Design and Cryptographic Security Analysis of E-Voting Protocols

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Johannes Müller

aus Köln

Hauptberichter:	Prof. Dr. Ralf Küsters
Mitberichter:	Prof. Dr. Peter Y. A. Ryan

Tag der mündlichen Prüfung: 27.05.2019

Institut für Informationssicherheit der Universität Stuttgart

2019

Contents

List of Figures

Abbreviations

	E-Voting (cf. Section 2.1)
ch	choice
n_{choices}	number of choices
C	set of choices
$\mathsf{abstain}$	choice to abstain
f_{res}	result function
V	voter
n_{voters}	number of voters
$n_{\mathsf{voters}}^{\mathsf{honest}}$	number of honest voters
$n_{\mathsf{voters}}^{\mathsf{dishonest}}$	number of dishonest voters
Vote	voting program of voter
b	ballot
$\mathsf{Verify}_{\mathsf{V}}$	verification program of voter
$p_{\mathsf{verif}}^{\mathsf{vote}}$	probability that voter verifies
$p_{\mathsf{verif}}^{\mathsf{abst}}$	probability that abstaining voter verifies
VSD	voter supporting device
$\mathsf{VoteVSD}$	voting program of voter supporting device
$\mathsf{VerifyVSD}$	verification program of voter supporting device
$p_{\mathsf{verif}}^{\mathsf{vsd}}$	probability that voter supporting devices verifies
T	trustee
n_{trustees}	number of trustees
$n_{\mathsf{trustees}}^{\mathsf{honest}}$	number of honest trustees
Tally	tallying program of trustees
M	mix server
n_{servers}	number of mix servers
B	bulletin board
AS	authentication server
	Computational model (cf. Section 2.2)
S	scheduler
ITM	interactive Turing machine

π / $\pi^{(\ell)}$	process (with security parameter 1^ℓ)
$\pi_1 \| \pi_2$	connection of processes π_1 and π_2
P	protocol
Σ	set of agents of a protocol
a	agent
$\hat{\pi}_{\mathsf{a}}$	honest program of agent a
A	adversary
r	protocol run
γ	property/set of protocol runs of a protocol
$\neg\gamma$	complement of γ
	Verifiability (cf. Section 2.3)
J	judge
accept	protocol run is accepted (by judge)
reject	protocol run is rejected (by judge)
φ	trust assumptions
$\gamma(k, \varphi)$	end-to-end verifiability goal
	Accountability (cf. Section 2.4)
ψ	verdict
$\mathsf{dis}(\mathsf{a})$	verdict that agent a is dishonest
C	accountability constraint
Φ	accountability property
	Privacy (cf. Section 2.5)
$\mathsf{V}_{\mathsf{obs}}$	voter under observation
$\mathcal{I}_{\mathsf{voting}}$	ideal voting protocol
δ^{ideal}	privacy level of ideal voting protocol
	Encryption (cf. Appendix A.1)
$\mathcal{E}$	public-key encryption scheme
KeyGen	key generation algorithm
$\mathsf{KeyShareGen}$	key share generation algorithm
$\mathsf{PublicKeyGen}$	public key generation algorithm
Enc	encryption algorithm
Dec	decryption algorithm
$\mathsf{DecShare}$	decryption share algorithm
sk	secret key
sk_k	k-th secret key share
pk	public key
dec_k	k-th decryption share

	Signatures (cf. Appendix A.2)
$\mathcal{S}$	signature scheme
KeyGen	key generation algorithm
Sign	signature algorithm
Verify	verification algorithm
MPC	multi-party computation
NIZKP	non-interactive zero-knowledge proof (cf. Appendix A.3)

Abstract

Electronic voting (e-voting) systems are used in numerous countries for political elections, but also for less critical elections within clubs and associations, and hence affect the lives of millions of people. It is therefore important to ensure that single voters' choices remain *private*, and to be able to *verify* that an election result coincides with the voters' intention. Unfortunately, for most e-voting systems employed in real elections, these fundamental security and privacy properties cannot be guaranteed, so that in particular the legitimacy of such political elections is challenged.

This demonstrates the importance of employing e-voting systems that are rootedly designed to guarantee the required security. However, it turned out to be highly challenging to construct secure yet practical e-voting systems since one always has to find a balance between the (possibly conflicting) requirements of the given kind of election.

In the first two chapters of the thesis' main part, we present two practical e-voting systems which are both meant for low-risk and non-political elections, e.g., within clubs or associations. We have implemented both systems to demonstrate their practicability. The first system, called *sElect*, is designed to be as simple as possible while still guaranteeing a good level of security. The second system, called *Ordinos*, provides a superior level of privacy as it only reveals the most necessary information about the election outcome, e.g., solely the winner's name but nothing else. We will rigorously analyze the security of sElect and Ordinos. To do this, we formally define the required security properties and then mathematically prove that sElect and Ordinos achieve them.

In the third chapter of the thesis' main part, we provide substantial work on the fundamental notion of verifiability of e-voting systems. We analyze and compare all formal verifiability definitions from the literature regarding how meaningful, expressive, or general they are.

Kurzzusammenfassung

Elektronische Wahlsysteme werden in zahlreichen Ländern der Welt für politische Wahlen, aber auch für weniger kritische Abstimmungen in Vereinen oder Verbänden, benutzt und beeinflussen so das Leben vieler Menschen. Daher ist es wichtig sicherzustellen, dass die einzelnen Wählerstimmen *geheim* bleiben und dass man *verifizieren* kann, dass das Resultat einer Wahl mit dem Willen der Wähler übereinstimmt. Diese fundamentalen Sicherheits- und Vertraulichkeitseigenschaften können jedoch für die meisten in der Praxis eingesetzten elektronischen Wahlsysteme nicht garantiert werden. Dies stellt insbesondere die Legitimität solcher politischer Wahlen infrage.

Deshalb ist es wichtig, elektronische Wahlsysteme zu benutzen, die von Grund auf so entworfen wurden, dass sie die gewünschte Sicherheit liefern. Es hat sich allerdings als äußerst anspruchsvoll herausgestellt, elektronische Wahlsysteme zu konstruieren, die sowohl sicher als auch praktikabel sind, da man dabei stets eine Balance zwischen Eigenschaften finden muss, die sich gegenseitig ungünstig beeinflußen (können).

In den ersten zwei Hauptteilen dieser Arbeit werden wir zwei praktische elektronische Wahlsysteme vorstellen, die beide für risikoarme, nichtpolitische Wahlen entworfen wurden, etwa in Vereinen oder Verbänden. Wir haben beide Systeme implementiert, um ihre Praktibilität zu demonstrieren. Das erste System heißt *sElect* und ist so gestaltet, dass es möglichst einfach ist und gleichzeitig ein gutes Maß an Sicherheit garantiert. Das zweite System heißt *Ordinos* und liefert einen hohen Grad an Geheimhaltung, indem es nur die absolut notwendigen Informationen als Resultat ausgibt, wie beispielsweise nur den Namen des Gewinners und sonst nichts. Wir werden die Sicherheit von sElect und Ordinos rigoros untersuchen. Dazu definieren wir formal die gewünschten Sicherheitseigenschaften und beweisen dann mathematisch, dass sElect und Ordinos diese Eigenschaften erfüllen.

Im dritten Hauptteil liefern wir eine grundlegende Arbeit zur elementaren Idee von Verifizierbarkeit elektronischer Wahlsysteme. Dazu analysieren und vergleichen wir alle formalen Verifizierbarkeitsdefinitionen aus der Literatur im Hinblick darauf, wie sinnvoll, aussagekräftig oder allgemeingültig sie sind.

