g., [59]) where, unlike in our setting, each trustee learns how individual voters voted, do not achieve this requirement.

[3] https://www.debian.org/vote/ (accessed 06.08.2019).

[4] https://www.polyas.com/churches/church-council-elections/case-study (accessed 06.08.2019).

[5] R. Küsters, T. Truderung, M. Volkamer, B. Beckert, A. Brelle, R. Grimm, N. Huber, M. Kirsten, J. Müller-Quade, M. Noppel, K. Reinhard, J. Schwab, R. Schwerdt, and C. Winter, "GI Elections with POLYAS: a Road to End-to-End Verifiable Elections," in *Fourth International Joint Conference on Electronic Voting (E-Vote-ID 2019)*, 2019.

[6] https://www.forbes.com/sites/rebeccaheilweil1/2017/12/02/eight-companies-that-want-to-revolutionize-voting-technology/#29da2b3712c1 (accessed 06.08.2019).

[7] https://blogs.microsoft.com/on-the-issues/2019/09/24/electionguard-available-today-to-enable-secure-verifiable-voting/ (accessed 29.10.2019).

[8] J. Epstein, "Weakness in Depth: A Voting Machine's Demise," *IEEE Security & Privacy*, vol. 13, no. 3, pp. 55–58, 2015. [Online]. Available: http://dx.doi.org/10.1109/MSP.2015.46

[9] 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," 2010, pp. 1–14.

[10] D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security Analysis of the Estonian Internet Voting System," 2014, pp. 703–715.

[11] 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.

[12] R. Küsters, J. Müller, E. Scapin, and T. Truderung, "sElect: A Lightweight Verifiable Remote Voting System," in *CSF 2016*, 2016, pp. 341–354.

[13] X. Boyen, T. Haines, and J. Müller, "A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing," *IACR Cryptology ePrint Archive*, vol. 2020, p. 115, 2020. [Online]. Available: https://eprint.iacr.org/2020/115

[14] M. R. Clarkson, S. Chong, and A. C. Myers, "Civitas: Toward a Secure Voting System," in *S&P 2008*, 2008, pp. 354–368.

[15] B. Adida, "Helios: Web-based Open-Audit Voting," in *USENIX 2008*, 2008, pp. 335–348.

[16] 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 *EVT 2008*.

[17] 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.

[18] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène, "Election Verifiability for Helios under Weaker Trust Assumptions," in *ESORICS 2014*, 2014, pp. 327–344.

[19] 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.

[20] ——, "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

[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] C. Burton, C. Culnane, J. Heather, T. Peacock, P. Y. A. Ryan, S. Schneider, V. Teague, R. Wen, Z. Xia, and S. Srinivasan, "Using Prêt à Voter in Victoria State Elections," in *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, Bellevue, WA, USA, August 6-7, 2012*, J. A. Halderman and O. Pereira, Eds. USENIX Association, 2012. [Online]. Available: https://www.usenix.org/conference/evtwote12/workshop-program/presentation/burton

[23] C. Culnane, P. Y. A. Ryan, S. A. Schneider, and V. Teague, "vVote: A Verifiable Voting System," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 1, p. 3, 2015. [Online]. Available: http://doi.acm.org/10.1145/2746338

[24] D. Galindo, S. Guasch, and J. Puiggali, "2015 Neuchâtel's Cast-as-Intended Verification Mechanism," in *E-Voting and Identity - 5th International Conference, VoteID 2015, Bern, Switzerland, September 2-4, 2015, Proceedings*, ser. Lecture Notes in Computer Science, R. Haenni, R. E. Koenig, and D. Wikström, Eds., vol. 9269. Springer, 2015, pp. 3–18. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-22270-7_1

[25] K. Gjøsteen, "The Norwegian Internet Voting Protocol," in *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 7187. Springer, 2011, pp. 1–18.

[26] Der Bundesrat. Das Portal der Schweizer Regierung, "E-Voting – Wie wird die Sicherheit gehandhabt?" 2018, https://www.bk.admin.ch/bk/de/home/politische-rechte/e-voting/sicherheit-beim-e-voting.html (accessed 22.11.2018).

[27] A. Szepieniec and B. Preneel, "New techniques for electronic voting," in *USENIX Journal of Election Technology and Systems (JETS)*, 2015.

[28] J. D. Benaloh, "Improving Privacy in Cryptographic Elections (technical report)," Tech. Rep., 1986.

[29] A. Hevia and M. A. Kiwi, "Electronic Jury Voting Protocols," in *LATIN 2002: Theoretical Informatics, 5th Latin American Symposium, Cancun, Mexico, April 3-6, 2002, Proceedings*, 2002, pp. 415–429.

[30] S. Canard, D. Pointcheval, Q. Santos, and J. Traoré, "Practical Strategy-Resistant Privacy-Preserving Elections," in *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018*, 2018, pp. 331–349.

[31] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology - EUROCRYPT '99, Proceeding*, 1999, pp. 223–238.

[32] H. Lipmaa and T. Toft, "Secure Equality and Greater-Than Tests with Sublinear Online Complexity," in *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part II*, 2013, pp. 645–656.

[33] B. Schoenmakers and M. Veeningen, "Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems," in *Applied Cryptography and Network Security, ACNS 2015, Revised Selected Papers*, 2015, pp. 3–22.

[34] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, "Ordinos: A Verifiable Tally-Hiding Remote E-Voting System," *Cryptology ePrint Archive*, no. 2020/405, 2020, http://eprint.iacr.org/2020/405.

[35] 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.

[36] R. Küsters, T. Truderung, and A. Vogt, "Clash Attacks on the Verifiability of E-Voting Systems," in *S&P 2012*, 2012, pp. 395–409.

[37] V. Cortier and B. Smyth, "Attacking and Fixing Helios: An Analysis of Ballot Secrecy," in *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF, 2011*, 2011, pp. 297–311.

[38] F. Karayumak, M. M. Olembo, M. Kauer, and M. Volkamer, "Usability Analysis of Helios - An Open Source Verifiable Remote Electronic Voting System," in *EVT/WOTE '11*, 2011.

[39] 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.

[40] C. Culnane and S. A. Schneider, "A Peered Bulletin Board for Robust Use in Verifiable Voting Systems," in *IEEE 27th Computer Security Foundations Symposium, CSF, 2014*, 2014, pp. 169–183.

[41] A. Kiayias, A. Kuldmaa, H. Lipmaa, J. Siim, and T. Zacharias, "On the Security Properties of e-Voting Bulletin Boards," in *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Proceedings*, 2018, pp. 505–523.

[42] R. Küsters, T. Truderung, and A. Vogt, "Accountability: Definition and Relationship to Verifiability," in *CCS 2010*, 2010, pp. 526–535.

[43] V. Cortier and J. Lallemand, "Voting: You Can't Have Privacy without Individual Verifiability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, 2018, pp. 53–66.

[44] D. Bernhard, O. Pereira, and B. Warinschi, "How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 626–643.

[45] R. Küsters, T. Truderung, and A. Vogt, "Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study," in *S&P 2011*, 2011, pp. 538–553.

[46] D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi, "SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions," in *S&P 2015*, 2015, pp. 499–516.

[47] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*. IEEE Computer Society, 2001, pp. 136–145.

[48] R. Küsters, "Simulation-Based Security with Inexhaustible Interactive Turing Machines," in *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*. IEEE Computer Society, 2006, pp. 309–320, see http://eprint.iacr.org/2013/025/ for a full and revised version.

[49] T. E. Gamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in *Advances in Cryptology, Proceedings of CRYPTO '84*, 1984, pp. 10–18.

[50] 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.

[51] J. Algesheimer, J. Camenisch, and V. Shoup, "Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products," in *Advances in Cryptology - CRYPTO 2002, Proceedings*, 2002, pp. 417–432.

[52] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savas, "Depth Optimized Efficient Homomorphic Sorting," in *Progress in Cryptology - LATINCRYPT 2015, Proceedings*, 2015, pp. 61–80.

[53] K. Ramchen, C. Culnane, O. Pereira, and V. Teague, "Universally Verifiable MPC and IRV Ballot Counting," in *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, 2019, pp. 301–319.

[54] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Advances in Cryptology - CRYPTO 2012. Proceedings*, 2012, pp. 643–662.

[55] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft, "Confidential Benchmarking Based on Multiparty Computation," in *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Revised Selected Papers*, 2016, pp. 169–187.

[56] C. Baum, I. Damgård, and C. Orlandi, "Publicly Auditable Secure Multi-Party Computation," in *Security and Cryptography for Networks - 9th International Conference, SCN 2014. Proceedings*, 2014, pp. 175–196.

[57] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure Multi-Party Computation with Identifiable Abort," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Proceedings, Part II*, 2014, pp. 369–386.

[58] C. Baum, E. Orsini, and P. Scholl, "Efficient Secure Multiparty Computation with Identifiable Abort," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Proceedings, Part I*, 2016, pp. 461–490.

[59] E. Cuvelier and O. Pereira, "Verifiable Multi-party Computation with Perfectly Private Audit Trail," in *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016. Proceedings*, ser. Lecture Notes in Computer Science, vol. 9696. Springer, 2016, pp. 367–385.

[60] R. Küsters, M. Tuengerthal, and D. Rausch, "The IITM Model: a Simple and Expressive Model for Universal Composability," Cryptology ePrint Archive, Tech. Rep. 2013/025, 2013, available at http://eprint.iacr.org/2013/025. To appear in the Journal of Cryptology.

[61] B. Schoenmakers and P. Tuyls, "Efficient Binary Conversion for Paillier Encrypted Values," in *Advances in Cryptology - EUROCRYPT 2006, Proceedings*, 2006, pp. 522–537.

Appendix A. General Computational Model

In this section, we explain our computational model (Section 3) 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. Typically, a protocol P 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, e.g., i) register, ii) vote, iii) tally, iv) verify.

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 message corrupt. 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 corrupt messages. Also, agents might only accept corrupt messages upon initialization, modeling static corruption. In our security analysis of Ordinos, we assume static corruption.

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 corrupt 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 corrupt messages.

Property. A *property* γ *of* P is a subset of the set of all runs of P.[18] By $\neg\gamma$ we denote the complement of γ.

Appendix B. Formal Definition of Goal $\gamma(k, \varphi)$

In this section, we formally define the goal $\gamma(k, \varphi)$ which we have described on a high level in Section 4.1.