Chapter 1

Introduction

Systems for electronic voting (e-voting systems) have been employed in many countries for national and municipal (political) elections, for instance in the US, Norway, Estonia, India, Belgium, Switzerland, and Brazil. E-voting systems are also often used in other kinds of elections within companies, associations, clubs, etc. Many companies build e-voting systems and offer e-voting services. There are roughly two types of e-voting systems: (*i*) those where the voter has to go to a polling station in order to cast her vote using a voting machine, and (*ii*) those that allow the voter to cast her vote over the Internet using her electronic devices. In this thesis, we focus on the latter scenario, i.e., on remote e-voting, but most of the methods and techniques to be developed in this thesis should also apply to the former scenario.

Some of the most important security properties modern e-voting systems should satisfy are (vote) privacy, verifiability, accountability, and coercion-resistance as explained next.

Privacy means that outside observers or even insiders (e.g., voting authorities) should not be able to tell how specific voters voted.

As for *verifiability*, we note that in most existing e-voting systems employed in elections so far, voters do not have any guarantees that their votes have actually been counted. The voters' computers, the voting machines or the voting servers might have programming errors or, even worse, might have been manipulated deliberately by insider or outsider attackers. E-voting systems are complex hardware and software systems, and as in all such systems, programming errors and security vulnerabilities are unavoidable. Not surprisingly, numerous problems with e-voting systems have been reported in various countries (see, e.g., [JRSW04, CFH+07, Tod08, Dan09, WWH+10, WWIH12, SFD+14, Eps15, And11, Loe14, IT17]). We illustrate three of these examples below:

- In the 2011 New Jersey Primary Election [And11], votes for competing

parties were swapped "as a result of human error in the programming of the voting machine used in this election".[1] The bug was only detected by chance because the election result was obviously flawed. The bug would most likely have been remained undetected otherwise, as the voting protocol was not verifiable.

- An e-voting system that was used in Belgium for European, federal and regional elections in 2014, caused "incoherent election results when it tried to add up preferential votes", according to a spokesman of the ministry [Loe14]. The bug was detected because the software output different results for the same input. As in the example above, it was fortunate that the election result was obviously incorrect, as the underlying voting protocol did not provide any means to verify the correctness of the election result.

- As part of a hacking competition, thirty voting machines that were used in different national elections in the US were tested [IT17]. It was demonstrated that most of these machines provided only low level of security. For example, some had physical ports open that could be used to install malicious software to tamper with votes. Some of the machines included poorly secured Wi-Fi connectivity, allowing hackers to access and manipulate the machines remotely.

Therefore, one aims at *verifiable* e-voting systems. Such systems guarantee that if in an election the published result is not correct, i.e., it does not correspond to the votes actually cast by eligible voters, then this is detected (with some probability) by voters or possibly external observers. Importantly, this property should hold true even if voting machines and servers have programming errors or are outright malicious.

In practice, an even stronger property, called *accountability*, is desirable, which not only requires that manipulations can be detected but that specific misbehaving parties can be identified (and hence, punished).

In modern e-voting systems, verifiability and accountability are generally achieved in the following way: As well as the result of the election, systems publish additional data and voters are provided with some kind of a receipt which they can use to check that their votes were actually counted. However, care has to be taken in order not to jeopardize the voters' privacy.

Some modern e-voting system even provide so-called *coercion resistance*. That is, it should be impossible to coerce voters to vote in a certain way and to protect against vote-selling.

[1] Election officials declared that they wanted to avoid the cost to the county of hiring a programmer.

1.1 Cryptographic Security Analysis

In order to find out whether a given voting system achieves its desired security properties, informally analyzing its security is not sufficient since critical aspects can easily be overlooked. Therefore, it is necessary to formally analyze the security of voting systems based on reasonable and formal security definitions.

There have been major achievements in the field of rigorous cryptographic analysis of e-voting systems in the last decade or so. Formal definitions for the central security requirements have been proposed and studied (see, e.g., [KTV10a, KTV10b, CGK+16a, KTV11, BCG+15]). Some of these definitions are formulated in general and widely applicable frameworks so that they can be applied to virtually any e-voting protocols. These frameworks and definitions have been applied to perform rigorous security analysis of various existing e-voting systems (see, e.g., [CS11, ACW13, CEK+15, CW17, KT16, KTV14, KTV12b, KTV11, KTV10c, KTV10a, KTV10b]), often with surprising results, and newly proposed systems more and more often come with security proofs right away (see, e.g., [KMST16a, KZZ15b, KZZ15a, KZZ17, CCFG16]).

The history of e-voting demonstrates that designing and employing secure yet practical e-voting systems is a challenging task. The reason is that different requirements need to be guaranteed which may influence each other (adversely). Hence, there does not seem to exist a "one size fits all" e-voting system which achieves all desired requirements. So, when designing an e-voting system, one has to find a good balance between security, usability, and efficiency. In particular, one has to always consider for which kind of election the e-voting system is supposed to be used. This depends, for example, on the given infrastructure, (technical) abilities of the voters and developers, and the power of possible attackers.

In the main part of this thesis, we introduce, formally analyze, and implement two remote e-voting systems, each of which provides its own balance between security and simplicity. Further, we analyze all important verifiability definitions in order to improve the understanding of this fundamental security requirement of e-voting systems. In the next section, we provide a more detailed overview of these contributions.

1.2 Contributions and Structure of the Thesis

Our contributions can be outlined as detailed next.

The underlying ideas have been developed during joint discussion and meetings with the co-authors of the publications listed below. The majority of the technical work was done by Johannes Müller (see below for details).

sElect: A Lightweight Verifiable Remote E-Voting System

In Chapter 3, we propose a new practical voting system called *sElect* (secure/simple elections). This system, which we implemented as a platform independent web-based application, is meant for *low-risk elections*, such as elections within clubs and associations, rather than national elections, where—besides a reasonable level of security—simplicity and convenience are important. sElect is designed to be particularly simple and lightweight in terms of its structure, the cryptography it uses, and the user experience.

sElect combines several concepts, such as verification codes (see, e.g., [DLM82]) and Chaumian mix nets [Cha81], in a novel way. One of the unique features of sElect is that it supports fully automated verification, which does not require any user interaction and is triggered as soon as a voter looks at the election result.

Despite its simplicity, we formally prove that sElect provides a good level of privacy, verifiability, and accountability for low-risk elections. The system is not meant to defend against coercion and mostly tries to defend against untrusted or malicious authorities, including inadvertent programming errors or deliberate manipulation of servers, but excluding targeted and sophisticated attacks against voters' devices.

This chapter is based on the following two publications. The second publication is the full version of the first one, including the complete protocol model and formal proofs.

- Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 341–354, 2016. See also [KMST16a].

- Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. *IACR Cryptology ePrint Archive*, 2016:438, 2016. See also [KMST16b].

The system has been implemented by Enrico Scapin. See Scapin's dissertation [Sca18] for details.

Ordinos: A Verifiable Tally-Hiding Remote E-Voting System

Almost all e-voting systems reveal the complete election result, consisting of the exact number of votes per candidate or even all individual votes. This is often undesirable for various reasons. For example, in elections with only a few voters (e.g., boardroom or jury votes), revealing the complete result leads to a low privacy level, possibly deterring voters from voting according to their actual preference. Instead, merely revealing the winner or a ranking of candidates is often sufficient. This property is called tally-hiding. Although tally-hiding offers completely new options for verifiable e-voting, it so far has not received much attention in the literature.

In Chapter 4, we present Ordinos, the first provably secure verifiable tally-hiding e-voting system. Ordinos is a generic extension of the prominent Helios remote e-voting system [Adi08]. Whereas Helios always reveals the full result, Ordinos supports several tally-hiding result functions, including revealing only the winner of an election, the k best/worst candidates, or the overall ranking, with or without disclosing the number of votes per candidate.