Goal $\gamma(k, \varphi)$**.** In order to define the number of manipulated votes, we consider a specific distance function d. In order to define d, we first define a function $f_\mathsf{count} \colon \mathsf{C}^* \to \mathbb{N}^\mathsf{C}$ which, for a vector $(\mathsf{ch}_1, \ldots, \mathsf{ch}_l) \in \mathsf{C}^*$ (representing a multiset of voters' choices), counts how many times each choice occurs in this vector. For example, $f_\mathsf{count}(B, C, C)$ assigns 1 to B, 2 to C, and 0 to all the remaining choices. Now, for two vectors of choices $\vec{\mathsf{c}}_0, \vec{\mathsf{c}}_1$, the distance function d is defined by

$$d(\vec{\mathsf{c}}_0, \vec{\mathsf{c}}_1) = \sum_{\mathsf{ch} \in \mathsf{C}} \left| f_\mathsf{count}(\vec{\mathsf{c}}_0)[\mathsf{ch}] - f_\mathsf{count}(\vec{\mathsf{c}}_1)[\mathsf{ch}] \right| .$$

For example, $d((B, C, C), (A, C, C, C)) = 3$.

Now, let $f_\mathsf{res} \colon \mathsf{C}^* \to \{0,1\}^*$ be a result function, and, for a given protocol run r, let $(\mathsf{ch}_i)_{i \in I_\mathsf{honest}}$ be the vector of choices made by the honest voters I_honest in r.[19] Then, the goal $\gamma(k, \varphi)$ is satisfied in r (i.e., r belongs to $\gamma(k, \varphi)$) if either (a) the trust assumption φ does not hold true in r, or (b) φ holds true in r and there exist valid choices $(\mathsf{ch}'_i)_{i \in I_\mathsf{dishonest}}$ (representing possible choices of the dishonest voters $I_\mathsf{dishonest}$ in r) and choices $\vec{\mathsf{c}}_\mathsf{real} = (\mathsf{ch}_i^\mathsf{real})_{i \le n_\mathsf{voters}}$ such that:

(i) an election result is published in r and this result is equal to $f_\mathsf{res}(\vec{\mathsf{c}}_\mathsf{real})$, and
(ii) $d(\vec{\mathsf{c}}_\mathsf{ideal}, \vec{\mathsf{c}}_\mathsf{real}) \le k$,

where $\vec{\mathsf{c}}_\mathsf{ideal}$ consists of the actual choices $(\mathsf{ch}_i)_{i \in I_\mathsf{honest}}$ made by the honest voters (recall the notion of actual choices from Section 3) and the possible choices $(\mathsf{ch}'_i)_{i \in I_\mathsf{dishonest}}$ made by the dishonest voters.

We note that for Ordinos we consider tallying functions f_tally which work on aggregated votes, i.e., vectors encoding for each choice/candidate the number of votes for this choice. That is, we consider result functions f_res of the form $f_\mathsf{res}(\vec{c}) = f_\mathsf{tally}(f_\mathsf{count}(\vec{c}))$.

18. 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.

19. Recall that the set of honest/dishonest parties is determined at the beginning of each protocol run.

Appendix C. Non-Interactive Zero-Knowledge Proofs

We refer to our technical report [34] for the general definition of (NI)ZKPs. We now describe the NIZKPs used in the Ordinos.

Let $(\mathsf{KeyShareGen}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$ be a (threshold) public-key encryption scheme. Then, the zero-knowledge proofs used in the voting protocol are formally defined as follows:

- *NIZKP* $\pi^{\mathsf{KeyShareGen}}$ *of knowledge and correctness of the private key share.* For a given public key pk_i, the statement is:

$$\exists \mathsf{sk}_i \ \exists r \colon (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyShareGen}(r)$$

- *NIZKP* π^{Enc} *of knowledge and correctness of plaintext(s).* Let R_m be an n-ary relation over the plaintext space. For $(\mathsf{c}_1, \ldots, \mathsf{c}_n, \mathsf{pk})$, the statement is:

$$\exists (\mathsf{m}_1, \ldots, \mathsf{m}_n) \in R_\mathsf{m} \ \forall i \ \exists \mathsf{r}_i \colon \mathsf{c}_i = \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_i; \mathsf{r}_i).$$

Appendix D. Judging Procedure

In this section, we describe the judging procedure of Ordinos.

Judge J**.** We assume that J is honest. We note that the honest program $\hat{\pi}_\mathsf{J}$ of J, as defined below, uses only publicly available information, and therefore every party, including the voters as well as external observers, can run the judging procedure.

The program $\hat{\pi}_\mathsf{J}$, whenever triggered by the scheduler S, reads data from the bulletin board and verifies its correctness, including correctness of posted complaints. The judge outputs verdicts (as described below) on a distinct tape. More precisely, the judge outputs verdict in the following situations:

(J1) If a party a deviates from the protocol specification in an obvious way, then J blames a individually by outputting the verdict $\mathsf{dis}(\mathsf{a})$. This is the case if the party a, for example, (i) does not publish data when expected, or (ii) publishes data which is not in the expected format, or (iii) publishes a NIZKP which is not correct, etc.

(J2) If a voter V_i posts an authenticated complaint in the voting phase that she has not received a valid acknowledgement from the authentication server AS, then the judge outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$, which means that (the judge believes that) one of the parties V_i, VSD_i, AS is dishonest but cannot determine which of them.

(J3) If a voter V_i posts an authenticated complaint claiming that she did not vote, but her name was posted by the authentication server AS in one of the ballots in $\vec{\mathsf{b}}$, the judge outputs the verdict $\mathsf{dis}(\mathsf{AS}) \vee \mathsf{dis}(\mathsf{V}_i)$.

(J4) If, in the verification phase, a valid complaint is posted containing an acknowledgement of AS, i.e., the complaint contains a signature of AS on a ballot which is not in the list of ballots $\vec{\mathsf{b}}$ published

by AS, then the judge blames AS individually by outputting the verdict $\mathsf{dis}(\mathsf{AS})$.