We carry out a detailed cryptographic analysis proving that Ordinos provides privacy, verifiability, and accountability. With result functions that hide most of the full election result, the level of privacy Ordinos provides can be much better than Helios. Our cryptographic analysis of Ordinos is based on generic properties of the cryptographic primitives employed. This means that they can be instantiated by arbitrary cryptographic constructions satisfying these properties. We propose one such instantiation using among others Paillier public-key encryption, an MPC protocol for greater-than by Lipmaa and Toft [LT13], as well as NIZKPs by Schoenmakers and Veeningen [SV15]. We implemented Ordinos based on this instantiation and evaluated its performance, demonstrating its practicability.

This chapter is based on the following paper.

- Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding Remote E-Voting System. *Currently under submission.*

The proof of the ideal privacy level was done by Andreas Vogt. The cryptographic primitives and the MPC protocol were implemented by Johannes Müller, and Julian Liedtke has extended the implementation so that it can be executed over a local network or the Internet.

Verifiability Notions for E-Voting Protocols

There have been intensive research efforts in the last two decades or so to design and deploy electronic voting (e-voting) protocols/systems which allow voters and/or external auditors to check that the votes were counted correctly. As described in Section 1.1, this security property is called *verifiability*. It is meant to defend against voting devices and servers that have programming errors or are outright malicious. In order to properly evaluate and analyze e-voting protocols w.r.t. verifiability, one fundamental challenge has been to formally capture the meaning of this security property. While the first formal definitions of verifiability were devised as early as the late 1980s, new verifiability definitions are still being proposed. The definitions differ in various aspects, including the classes of protocols they capture and even their formulations of the very core of the meaning of verifiability. This is an unsatisfying state of affairs, leaving the research on the verifiability of e-voting protocols in a fuzzy state.

In Chapter 5, we review all formal definitions of verifiability proposed in the literature and cast them in a framework proposed by Küsters, Truderung, and Vogt (the KTV framework, see Section 2.3), yielding a uniform treatment of verifiability. This enables us to provide a detailed comparison of the various definitions of verifiability from the literature. We thoroughly discuss advantages and disadvantages, and point to limitations and problems. Finally, from these discussions and based on the KTV framework, we distill a general definition of verifiability, which can be instantiated in various ways, and provide precise guidelines for its instantiation. The concepts for verifiability we develop should be widely applicable also beyond the framework used here. Overall, our work offers a well-founded reference point for future research on the verifiability of e-voting systems.

This chapter is based on the following two publications. The second publication is the full version of the first one, including analyses of further verifiability definitions.

- Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Sok: Verifiability Notions for E-Voting Protocols. In *IEEE Symposium on Security and Privacy, S&P 2016, San Jose, CA, USA, May 22-26, 2016*, pages 779–798, 2016. See also [CGK+16a].

- Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Verifiability Notions for E-Voting Protocols. *IACR Cryptology ePrint Archive*, 2016:287, 2016. See also [CGK+16b].

Further Contributions

Rigorous cryptographic security analysis plays an important role in the design of modern e-voting systems. There has been huge progress in this field in the last decade or so in terms of formalizing security requirements and formally analyzing e-voting systems. In the following paper, we have summarized some of the achievements and lessons learned, which, among others, challenge common believes about the role of and the relationships between central security requirements. The majority of papers on which the following paper is based did not have Johannes Müller as an author.

- Ralf Küsters and Johannes Müller. Cryptographic Security Analysis of E-voting Systems: Achievements, Misconceptions, and Limitations. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, pages 21–41, 2017. Invited paper. See also [KM17].

In contrast to the contributions/papers above, this work is not captured in its own chapter. Instead, fragments of this work can be found in Chapter 2, where we formally introduce the fundamental security definitions for e-voting systems and provide some background on their relationships.

Chapter 2

Secure Electronic Voting

In this chapter, we first provide some background on e-voting and introduce notation that we use throughout the work. In Section 2.2, we describe the framework in which we model e-voting protocols. In Sections 2.3 to 2.5, we introduce the formal definitions of verifiability, accountability, and privacy that we apply to formally analyze e-voting protocols.

2.1 E-Voting in a Nutshell

An electronic voting system is a distributed system. In such a system, a *voter*, possibly using some *voter supporting device (VSD)* (e.g., a desktop computer or smartphone), computes a ballot, typically containing the voter's choice in an encrypted or encoded form, and casts it. Often this means that the ballot is put on a bulletin board (see also below). The ballots are collected (e.g., from the bulletin board) and tallied by *trustees/voting authorities*. In modern e-voting protocols, the tallying is, for example, done by combining all ballots into one, using homomorphic encryption, and then decrypting the resulting ballot, or by using mix-nets, where the ballots before being decrypted are shuffled. At the beginning of an election, the voting authorities produce the election parameters prm, typically containing keys and a set of valid choices C, the *choice space*. In general, C can be an arbitrary set, containing just the set of candidates, if voters can choose one candidate among a set of candidates, or even tuples of candidates, if voters can choose several candidates or rank them. We emphasize that we consider abstention to be one of the choices in C.

In this work, we denote the voters by $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$ and their VSDs by $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$ (if any). In order to cast a vote, a voter V_i first picks her choice $\mathsf{ch}_i \in \mathsf{C}$. She then runs her voting procedure $\mathsf{Vote}(\mathsf{ch}_i)$, which

in turn might involve providing her VSD with her choice. The VSD runs some procedure $\mathsf{VoteVSD}$, given certain parameters, e.g., the voter's choice. The result of running the voting procedure is a ballot b_i, which, for example, might contain ch_i in encrypted form. Some models or voting protocols do not distinguish between the voter and her VSD, and in such a case, we simply denote the voter's voting procedure by Vote.

Often voters have to perform some verification procedure during or at the end of the election in order to prevent/detect malicious behavior by their VSDs or the voting authorities. We denote such a procedure by Verify. This procedure might for example involve checking that the voter's ballot appears on the bulletin board or performing certain cryptographic tasks. Carrying out Verify will often require some trusted device.

We denote the trustees by $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$.[1] As mentioned, they collect the ballots, tally them, and output the election result res, which belongs to what we call the *result space* (fixed for a given election). The result is computed according to a *result function* f_{res} which takes as input the voters' choices $\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}}$ and outputs res. (Of course, dishonest trustees might try to manipulate the election outcome, which by the verifiability property, as discussed in Section 2.3, should be detected.) The result function should be specified by the election authorities before an election starts.

At the end or throughout the election, *auditors/judges* might check certain information in order to detect malicious behavior. Typically, these checks are based solely on publicly available information, and hence, in most cases their task can be carried out by any party. They might, for example, check certain zero-knowledge proofs. In what follows, we consider the auditors/judges to be one party J, who is assumed to be honest.

As already noted above, most election protocols assume an append-only *bulletin board* B. An honest bulletin board stores all the input it receives from arbitrary participants in a list, and it outputs the list on request. Typically, public parameters, such as public keys, the election result, voters' ballots, and other public information, such as zero-knowledge proofs generated by voting authorities, are published on the bulletin board. As we will see, in most models (and many protocols) a single honest bulletin board is assumed. However, trust can be distributed [CS14, KKL+18]. Providing robust and trustworthy bulletin boards, while very important, is mainly considered to be a task orthogonal to the rest of the election protocol. For this reason, we will mostly refer to *the* (honest) bulletin board B, which in practice might involve a distributed solution rather than a single trusted server.

[1] We note that trustees can also mix servers. For the sake of simplicity, we only use the term "trustee" in what follows and regard mix servers as a special case of trustees.

2.2 Computational Model

In this section, we describe the computational model [KTV10b] that we will use in this thesis to formally model e-voting protocols and analyze their security.

Processes are the core of the computational model. Based on them, protocols are defined.

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

We write a process π as $\pi = p_1||\cdots||p_l$, where $p_1 \ldots, p_l$ are programs. If π_1 and π_2 are processes, then $\pi_1||\pi_2$ is a process, provided that the processes are connectible: two processes are *connectible* if common external channels, i.e., channels with the same name, have opposite directions (input/output); internal channels are renamed, if necessary. A process π where all programs are given the security parameter 1^ℓ is denoted by $\pi^{(\ell)}$. In the processes we consider the length of a run is always polynomially bounded in ℓ. Clearly, a run is uniquely determined by the random coins used by the programs in π.

Protocol. A *protocol* P is defined by a set of agents Σ (also called *parties* or *protocol participants*), and a program $\hat{\pi}_{\mathsf{a}}$ which is supposed to be run by the agent. This program is the *honest program* of a. Agents are pairwise connected by channels and every agent has a channel to the adversary (see below).[2]

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

[2] We note that in [KTV10b] agents were assigned sets of potential programs they could run plus an honest program. Here, w.l.o.g., they are assigned only one honest program (which, however, might be corrupted later on).

scheduler would trigger protocol participants according to the phases of an election, e.g., (*i*) register, (*ii*) vote, (*iii*) tally, (*iv*) verify.

If $\hat{\pi}_{\mathsf{a}_1}, \ldots, \hat{\pi}_{\mathsf{a}_n}$ are the honest programs of the agents of P, then we denote the process $\hat{\pi}_{\mathsf{a}_1}|| \ldots ||\hat{\pi}_{\mathsf{a}_n}$ by π_P.

The process π_P is always run with an adversary A. The adversary may run an arbitrary probabilistic polynomial-time program and has channels to all protocol participants in π_P. Hence, a *run r of P* with adversary (adversary program) π_{A} is a run of the process $\pi_P||\pi_{\mathsf{A}}$. We consider $\pi_P||\pi_{\mathsf{A}}$ to be part of the description of r, so that it is always clear to which process, including the adversary, the run r belongs.

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 $\mathsf{corrupt}$. 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 or a judge, will typically not be corruptible, i.e., they would ignore $\mathsf{corrupt}$ messages. Also, agents might only accept $\mathsf{corrupt}$ message upon initialization, modeling static corruption. Altogether, this allows for great flexibility in defining different kinds of corruption, including various forms of static and dynamic 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 $\mathsf{corrupt}$ 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 $\pi_P||\pi_{\mathsf{A}}$, i.e., a always ignores all $\mathsf{corrupt}$ messages.

Property. A *property γ of P* is a subset of the set of all runs of P.[3] By $\neg\gamma$ we denote the complement of γ.

Negligible, overwhelming, δ-bounded. As usual, a function f from the natural numbers to the interval $[0, 1]$ is *negligible* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \frac{1}{\ell^c}$ for all $\ell > \ell_0$. The function f is *overwhelming* if the function $1 - f$ is negligible. A function f is *δ-bounded* if, for every $c > 0$ there exists ℓ_0 such that $f(\ell) \leq \delta + \frac{1}{\ell^c}$ for all $\ell > \ell_0$.

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

2.3 Verifiability

In Section 1.1, we have seen that numerous e-voting systems suffer from flaws that make it possible for more or less sophisticated attackers to change the election result. Therefore, modern e-voting systems strive for what is called verifiability, more precisely *end-to-end verifiability*. Roughly speaking, end-to-end verifiability requires that voters and possibly external auditors should be able to check whether the published election result is correct, i.e., corresponds to the votes cast by the voters, even if voting devices and servers have programming errors or are outright malicious.

In the remainder of this section, we first recapitulate the general verifiability definition by Küsters et al. [KTV10b] (Section 2.3.1) and then show how this general definition can be instantiated to model end-to-end verifiability (Section 2.3.2). This definition of end-to-end verifiability will be used to analyze verifiability of the sElect e-voting system in Section 3.4 and of the Ordinos e-voting system in Section 4.4. In Section 2.3.3, we then discuss the prominent notions of individual and universal verifiability. Following [KMST16a, KTV11, KTV12b], we show that, unlike commonly believed, these two notions fail to provide a solid basis for verifiability. In particular, they are neither necessary nor sufficient to achieve end-to-end verifiability. In Chapter 5, we will provide a detailed and extensive analysis of all formal verifiability definitions that have been proposed in the literature so far.

2.3.1 Generic Verifiability Definition

About 30 years ago, Benaloh provided a first definition of end-to-end verifiability [Ben87]. As discussed in Section 5.3, while Benaloh's definition is fairly simple and captures the essence of verifiability, it requires unrealistically strong properties so that it would reject even reasonable e-voting systems.

In [KTV10b], Küsters, Truderung, and Vogt introduced a generic framework (the *KTV framework*) for verifiability and, more precisely, the even stronger notion of accountability (see Section 2.4). They also instantiated the framework to define end-to-end verifiability; also called *global verifiability* in [KTV10b], in contrast to individual and universal verifiability (see Section 2.3.2 and 2.3.3). This framework and definition since then have been used to analyze several e-voting protocols and mix nets, such as Helios, ThreeBallot, VAV, Wombat Voting, sElect, Ordinos, Chaumian RPC mix nets, and re-encryption RPC mix nets [KTV10b, KTV14, KTV12b, KTV11, KT16, KMST16a]. It can also be applied to other domains, such as auctions and contract signing [KTV10b]. Interestingly, in Chapter 5, we will

demonstrate that is possible to cast all formal verifiability definitions from the literature into the generic KTV framework.

In what follows, we recall the KTV framework and then, in Section 2.3.2, its instantiation which captures end-to-end verifiability. We note that in the original publication [KTV10b], formalizations both in a symbolic as well as a computational model were presented. Here, as throughout the thesis, we concentrate on the computational model, as introduced in Section 2.2.

The KTV framework comes with a general definition of verifiability which in particular can be instantiated to model end-to-end verifiability (see Section 2.3.2). The definition assumes a *judge* J whose role is to accept or reject a protocol run by writing accept or reject on a dedicated channel ψ_{J}. To make a decision, the judge runs a so-called *judging procedure*, which performs certain checks (depending on the protocol specification), such as verification of all zero-knowledge proofs (if any) and taking voter complaints into account. Intuitively, J accepts a run if the protocol run looks as expected. The judging procedure should be part of the protocol specification. So, formally the judge should be one of the protocol participants in the considered protocol P, and hence, precisely specified. The input to the judge typically 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 definition of verifiability is centered around the notion of a *goal* of the protocol. Formally, a goal is simply a property γ of the system, i.e., a set of runs (see Section 2.2). Intuitively, such a goal specifies those runs which are "correct" in some protocol-specific sense. For e-voting, intuitively, the goal would contain those runs where the announced result of the election corresponds to the actual choices of the voters.

Now, the idea behind the definition is very simple. The judge J should accept a run only if the goal γ is met, and hence, the published election result corresponds to the actual choices of the voters. More precisely, the definition requires that the probability (over the set of all runs of the protocol) that the goal γ is not satisfied but the judge nevertheless accepts the run is δ-bounded. 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.

By $\mathsf{Pr}[\pi^{(\ell)} \mapsto (\mathsf{J}\colon \mathsf{accept})]$ we denote the probability that π, with security parameter 1^{ℓ}, produces a run which is accepted by J. Analogously, by $\mathsf{Pr}[\pi^{(\ell)} \mapsto \neg\gamma,\ (\mathsf{J}\colon \mathsf{accept})]$ we denote the probability that π, with security

parameter 1^ℓ, produces a run which is not in γ but nevertheless accepted by the judge J.

Definition 1 (Verifiability)**.** *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 $\pi = (\pi_P || \pi_\mathsf{A})$, the probability*

$$\mathsf{Pr}[\pi^{(\ell)} \mapsto \neg\gamma,\ (\mathsf{J}\colon \mathsf{accept})]$$

is δ-bounded as a function of ℓ.