(J5) During the execution of $\mathsf{P}_{\mathsf{MPC}}$ the judge runs the judging procedure $\mathsf{J}_{\mathsf{MPC}}$ of $\mathsf{P}_{\mathsf{MPC}}$. If $\mathsf{J}_{\mathsf{MPC}}$ outputs a verdict, then J also outputs this verdict.

(J6) If, in the submission phase, a voter V_i posts an authenticated complaint claiming that her VSD_i did not produce a correct ballot b_i for her chosen input, then the judge outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{VSD}_i)$, which means that (the judge believes that) either V_i or VSD_i is dishonest but cannot determine which of them.

If none of these situations occur, the judge J outputs accept on a distinct tape.

Appendix E. Accountability

In this section, we first recall the accountability framework and definition that has been introduced in [42]. Then we apply this definition to analyze accountability of Ordinos.

E.1. Accountability Framework

To specify accountability in a fine-grained way, the notions of verdicts, constraints and accountability properties are used.

Verdicts. A verdict can be output by the judge (on a dedicated output channel) and states which parties are to be blamed (that is, which ones, according to the judge, have misbehaved). In the simplest case, a verdict can state that a specific party misbehaved (behaved dishonestly). Such an *atomic verdict* is denoted by $\mathsf{dis}(\mathsf{a})$ (or $\neg\mathsf{hon}(\mathsf{a})$). It is also useful to state more fine grained or weaker verdicts, such as "a or b misbehaved". Therefore, in the general case, we will consider verdicts which are boolean combinations of atomic verdicts.

More formally, given a run r of a protocol P (i.e., a run of some instance $\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}}$ of P), we say that a verdict ψ *is true in* r, if and only if the formula ψ evaluates to true with the proposition $\mathsf{dis}(\mathsf{a})$ set to false if party a is honest in r, i.e., party a runs $\hat{\pi}_{\mathsf{a}}$ in r and has not been (statically) corrupted in r. For the following, recall that the instance $\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}}$ is part of the description of r. By this, we can talk about sets of runs of different instances.

In fact, in our formal analysis of Ordinos, we use in some cases verdicts of the form $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$ stating that either the i-th voter V_i or the authentication server AS misbehaved (but the verdict leaves open, as it might not be clear, which one of them).

Accountability constraints. Who should be blamed in which situation is expressed by a set of *accountability constraints*. Intuitively, for each undesired situation, e.g., when the goal $\gamma(k, \varphi)$ is not met in a run of $\mathsf{P}_{\mathsf{Ordinos}}$, we would like to describe who to blame.

More formally, an accountability constraint is a tuple $(\alpha, \psi_1, \ldots, \psi_k)$, written $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$, where α is a property of P (recall that, formally, this is a set of runs of P) and $\psi_1, \ldots, \psi_k$ are verdicts. Such a constraint *covers* a run r if $r \in \alpha$. Intuitively, in a constraint $\Gamma = (\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$ the set α contains runs in which some desired goal of the protocol is *not* met (due to the misbehavior of some protocol participant). The formulas $\psi_1, \ldots, \psi_k$ are the possible (minimal) verdicts that are supposed to be stated by J in such a case; J is free to state stronger verdicts. Formally, for a run r, J *ensures* Γ *in* r, if either $r \notin \alpha$ or J states a verdict ψ in r that implies one of the verdicts $\psi_1, \ldots, \psi_k$ (in the sense of propositional logic).

Accountability property. A set Φ of accountability constraints for a protocol P is called an *accountability property* of P. An accountability property Φ should be defined in such a way that it covers all relevant cases in which a desired goal is not met, i.e., whenever some desired goal of P is not satisfied in a given run r due to some misbehavior of some protocol participant, then there exists a constraint in Φ which covers r. In particular, in this case the judge is required to state a verdict.

Notation. Let P be a protocol with the set of agents Σ and an accountability property Φ of P. Let π be an instance of P and $\mathsf{J} \in \Sigma$ be an agent of P. We write $\Pr[\pi^{(\ell)} \mapsto \neg(\mathsf{J}\colon \Phi)]$ to denote the probability that π, with security parameter 1^ℓ, produces a run such that J does not ensure Γ in this run for some $\Gamma \in \Phi$.

Definition 3 (Accountability)**.** *Let* P *be a protocol with the set of agents* Σ*. Let* $\delta \in [0, 1]$ *be the tolerance,* $\mathsf{J} \in \Sigma$ *be the judge, and* Φ *be an accountability property of* P*. Then,* the protocol P is (Φ, δ)-accountable w.r.t. the judge J *if for all adversaries* π_{A} *and* $\pi = (\hat{\pi}_{\mathsf{P}} \| \pi_{\mathsf{A}})$*, the probability* $\Pr[\pi^{(\ell)} \mapsto \neg(\mathsf{J}\colon \Phi)]$ *is* δ*-bounded as a function of* ℓ*.*

Similarly to the verifiability definition, we also require that the judge J is *computationally fair* in P, i.e., for all instances π of P, the judge J states false verdicts only with negligible probability. For brevity of presentation, this is omitted here (see [42] for details). This condition is typically easy to check. In particular, it is easy to check that the judging procedure for Ordinos does not blame honest parties.