A protocol P could trivially satisfy verifiability with a judge who never accepts a run. Therefore, one of course would also require a *soundness* or *fairness* condition. That is, one would except at the very least that if the protocol runs with a benign adversary, which, in particular, would not corrupt parties, then the judge accepts a run. Formally, for a benign adversary π_A we require that $\mathsf{Pr}[\pi^{(\ell)} \mapsto (\mathsf{J}\colon \mathsf{accept})]$ is overwhelming. One could even require that the judge accepts a run as soon as a certain subset of protocol participants are honest, e.g., the voting authorities (see, e.g., [KTV10b] for a more detailed discussion). These kinds of fairness/soundness properties can be considered to be sanity checks of the judging procedure and are typically easy to check. Most definitions of verifiability in the literature do not explicitly mention this property. For brevity of presentation, we therefore mostly ignore this issue here as well.

Definition 1 captures the essence of the notion of verifiability in a very simple way, as explained above. In addition, it provides great flexibility and it is applicable to arbitrary classes of e-voting protocols. This is in contrast to most other definitions of verifiability, as we will see in Chapter 5, which are mostly tailored to specific classes of protocols. This flexibility in fact lets us express the other definitions in terms of Definition 1. There are two reasons for the flexibility. First, the notion of a protocol P used in Definition 1 is very general: a protocol is simply an arbitrary set of interacting Turing machines, with one of them playing the role of the judge. Second, the goal γ provides great flexibility in expressing what an e-voting protocol is supposed to achieve in terms of verifiability.

Remark 1. *Note that whether a verifiable protocol is in fact* verified *typically depends on the specific protocol run. For example, while a given protocol may, in theory, be perfectly verifiable, it is possible that the required checks to ensure a goal γ are so difficult to use that they are not executed in practice (with sufficiently high probability). Hence, verifiability is necessary but not sufficient to guarantee that a goal γ (e.g., the correctness of the election result) is actually verified.*

2.3.2 End-to-End Verifiability

On a high level, Küsters et al. capture end-to-end verifiability in the KTV framework as follows. The probability that a run is accepted (by a judge or other observers), but the published election result does not correspond to the actual votes cast by the voters is small (bounded by some parameter δ). More specifically, the result should contain all votes of the honest voters, except for at most k honest votes (for some parameter $k \geq 0$), and it should contain at most one vote for every dishonest voter.

More precisely, Küsters et al. [KTV10b] proposed the following instantiation of the generic verifiability definition to capture end-to-end verifiability. To this end, they introduce a family of goals which has been slightly refined in [CGK+16a] to yield a family of goals of the form $\gamma(k, \varphi)$. The parameter φ is a Boolean formula that describes which protocol participants are assumed to be honest in a run, i.e., those participants which cannot be corrupted by the adversary. For example, if we want to model that the scheduler S, the judge J, and the bulletin board B are assumed to be honest while all other participants can actively deviate from their honest programs, we set $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$.

On a high level, the parameter k denotes the maximum number of choices made by the honest voters that the adversary is allowed to manipulate. So, roughly speaking, altogether the goal $\gamma(k, \varphi)$ contains all runs of a voting protocol P where either (i) the trust assumption φ is violated (e.g., at least one of the parties S, J, or B in the example above have been corrupted), or (ii) where φ holds true and the adversary has manipulated at most k votes of honest voters and every dishonest voter votes at most once. Before we formally define the goal $\gamma(k, \varphi)$ below, we illustrate the goal to provide some intuition.

Example 1. *Consider a run of an e-voting protocol with three honest voters and two dishonest voters. We set* $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$ *as above. Assume that there are two candidates/choices* A *and* B*, and that the tallying function returns the number of votes for each candidate. Now, if all honest voters vote for, say,* A*, the final result equals* $(A, B) = (2, 2)$*, and* φ *holds true, then* $\gamma(k, \varphi)$ *is achieved for all* $k \geq 1$*: one vote of an honest voter is missing, and there is at most one vote for every dishonest voter. Conversely,* $\gamma(0, \varphi)$ *is not achieved because this would require that all votes of the honest voters are counted, which is not the case here.*

We now formally define the 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

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

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.[4] 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 $\vec{\mathsf{c}}_{\mathsf{real}} = (\mathsf{ch}_i^{\mathsf{real}})_{i \leq n_{\mathsf{voters}}}$ 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) $d(\vec{\mathsf{c}}_{\mathsf{ideal}}, \vec{\mathsf{c}}_{\mathsf{real}}) \leq k$,

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 2.2) and the possible choices $(\mathsf{ch}'_i)_{i \in I_{\mathsf{dishonest}}}$ made by the dishonest voters.

With this definition of goals, Definition 1 captures end-to-end verifiability: the probability that the judge accepts a run where more than k votes of honest voters were manipulated or dishonest voters could cast too many votes, is bounded by δ. In security statements about concrete e-voting protocols (see, e.g., Theorem 2 or 5), δ will typically depend on various parameters, such as k and the probability that voters perform certain checks. While $k = 0$ is desirable, this is in most cases impossible to achieve because, for example, voters might not always perform the required checks, and hence, there is a chance that manipulation of votes goes undetected.

Importantly, this definition of end-to-end verifiability allows one to *measure* the level of end-to-end verifiability an e-voting protocol provides.

2.3.3 Individual and Universal Verifiability

Sako and Kilian [SK95] introduced the notions of *individual* and *universal verifiability*. These requirements (and subsequent notions, such as *cast-as-intended*, etc.) have become very popular and are still used to design and

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

analyze e-voting systems. According to Sako and Kilian, an e-voting system achieves individual verifiability if "a sender can verify whether or not his message has reached its destination, but cannot determine if this is true for the other voters". Universal verifiability guarantees that it is possible to publicly verify that the tallying of the ballots is correct. That means that the final election result exactly reflects the content of those ballots that have been accepted to be tallied.

The notions of individual and universal verifiability have later been formalized by Chevallier-Mames et al. [CFP+10] (only universal verifiability), Cortier et al. [CEK+15], and Smyth et al. [SFC15]. As demonstrated in Chapter 5, these notions can also be captured in the KTV framework.

A Common Misconception. Unfortunately, it is often believed (see, e.g., [SFC15]) that individual together with universal verifiability implies end-to-end verifiability, which is the security property that e-voting systems should achieve. However, in [KTV10b, KTV12b, KMST16a], Küsters et al. have demonstrated that individual and universal verifiability are *neither sufficient nor necessary* for end-to-end verifiability.

In short, there are e-voting systems, such as ThreeBallot and VAV [Smi07] as well as variants of Helios, that arguably provide individual and universal verifiability but whose verifiability is nevertheless broken, i.e., they do not provide end-to-end verifiability. Conversely, there are e-voting systems, such as sElect [KMST16a] (see also Chapter 3), which provide end-to-end verifiability without having to rely on universal verifiability.

In what follows, we explain these results in more detail.

Not Sufficient

We recall several attacks that break the end-to-end verifiability of e-voting systems, even though these systems provide individual and universal verifiability. The first class of attacks uses that (dishonest) voters, possibly with the help of malicious authorities, might cast malformed ballots. In the second class of attacks (so-called *clash attacks*), the same receipt is shown to different voters who voted for the same candidate, allowing malicious voting devices and authorities to drop or manipulate ballots.

An Illustrative Example: A Modification of Helios. Helios [Adi08] is one of the most prominent remote e-voting systems which, on a high level, works as follows. Trustees share a secret key sk which belongs to a public/private ElGamal key pair $(\mathsf{pk}, \mathsf{sk})$. Voters encrypt the candidate of their choice under the public key pk and submit the resulting ciphertext to the bulletin board. Then all ciphertexts are publicly multiplied so that, by the

homomorphic property of the ElGamal public-key encryption scheme, the resulting ciphertext encrypts the number of votes for each candidate. Finally, the trustees perform distributed and verifiable decryption of this ciphertext and publish the resulting plaintext as the outcome of the election.

In order to guarantee the integrity of the election result, several zero-knowledge proofs (ZKP) are used. Among others, a voter has to prove that her ciphertext encrypts a valid choice, and, for privacy reasons, that she knows which choice it encrypts.

It has been formally proven in [KTV12b, CGGI14] that Helios is end-to-end verifiable under certain assumptions. Furthermore, assuming that the voting devices are honest, Helios provides individual verifiability because each voter can check whether her ballot appears on the bulletin board. Universal verifiability follows from the fact that the multiplication of the ciphertexts on the bulletin board is public and that the tellers perform verifiable decryption. Thus, Helios provides end-to-end verifiability as well as individual and universal verifiability.

To see that individual and universal verifiability together do not imply end-to-end verifiability consider a modification of Helios in which voters do not have to prove that their votes are correct, i.e., dishonest voters may cast malformed ballots without being detected. Then a (single!) dishonest voter could completely spoil the election result by encrypting an invalid choice. Such a malformed ballot might contain negative votes for certain candidates, and hence, effectively subtracting votes from candidates, or the malformed ballot might contain many more votes for a candidate than allowed. So, such a system certainly does not provide end-to-end verifiability. At the same time, such a system can still be considered to provide individual and universal verifiability. Voters can still check that their ballots appear on the bulletin board (individual verifiability), and ballots on the bulletin board can still be tallied in a universally verifiable way. But dishonest voters might have spoiled the election result completely and this is not detected.[5]

This simple example demonstrates that, even if a voting system achieves individual and universal verifiability, its overall verifiability can nevertheless completely and trivially be broken.

Another Example: ThreeBallot. The attack illustrated above conceptually also applies to the ThreeBallot voting system [Smi07] (also to VAV), but the details of the attack differ. We start by briefly describing how ThreeBallot works.

In ThreeBallot, a voter is given a multi-ballot consisting of three simple

[5]Note that the arguments hold true even when assuming that only eligible voters (honest or dishonest) may vote.

ballots. On every simple ballot, the candidates, say A and B, are printed in the same fixed order, say A is listed first and B is listed second. In the secrecy of a voting booth, the voter is supposed to fill out all three simple ballots in the following way: she marks the candidate of her choice on exactly *two* simple ballots and every other candidate on exactly *one* simple ballot. Assume, for example, that a voter votes for candidate A. Then

$$\binom{x}{o}, \binom{x}{o}, \binom{o}{x} \text{ or } \binom{x}{x}, \binom{o}{o}, \binom{x}{o}$$

would be valid multi-ballots to vote for A. After this, the voter feeds all three simple ballots to a voting machine (a scanner) and indicates the simple ballot she wants to get as a receipt. The machine checks the well-formedness of the multi-ballot, prints secretly (pairwise independent) random numbers on each simple ballot, and provides the voter with a copy of the chosen simple ballot, with the random number printed on it. Note that the voter does not get to see the random numbers of the remaining two simple ballots. The scanner keeps all simple ballots (now separated) in a ballot box.

In the tallying phase, the voting machine posts on the bulletin board (electronic copies of) all the cast simple ballots in random order. From the ballots shown on the bulletin board, the result can easily be computed: The number of votes for the ith candidate is the number of simple ballots with the ith position marked minus the total number of votes (since every voter marks every candidate at least ones).

ThreeBallot offers (some level of) individual verifiability because each voter may check whether the simple ballot she has taken as a receipt appears on the bulletin board. Thus, it should be risky for any party to remove or alter simple ballots. Additionally, ThreeBallot offers universal verifiability because the tallying is completely public. However, as Küsters et al. [KTV11] have pointed out, ThreeBallot does not offer end-to-end verifiability. One variant of the attack presented in [KTV11] assumes that the scanner is dishonest. To illustrate the attack, assume that an honest voter votes for, say, candidate A by submitting a multi-ballot of one of the forms shown above. Now, a dishonest voter which collaborates with the dishonest scanner could create a malformed ballot of the form

$$\binom{o}{x}, \binom{o}{x}, \binom{o}{x},$$

which, together with the ballot of the honest voter (no matter which one of the two kinds shown above), yields two (valid!) votes for candidate B and no vote for candidate A. Clearly, end-to-end verifiability is broken: a

vote for A and one invalid ballot result in two valid votes for B. But no honest voter would complain because none of their single/multi-ballots were manipulated. So, this attack neither invalidates individual verifiability nor universal verifiability, showing again that these notions together do not imply end-to-end verifiability, and are really insufficient.

Clash Attacks. The idea of individual and universal verifiability not only fails due to undetected malformed ballots. Another problem are *clash attacks* [KTV12b], which might break end-to-end verifiability, while individual and universal verifiability together again do not detect such attacks. As demonstrated in [KTV12b], several e-voting system are vulnerable to clash attacks, including several variants of Helios.

To illustrate the attack, consider the Helios voting system, where the voting devices might be dishonest and where the ballots of the voters are published on the bulletin board without voter names or pseudonyms attached to them. Now, if two voters vote for the same candidate, the voting devices might use the same randomness to create the ballots, and hence, the two ballots are identical. However, instead of putting both ballots on the bulletin board, authorities might add only one of them to the bulletin board and the other ballot might be replaced by one for another candidate. The two voters can check individually that "their" ballot appears on the bulletin board (individual verifiability); they do not realize that they are looking at the same ballot, i.e., they do not realize the "clash". Universal verifiability is obviously guaranteed as well. Still, the system does not provide end-to-end verifiability: a vote of an honest voter was replaced in an undetectable way by another vote.

Adding More Subproperties? Now that we have seen that individual and universal verifiability do not imply the desired security property end-to-end verifiability, it might be tempting to search for more subproperties that would then, eventually, yield a sufficiently strong verifiability notion.

In Chapter 5, we will demonstrate that all verifiability notions proposed in the literature so far that are split up into additional subproperties, such as individual and universal verifiability, do not provide end-to-end verifiability, even if more subproperties are added. In [CEK+15], for example, a subproperty was introduced that rules out clash attacks but the resulting verifiability notion is still too weak (see Section 5.8 for details).

When existing systems are analyzed w.r.t. verifiability or new systems are proposed, one should always check for end-to-end verifiability as introduced above, as end-to-end verifiability is the kind of verifiability modern e-voting systems ultimately should aim for. While subproperties, such as individual and universal verifiability, can guide the design of e-voting systems, unless

formally proven that their combination in fact implies end-to-end verifiability, such properties alone might miss important aspects and can therefore not replace end-to-end verifiability.

Not Necessary

The examples and attacks above illustrate that the notions of individual and universal verifiability are not sufficient to provide end-to-end verifiability. In Chapter 3, we demonstrate that they are not necessary to achieve end-to-end verifiability either. More specifically, in Section 3.4, we formally prove that the remote e-voting system sElect provides end-to-end verifiability (under reasonable assumptions) because it is extremely risky for an adversary to manipulate or drop even only a few votes. At the same time, sElect does not rely on universal verifiability. Jumping ahead, the Chaumian mix net employed in sElect is not verifiable by itself: it takes the voters to perform a simple check (see Section 3.2 for details). Therefore, the example of sElect shows that universal verifiability is not necessary for end-to-end verifiability.

2.4 Accountability

In e-voting systems, and for many other cryptographic tasks and protocols (e.g., secure multi-party computation, identity-based encryption, and auctions), it is extremely important that (semi-)trusted parties can be held accountable in case they misbehave. This fundamental security property is called *accountability*,[6] and it is a stronger form of verifiability: it not only allows one to verify whether a desired property is guaranteed, for example that the election outcome is correct, but it also ensures that misbehaving parties can be identified if this is not the case.

Accountability is important for several practical reasons. First of all, accountability strengthens the incentive of all parties to follow their roles because they can be singled out in case they misbehave and then might have to face, for example, severe financial or legal penalties, or might lose their reputation. Furthermore, accountability can resolve disputes that occur when it is only known that some party misbehaved but not which one. This can, for instance, help to increase the robustness of cryptographic protocols because misbehaving parties, such as a dishonest trustee in an e-voting protocol, can be excluded and the protocol can be re-run without the parties that misbehaved.