Individual accountability. In practice, so-called *individual accountability* is highly desirable in order to deter parties from misbehaving. Formally, $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_k)$ provides *individual accountability* if for every $i \in \{1, \ldots, k\}$ there exists a party a such that ψ_i implies $\mathsf{dis}(\mathsf{a})$. In other words, each $\psi_1, \ldots, \psi_k$ determines at least one misbehaving party.

E.2. Accountability of Ordinos

We are now able to precisely analyze the accountability level provided by Ordinos. For this, we first define the accountability constraints and property of Ordinos. Then, we state and prove the accountability theorem.

Accountability constraints. In the case of Ordinos, we have the following accountability constraints.

Let χ_i denote the set of runs of an instance of $\mathsf{P}_{\mathsf{Ordinos}}$ where voter V_i complains that she did not get a receipt from AS via VSD_i. In such runs, the judge cannot be sure who to blame individually: (i) AS who might not have replied as expected (but claims, for instance, that the ballot was not cast), or (ii) VSD_i who might not have submitted

a ballot or forwarded the (correct) acknowledgement to VVD_i, or (iii) the voter who might not have cast a ballot but nevertheless claims that she has.[20] This is captured by the accountability constraint $\chi_i \Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$. Recall that we say that the judge J ensures this constraint in a run r, if $r \notin \chi_i$ or the verdict output by the J in r implies $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$ in the sense of propositional logic.

Let χ_i' contain all runs of $\mathsf{P}_{\mathsf{Ordinos}}$ where the voter V_i complains that she did not vote but her name is contained in a ballot in $\vec{\mathsf{b}}$ published by AS. Then, the accountability constraint for this situation is $\chi_i' \Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

Let χ_i'' contain all runs of $\mathsf{P}_{\mathsf{Ordinos}}$ where the voter V_i complains that the ballot auditing was not successful. Then, the accountability constraint for this situation is $\chi_i'' \Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{VSD}_i)$.

The accountability theorem for Ordinos (see below) states that if the adversary breaks the goal $\gamma(k,\varphi)$ in a run of $\mathsf{P}_{\mathsf{Ordinos}}$ but neither χ_i, χ_i' nor χ_i'' occur (for some voter V_i), then (at least) one misbehaving party can be blamed individually (with a certain probability). The accountability constraint for this situation is

$$\neg\gamma(k,\varphi) \wedge \neg\chi \Rightarrow \mathsf{dis}(\mathsf{AS})|\mathsf{dis}(\mathsf{T}_1)|\ldots|\mathsf{dis}(\mathsf{T}_{n_{\mathsf{trustees}}}),$$

where $\chi = \bigcup_{i\in\{1,\ldots,n_{\mathsf{voters}}\}}(\chi_i \cup \chi_i' \cup \chi_i'')$. Now, the judge J ensures this constraint in a run r if $r \notin \neg\gamma(k,\varphi) \wedge \neg\chi$ or the verdict output by J in r implies $\mathsf{dis}(\mathsf{a})$ for some party a mentioned in the constraint.

Accountability property. For $\mathsf{P}_{\mathsf{Ordinos}}$ and the goal $\gamma(k,\varphi)$, we define the accountability property Φ_k to consist of the constraints mentioned above for the cases χ_i, χ_i', χ_i'' (for all $i \in \{1,\ldots,n_{\mathsf{voters}}\}$), and $\neg\gamma(k,\varphi) \wedge \neg\chi$. Clearly, this accountability property covers $\neg\gamma(k,\varphi)$ by construction, i.e., if $\gamma(k,\varphi)$ is not satisfied, these constraints require the judge J to blame some party. Note that in the runs covered by the last constraint of Φ_k all verdicts are atomic. This means that Φ_k requires that except for the cases where χ occurs, whenever the goal $\gamma(k,\varphi)$ is violated, an individual party is blamed, so-called *individual accountability*. This is due to the NIZKPs and signatures used in Ordinos.

For the accountability theorem, we make the same assumptions (V1) to (V3) as for the verifiability theorem (see Section 4), with the following refinement. Since, in general, verifiability does not imply accountability, we need to assume the MPC protocol not only provides verifiability but also accountability. Hence, we refine the verifiability assumption (V3) (Section 4) as follows so that Ordinos guarantees accountability.

(V3) The MPC protocol $\mathsf{P}_{\mathsf{MPC}}$ enjoys *individual accountability* (w.r.t. the goal $\gamma(0,\varphi)$ and accountability level 0), meaning that if the outcome of the protocol does not correspond to f_{tally}, then at least one of the trustees can always be blamed individually, because in this case the NIZKP π^{MPC} mentioned in Section 2 fails. (Our instantiation presented in Section 7 fulfills this assumption.)

Now, the following theorem states the accountability result of Ordinos.

20. Note that this is a very general problem which applies to virtually any remote voting protocol. In practice, the voter could ask the voting authority Auth to resolve the problem.

Theorem 4 (Accountability)**.** *Under the assumptions (V1) to (V3) stated above and the mentioned judging procedure run by the judge* J*,* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, f_{\mathsf{tally}})$ *is* $(\Phi_k, \delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}))$*-accountable w.r.t. the judge* J *where*

$$\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}) = \max\left(1-p_{\mathsf{verify}}, 1-p_{\mathsf{audit}}\right)^{\lceil\frac{k+1}{2}\rceil}.$$

The full proof is provided in our technical report [34]. We note that it would be a bit more accurate to split up p_{verify} into two probabilities because it is more likely that a voter who voted checks whether her ballot appears on the bulletin board than that a voter who did not vote checks whether her ID does not appear. This has, for example, been taken into account in the security analysis of the Helios protocol [36]. We could do the same for Ordinos, but for simplicity and in order to concentrate more on the tally-hiding aspects, we do not distinguish these two cases here. Also, checks by voters who abstained are mainly about preventing ballot stuffing, which can be dealt with by other means as well (see also Footnote 6). Similarly for p_{audit}, the result could be generalized, as mentioned in Footnote 9.