[6] In the context of secure MPC, accountability is sometimes called *identifiable abort* [IOZ14].

Unfortunately, despite its importance, accountability is often not taken into account (at least not explicitly), neither to design e-voting protocols nor to analyze their security (see, e.g., [Adi08, CCM08, RBH+10, CGGI14, CRST15, KZZ15a, KZZ15b, RRI16, CCFG16, KZZ17]).

In [KTV10b], Küsters et al. provided a general formal definition of accountability and emphasized its importance. This formal definition has since been used to analyze different e-voting protocols (Helios, sElect, Ordinos, Bingo Voting), mix nets (re-encryption and Chaumian mix nets with random partial checking), auction schemes (PRST [PRST06]), and contract signing protocols (ASW [ASW00]). These analyses brought forward several accountability issues, e.g., for different versions of Helios [KTV12b].

In what follows, we precisely define the notion of accountability as introduced in [KTV10b]. We will apply this definition to formally establish the level of accountability of sElect and Ordinos (see Sections 3.5 and 4.4).

A Formal Accountability Definition. The accountability definition by Küsters, Truderung, and Vogt [KTV10b] is based on the same generic and expressive protocol model as the verifiability definition (see Section 2.3), and can therefore be applied to all classes of voting protocols and also to other domains.

In contrast to the verifiability definition, the judge now not only accepts or rejects a run, but may output detailed verdicts. A *verdict* is a positive Boolean formula ψ built from propositions of the form $\mathsf{dis}(\mathsf{a})$, for an agent a, where $\mathsf{dis}(\mathsf{a})$ means that (the judge thinks that) agent a misbehaved, i.e., did not follow the prescribed protocol. For example, in a voting protocol with voters $\mathsf{V}_1, \ldots, \mathsf{V}_n$, a bulletin board B, and trustees $\mathsf{T}_1, \ldots, \mathsf{T}_m$, if the judge J states, say, $\mathsf{dis}(\mathsf{B}) \wedge \mathsf{dis}(\mathsf{T}_1) \wedge \ldots \wedge \mathsf{dis}(\mathsf{T}_m)$, then this expresses that the judge believes that the bulletin board and all trustees misbehaved; the judge would state $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{B}) \vee (\mathsf{dis}(\mathsf{T}_1) \wedge \ldots \wedge \mathsf{dis}(\mathsf{T}_m))$ if she is not sure whether voter V_i. the bulletin board, or all trustees misbehaved.

Who should be blamed in which situation is expressed by a set Φ of what are called *accountability constraints*. These constrains are of the form

$$C = \alpha \Rightarrow \psi_1 | \cdots | \psi_k,$$

where α is a property of the voting system, similar to the goal γ in Section 2.3.2 (a set of runs of the system, where one run is determined by the random coins used by the parties), and $\psi_1, \ldots, \psi_k$ are verdicts. Intuitively, the set α contains runs in which some desired goal γ 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 (by the fairness condition these verdicts will be true). That is, if a run belongs to α, then C

requires that in this run the judge outputs a verdict ψ which logically implies one of ψ_i.

To illustrate the notion of accountability constraints, let us continue the example from above. Let α contain all runs in which the published election result is incorrect, e.g., $\alpha = \alpha(k, \varphi) = \neg\gamma(k, \varphi)$ with the goal $\gamma(k, \varphi)$ as defined in Section 2.3. Now, consider the following constraints:

$$C_1 = \alpha \Rightarrow \mathsf{dis}(\mathsf{B})|\mathsf{dis}(\mathsf{T}_1)|\cdots|\mathsf{dis}(\mathsf{T}_m),$$
$$C_2 = \alpha \Rightarrow \mathsf{dis}(\mathsf{V}_1) \vee \cdots \vee \mathsf{dis}(\mathsf{V}_n) \vee \mathsf{dis}(\mathsf{B}) \vee (\mathsf{dis}(\mathsf{T}_1) \wedge \cdots \wedge \mathsf{dis}(\mathsf{T}_m)),$$
$$C_3 = \alpha \Rightarrow \mathsf{dis}(\mathsf{B})|\mathsf{dis}(\mathsf{T}_1) \wedge \cdots \wedge \mathsf{dis}(\mathsf{T}_m).$$

Constraint C_1 requires that if in a run the published election result is incorrect, then at least one (individual) party among $\mathsf{B}, \mathsf{T}_1, \ldots, \mathsf{T}_m$ can be held accountable by the judge J; note that different parties can be blamed in different runs. Constraint C_2 states that if the published election result is not correct, then the judge J can leave it open whether one of the voters, the bulletin board B, or all trustees misbehaved. Constraint C_3 requires that it is possible to hold B or all trustees accountable.

As pointed out in [KTV10b], accountability constraints should provide at least *individual accountability*. That is, the postulated minimal verdicts should at least single out one misbehaving party. In the above example, C_1 and C_3 provide individual accountability, but C_2 does not. In fact, C_2 is very weak, too weak for practical purposes. If a judge states exactly this verdict, there are no real consequences for any party, since no individual party can be held accountable. This is particularly problematic if in such a "fuzzy" verdict not only voting authorities are involved but also voters.

A set Φ of constraints for a protocol P is called an *accountability property* of P. Typically, an accountability property Φ covers all relevant cases in which a desired goal γ for P is not met, i.e., whenever γ is not satisfied in a given run P due to some misbehavior of some protocol participant, then there exists a constraint C in Φ which covers r. We write $\Pr(\pi^{(\ell)} \to \neg(\mathsf{J}\colon \Phi))$ to denote the probability that π, with security parameter 1^ℓ, produces a run r such that J does not satisfies all accountability constrains for this run, i.e., there exists $C = (\alpha \Rightarrow \psi_1|\cdots|\psi_k)$ with $r \in \alpha$ but the judge outputs a verdict which does not imply some ψ_i.

Definition 2 (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, we say that* the protocol P is (Φ, δ)-accountable by the judge J *if for all adversaries π_A and $\pi = (\pi_P \| \pi_\mathsf{A})$, the probability*

$$\Pr(\pi^{(\ell)} \to \neg(\mathsf{J}\colon \Phi))$$

is δ-bounded as a function of ℓ.

Just as for the verifiability definition (Definition 1), the full definition in [KTV10b] additionally requires that the judge J is fair, i.e., that she states false verdicts only with negligible probability.

Küsters et al. also showed that verifiability (as defined in Definition 1) can be considered to be a weak form of accountability, and, as mentioned before, verifiability alone is typically too weak for practical purposes.

Instead of explicitly specifying Φ as necessary in the above definition, there have been attempts to find generic ways to define who actually caused a goal to fail and ideally to blame all of these parties. There has been work pointing into this direction (see, e.g., [DGK+15, FJW11, GM15]). But this problem turns out to be very tricky and has not been solved yet.

2.5 Privacy

In Section 2.5.1, we recapitulate the game-based privacy definition by Küsters et al. [KTV11]. This definition allows us to *measure* the level of privacy a protocol provides, unlike other definitions (see, e.g., [BCG+15]). In Section 3.6 and 4.5, respectively, we use this definition to analyze the privacy level of sElect and Ordinos.

In Section 2.5.2, we present the privacy level of an ideal voting protocol, depending on different parameters (e.g., the number of voters or the specific result function). These results are immediately relevant for our privacy analyses of sElect and Ordinos as we will formally reduce their privacy levels to the one of an ideal voting protocol (see Section 3.6 and 4.5 for details).

In Section 2.5.3, we provide some background on the relationship between privacy and coercion-resistance. We will demonstrate that coercion-resistance is neither sufficient nor necessary for privacy. Since coercion-resistance is not in the scope of this thesis,[7] we discuss the relationship on a high level without introducing the formal definition of coercion-resistance.