Appendix F. Secure Multiparty Computation

An MPC protocol is run among a set of trustees $\mathsf{T}_1,\ldots,\mathsf{T}_{n_{\mathsf{trustees}}}$ in order to evaluate a given function f_{MPC} over secret inputs. Some of these trustees may be corrupted by the adversary A. We are interested in the case that the adversary is allowed to let the corrupted parties actively deviate from their honest protocol specification, i.e., that corrupted trustees can run arbitrary ppt programs. Such adversaries are called *malicious* (in contrast to the weaker notion of *honest-but-curious* or *passive* adversaries). We assume that, before a protocol run starts, the set of corrupted parties is already determined and does not change throughout the run. Such adversaries are called *static* (in contrast to the stronger notion of *dynamic* adversaries).

In this section, we specify the security properties for the protocols that can be used in Ordinos. We can model each MPC protocol in the formal protocol model presented in Section 3. More precisely, each protocol $\mathsf{P}_{\mathsf{MPC}}$ is run among the set of trustees, a scheduler $\mathsf{S}_{\mathsf{MPC}}$, a bulletin board $\mathsf{B}_{\mathsf{MPC}}$ and a judge $\mathsf{J}_{\mathsf{MPC}}$. The roles of the latter parties are the same as for the voting protocol, in particular, they are all assumed honest (recall Section 3).

Typically, $\mathsf{P}_{\mathsf{MPC}}$ is split into a *setup* or *offline* protocol in which the trustees generate key material, special randomness, etc., and a *computing* or *online* protocol in which the trustees secretly evaluate f_{MPC} over some secret inputs. In what follows, we are only interested in the online protocol and assume that the offline protocol has been executed honestly.

On a high level, the input to the (online) protocol consists of a vector of plaintexts $(\mathsf{m}_1,\ldots,\mathsf{m}_m)$, each of which is encrypted under the public key pk of a $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E}$. Each trustee T_k holds a secret key share sk_k relating to the public key pk. If at least t trustees are honest, the (correct) output of

$\mathcal{I}_{\mathsf{MPC}}(\mathcal{E}, f_{\mathsf{MPC}})$

Parameters:

- A $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$
- Function $f_{\mathsf{MPC}} \colon \{0,1\}^* \to \{0,1\}^*$
- Number of honest trustees $n^{\mathsf{honest}}_{\mathsf{trustees}}$
- $K \leftarrow \emptyset$ (initially)

On $(\mathsf{getKeyShare}, k)$ from S do:

1) If $k \notin \{1, \ldots, n^{\mathsf{honest}}_{\mathsf{trustees}}\}$, return $\perp$.
2) $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyShareGen}$
3) $K \leftarrow K \cup \{k\}$
4) Store sk_k and return pk_k to S

On $(\mathsf{setKeyShare}, k, \mathsf{sk})$ from S do:

1) If $k \notin \{n^{\mathsf{honest}}_{\mathsf{trustees}} + 1, \ldots, n_{\mathsf{trustees}}\}$, return $\perp$.
2) Store $\mathsf{sk}_k \leftarrow \mathsf{sk}$
3) $K \leftarrow K \cup \{k\}$
4) Return success

On $(\mathsf{compute}, b, \mathsf{c}_1, \ldots, \mathsf{c}_m)$ from S do:

1) If $b = 0$, return $\perp$.
2) $\forall i \in \{1, \ldots, m\}$:
   a) $\forall k \in K \colon \mathsf{dec}_{i,k} \leftarrow \mathsf{DecShare}(\mathsf{c}_i, \mathsf{sk}_k)$
   b) $\mathsf{m}_i \leftarrow \mathsf{Dec}(\mathsf{dec}_{i,1}, \ldots, \mathsf{dec}_{i,n_{\mathsf{trustees}}})$
   c) If $\mathsf{m}_i = \perp$, return $\perp$.
3) Return $\mathsf{res} \leftarrow f_{\mathsf{MPC}}(\mathsf{m}_1, \ldots, \mathsf{m}_m)$ to S.

Figure 4: Ideal MPC protocol.

is $f_{\mathsf{MPC}}(\mathsf{m}_1, \ldots, \mathsf{m}_m)$, where f_{MPC} is the given function to be secretly evaluated.

In what follows, we precisely define the security properties, privacy and individual accountability, that $\mathsf{P}_{\mathsf{MPC}}$ is supposed to guarantee so that Ordinos provides privacy and accountability (Theorem 4 and 2).

F.1. Privacy

On a high level, an MPC protocol provides privacy if the adversary only learns the outcome of the MPC protocol but nothing else if he corrupts less than t trustees. We formally define this idea with an ideal MPC protocol as follows. We say that $\mathsf{P}_{\mathsf{MPC}}$ provides ideal privacy if it realizes the ideal MPC functionality $\mathcal{I}_{\mathsf{MPC}} = \mathcal{I}_{\mathsf{MPC}}(\mathcal{E}, f_{\mathsf{MPC}})$ (Fig. 4), in the usual sense of universal composability [47], [48], i.e., there exists an adversarial program S (the simulator) such that for all programs E (the environment), it holds that $\mathsf{E}|\mathsf{P}_{\mathsf{MPC}}$ and $\mathsf{E}|\mathsf{S}|\mathcal{I}_{\mathsf{MPC}}$ are indistinguishable.[21]

F.2. Individual Accountability

We require that if the real outcome of $\mathsf{P}_{\mathsf{MPC}}$ does not correspond to its input, then $\mathsf{P}_{\mathsf{MPC}}$ provides evidence to individually blame (at least) one misbehaving trustee T_k. More precisely, we require that the protocol $\mathsf{P}_{\mathsf{MPC}}$ provides individual accountability for the goal $\gamma_{\mathsf{MPC}}(\varphi)$, where the trust assumption is

$$\varphi = \mathsf{hon}(\mathsf{S}_{\mathsf{MPC}}) \wedge \mathsf{hon}(\mathsf{B}_{\mathsf{MPC}}) \wedge \mathsf{hon}(\mathsf{J}_{\mathsf{MPC}}),$$

and goal $\gamma_{\mathsf{MPC}}(\varphi)$ is the goal $\gamma(0, \varphi)$ w.r.t. the input plaintexts $(\mathsf{m}_1, \ldots, \mathsf{m}_m)$ to the MPC protocol (recall Section 4 for details). Formally, the accountability property Φ of $\mathsf{P}_{\mathsf{MPC}}$ consists of the constraint

$$\neg\gamma_{\mathsf{MPC}}(\varphi) \Rightarrow \mathsf{dis}(\mathsf{T}_1) | \ldots | \mathsf{dis}(\mathsf{T}_{n_{\mathsf{trustees}}}),$$

and accountability level 0. In other words, if the adversary tries to change the outcome, at least one of the corrupted trustees will be identified with overwhelming probability.

Appendix G. Instantiation

In this section, we further elaborate on the verifiability and privacy of our instantiation of Ordinos (Section 7).

Accountability of our Instantiation of Ordinos. In order to describe why our instantiation of Ordinos inherits the verifiability level of the general Ordinos protocol (Corollary 1), we show that our instantiation even inherits its accountability level (Theorem 4). Since accountability implies verifiability [42], the correctness of Corollary 1 will follow.

Our instantiations of $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ provide individual accountability, i.e., everyone can tell whether a trustee misbehaved, mainly due to the NIZKPs employed. More precisely, in $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$, the trustees only exchange shared decryptions (of some intermediate ciphertexts) each of which is equipped with a NIZKP of correct decryption. Hence, the output of the MPC protocols can only be false if one of the shared decryptions is false, and in this case, the responsible trustee can be identified. This implies that our protocol $\mathsf{P}_{\mathsf{MPC}}$ provides individual accountability w.r.t. the goal $\gamma(0, \varphi)$ and accountability tolerance 0 up to the point where $\vec{\mathsf{c}}_{\mathsf{rank}}$ is computed (with $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$ as before). In the second phase of $\mathsf{P}_{\mathsf{MPC}}$, again $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ are used as well as distributed *verifiable* decryption (which anyway is part of $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$). This phase therefore also provides individual accountability w.r.t. the goal $\gamma(0, \varphi)$ and accountability tolerance 0. Altogether, we obtain the following theorem.

Theorem 5 (Accountability)**.** *Let $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$. Then, the protocol $\mathsf{P}_{\mathsf{MPC}}$, as defined in Section 7, provides individual accountability for the goal $\gamma(0, \varphi)$ and accountability level 0.*

With this, assumption (V3) for Theorem 4 is satisfied. Since the distributed Paillier public-key encryption scheme is correct, the signature scheme $\mathcal{S}$ is EUF-CMA-secure, and the proof π^{Enc} is a NIZKP, also assumption (V1) is satisfied. With the judge J defined analogously to the one of the generic Ordinos system, we can therefore conclude that our instantiation enjoys the same level of accountability level as the generic Ordinos system.

Corollary 3 (Accountability)**.** *The instantiation of $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{Ordinos}})$ as described*

21. Here we use the security notion of strong simulatability, which has been shown in [60] to be equivalent to the security notion of universal composability, which involves a real adversary instead of just the simulator.

above provides $(\Phi_k, \delta_k(p_{\mathsf{verify}}))$*-accountability w.r.t. the judge* J *where we have* $\delta_k(p_{\mathsf{verify}}) = (1-p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$.

From this result, Corollary 1 follows.

Privacy of our Instantiation of Ordinos. Lipmaa and Toft [32] showed that $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ are secure MPC protocols in a completely malicious setting under the assumption that the underlying ABB is realized correctly. In our instantiation, the ABB is correctly realized by the (standard) NIZKPs from [33] and under the assumption that at least the threshold of t trustees are honest.

Now, in the first part of the tallying phase (with input $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ and output $\vec{\mathsf{c}}_{\mathsf{rank}}$), merely $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ are applied to $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ so that we can replace them with their respective ideal MPC protocols and simulate them accordingly towards the adversary. The output $\vec{\mathsf{c}}_{\mathsf{rank}}$ encrypts the overall position of each candidate in the final ranking. Hence, we can conclude that the first part provides the same privacy level as an ideal MPC protocol that takes as input the (encrypted) number of votes per candidate and outputs the (encrypted) overall candidate ranking and be simulated towards the adversary accordingly.

In the second part of the tallying phase (with input $\vec{\mathsf{c}}_{\mathsf{rank}}$), the specific result function is evaluated in a tally-hiding fashion. Again, depending on the specific function, we use $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ or $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ and also verifiable decryption in the form of a standard NIZKP π^{Dec}. Now, for all result functions that are defined in Section 7, it is easy to see that the outcome of the second part reveals exactly what is required by the respective result function. Taking this together with the fact that $\mathsf{P}^{\mathsf{eq}}_{\mathsf{MPC}}$ and $\mathsf{P}^{\mathsf{gt}}_{\mathsf{MPC}}$ provide ideal privacy, and that π^{Dec} can be simulated, we can conclude that the second part also provides ideal privacy for each such result function.

Finally, we can conclude that for all instantiations defined in Section 7, the complete tallying phase provides ideal privacy.

Appendix H. Comparison to Canard et al.

	3 candidates		5 candidates	
# voters	[30]	Ordinos	[30]	Ordinos
$2^{10}-1$	4.26	0.26 (16 bit)	9.53	1.27 (16 bit)
$2^{10}-1$	4.26	0.30 (32 bit)	9.53	1.35 (32 bit)
$2^{20}-1$	8.53	0.30 (32 bit)	19.05	1.36 (32 bit)

TABLE 1: Comparison to [30] (three trustees, time in minutes).

In Table 1, we briefly compare the performance of our implementation with theirs, using the only available benchmarks published in [30], where the tallying is done on a single machine, i.e., all trustees run on a "single computer with physical CPU cores (i5-4300U)". For the purpose of this comparison, we run our implementation also only on a single machine, using the same key size as Canard et al., namely 2048 bits. However, we note again that Canard et al. tackle a different kind of elections, making a fair comparison hard. Having said this, as can been seen from Table 1, our implementation is 5 to 13 times faster than the one by Canard et al. Note that the runtime difference of Ordinos for different numbers of voters is due to different bit lengths of integers. For 2^{10} voters we use 16-bit integers and for 2^{20} we use 32-bit integers. Since the round complexity of the MPC protocol [61] that is used by Canard et al. is much higher than the one of the MPC protocol that we implemented, we conjecture that the differences would further increase when the trustees in [30] would actually be connected over a network. As demonstrated in Section 8, in our case a network does in fact not cause much overhead.

Appendix I. Ideal Privacy

We now describe the formula $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$ for which Theorem 3 states that this level is indeed ideal. More precisely, we will show in our technical report [34] that an ideal voting protocol achieves $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$-privacy and this privacy level is ideal, namely there exists no $\delta < \delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$ such that the ideal protocol achieves δ-privacy.

Recall that privacy is defined w.r.t. an honest voter, called the voter under observation, for which the adversary has to decide whether this voter voted for ch or ch', for any choices ch_0 and ch_1.

Let $A^{i,\vec{R}}_{\mathsf{res}}$ denote the probability that the choices made by the honest voters yield the output res of the result function f_{res} (e.g., only the winner of the election or some ranking of the candidates), given that the voter under observation picks choice $i \in \mathsf{C}$ and the dishonest voters vote according the choice vector $\vec{R} = (\mathsf{ch}_i)_{i \in I_{\mathsf{dishonest}}}$. (Clearly, $A^{i,\vec{R}}_{\mathsf{res}}$ depends on μ. However, we omit this in the notation.) Furthermore, let $A^{i}_{\vec{r}}$ denote the probability that the choices made by the honest voters yield the choice vector $\vec{r} = (\mathsf{ch}_i)_{i \in I_{\mathsf{honest}}}$ given that the voter under observation chooses choice i. (Again, $A^{i}_{\vec{r}}$ depends on μ, which we omit in the notation.) In what follows, we write $\vec{r}+\vec{R}$ to denote a vector of integers indicating the number of votes each choice in C got according to $\vec{r}$ and $\vec{R}$.

It is easy to see that

$$A^{i,\vec{R}}_{\mathsf{res}} = \sum_{\vec{r}\colon\, f_{\mathsf{res}}(\vec{r}+\vec{R})=\mathsf{res}} A^{i}_{\vec{r}}$$

and

$$A^{i}_{\vec{r}} \;=\; \frac{n!}{r_1! \cdots r_{n_{\mathsf{option}}}!} \cdot p_1^{r_1} \ldots p_k^{r_{n_{\mathsf{option}}}} \cdot \frac{r_i}{p_i}$$

where $(p_1, \ldots p_{n_{\mathsf{option}}})$ is the probability distribution of the honest voters on the possible choices C defined by μ, where now we simply enumerate all choices and set $\mathsf{C} = \{1, \ldots, n_{\mathsf{option}}\}$.

Moreover, let $M^{*}_{j,j',\vec{R}} = \{\mathsf{res}\colon A^{j,\vec{R}}_{\mathsf{res}} \leq A^{j',\vec{R}}_{\mathsf{res}}\}$. Now, the intuition behind the definition of $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$ is as follows: If the observer, given an output res, wants to decide whether the observed voter voted for choice j or j', the best strategy of the observer is to opt for j' if $\mathsf{res} \in M^{*}_{j,j',\vec{R}}$, i.e., the output is more likely if the voter voted for candidate j'. This leads to the following definition:

$$\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}}) \tag{1}$$

$$= \max_{j,j' \in \{1,\ldots,n_{\mathsf{option}}\}} \max_{\vec{R}} \sum_{\mathsf{res} \in M^{*}_{j,j',\vec{R}}} (A^{j',\vec{R}}_{\mathsf{res}} - A^{j,\vec{R}}_{\mathsf{res}}) \tag{2}$$