2.5.1 Privacy Definition

Intuitively, the privacy definition in [KTV11] says that no (ppt) observer, who may control some parties, such as some authorities or voters, should be able to tell how an honest voter, the voter under observation $\mathsf{V}_{\mathsf{obs}}$, voted. More specifically, one considers two systems: in one system the voter under

[7] That is because (*i*) sElect and Ordinos have not been designed to provide coercion-resistance, and (*ii*) Chapter 5 is only concerned with verifiability.

consideration votes for choice ch_0 and in the other system the voter votes for choice ch_1; all other honest voters vote according to some probability distribution known by the observer. Now, the probability that the observer correctly says with which system he interacts should be bounded by some constant δ (plus some negligible function in the security parameter). Due to the parameter δ, the definition allows one to *measure* privacy.

As we will see in Section 3.6 and 4.5, this ability is crucial in the analysis of protocols which provide a reasonable but not perfect level of privacy. In fact, strictly speaking, most remote e-voting protocols do not provide a perfect level of privacy: this is because there is always a certain probability that voters do not check their receipts. Hence, the probability that malicious servers/authorities drop or manipulate votes without being detected is non-negligible. By dropping or manipulating votes, an adversaries obtains some non-negligible advantage in breaking privacy. Therefore, it is essential to be able to precisely tell how much an adversary can actually learn.

To define this notion formally, we first introduce the following notation. Let P be an e-voting protocol (in the sense of Section 2.2 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. Typically, 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 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 $(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}) \| \pi^* \| \pi_{\mathsf{A}})$.

Clearly, if an adversary (controlling the authentication server) drops all votes except for the one of the voter under observation, privacy can easily be broken. Therefore, the privacy 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 sElect and Ordinos, for example, only adversaries are admissible that drop/manipulate at most k votes of honest voters (see Section 3.6 for details).

Definition 3 (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 $\delta \in [0, 1]$. We say that P* achieves *δ-privacy* (w.r.t. *$\mathcal{A}$*)*, if*

$$\left| \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_0) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_1) \| \pi^* \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1] \right|$$

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* $\mathsf{ch} \in \mathsf{C}$.[8]

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 2.2, we denote this distribution by μ. Moreover, the level of privacy, also of an ideal voting protocol, will depend on the (possibly tally-hiding) result function, i.e., the information contained in the published result, as further investigated in Section 2.5.2.

2.5.2 Privacy of the Ideal Protocol

As mentioned above, we will reduce the privacy level of sElect and Ordinos to the privacy level of an ideal voting protocol. Therefore, in this section, we recapitulate the optimal privacy level for an ideal voting protocol. In [KTV11], a formula for the optimal privacy level has been derived proven for the specific "classical" result function that reveals the total number of votes per candidate. In [Ano18], these results have been generalized to arbitrary, in particular tally-hiding, result functions (e.g., only revealing the winner).

As discussed in Section 2.5.1, 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 (possibly tally-hiding) voting protocols, we now determine the optimal value of δ for the ideal (tally-hiding) voting protocol.

The ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}})$ is defined in Figure 2.1. 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

[8] That is, $\pi_\mathsf{A} \in \bigcap_{\mathsf{ch} \in \mathsf{C}} \mathcal{A}(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}) \| \pi^*)$.

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}.

$\underline{\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}})}$

Parameters:

- Function $f_{\mathsf{res}}\colon \{0,1\}^* \to \{0,1\}^*$
- Probability distribution μ over C
- Number of voters n_{voters}
- Number of honest voters $n_{\mathsf{voters}}^{\mathsf{honest}}$
- $I \leftarrow \emptyset$ (initially)

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

1. $\forall i \in \{1, \ldots, n_{\mathsf{voters}}^{\mathsf{honest}}\}$: store $\mathsf{ch}_i \leftarrow_\mu \mathsf{C}$
2. $I \leftarrow I \cup \{1, \ldots, n_{\mathsf{voters}}^{\mathsf{honest}}\}$
3. Return $\mathsf{success}$

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

1. If $i \notin \{n_{\mathsf{voters}}^{\mathsf{honest}} + 1, \ldots, n_{\mathsf{voters}}\}$, return $\perp$.
2. If $\mathsf{ch} \notin \mathsf{C}$, return $\perp$.
3. Store $\mathsf{ch}_i \leftarrow \mathsf{ch}$
4. $I \leftarrow I \cup \{i\}$
5. Return $\mathsf{success}$

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

1. If $b = 0$, return $\perp$.
2. Return $\mathsf{res} \leftarrow f_{\mathsf{res}}((\mathsf{ch}_i)_{i \in I})$

Figure 2.1: Ideal privacy functionality for voting protocol.

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 (tally-hiding) result function f_{res}

in relation to the number of voters n_{voters}, the number of honest voters $n^{\mathsf{honest}}_{\mathsf{voters}}$, and the probability distribution μ according to which the honest voters select their choices.

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 $A^{\mathsf{ch},\vec{R}}_{\mathsf{y}}$ denote the probability that the choices made by the honest voters yield the output y 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 $\mathsf{ch} \in \mathsf{C}$ and the dishonest voters vote according the choice vector $\vec{R} = (\mathsf{ch}_i)_{i \in I_{\mathsf{dishonest}}}$. (Clearly, $A^{\mathsf{ch},\vec{R}}_{\mathsf{y}}$ depends on μ. However, we omit this in the notation.) Furthermore, let $A^{\mathsf{ch}}_{\vec{r}}$ 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 ch. (Again, $A^{\mathsf{ch}}_{\vec{r}}$ 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

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

and

$$A^{\mathsf{ch}}_{\vec{r}} \;=\; \frac{n!}{r_1! \cdots r_n!} \cdot p_1^{r_1} \ldots p_n^{r_n} \cdot \frac{r_{\mathsf{ch}}}{p_{\mathsf{ch}}}$$

where $(p_1, \ldots, p_n)$ 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 $\mathsf{C} = \{1, \ldots, n\}$.

Moreover, let $M^*_{\mathsf{ch}_0,\mathsf{ch}_1,\vec{R}} = \{\mathsf{y}\colon A^{\mathsf{ch}_0,\vec{R}}_{\mathsf{y}} \leq A^{\mathsf{ch}_1,\vec{R}}_{\mathsf{y}}\}$. 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 y, wants to decide whether the observed voter voted for choice ch_0 or ch_1, the best strategy of the observer is to opt for ch_1 if $\mathsf{y} \in M^*_{\mathsf{ch}_0,\mathsf{ch}_1,\vec{R}}$, i.e., the output is more likely if the voter voted for choice ch_1. This leads to the following definition:

$$\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}}) = \max_{\mathsf{ch}_0,\mathsf{ch}_1 \in \mathsf{C}} \max_{\vec{R}} \sum_{\mathsf{y} \in M^*_{\mathsf{ch}_0,\mathsf{ch}_1,\vec{R}}} (A^{\mathsf{ch}_1,\vec{R}}_{\mathsf{y}} - A^{\mathsf{ch}_0,\vec{R}}_{\mathsf{y}}) \tag{2.1}$$

The following theorem shows that $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$ is indeed optimal (see [Ano18] for the proof of the theorem).

Theorem 1 (Ideal Privacy). *Let* $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$ *be defined as above. Then, the ideal protocol* $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu)$ *achieves* $\delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$*-privacy. Moreover, it does not achieve* δ'*-privacy for any* $\delta' < \delta^{\mathsf{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(f_{\mathsf{res}})$.

2.5.3 Relationship to Coercion-Resistance

To achieve verifiability, a voter typically obtains some kind of receipt which, together with additional data published in the election, she can use to check that her vote was counted. This, however, potentially opens up the possibility for vote buying and voter coercion. Besides verifiability, many voting systems therefore also intend to provide so-called *coercion-resistance*.

One would expect that privacy and coercion-resistance are closely related: If the level of privacy is low, i.e., there is a good chance of correctly determining how a voter voted, then this should give the coercer leverage to coerce a voter. Some works in the literature (e.g., [MN06, DKR06]) indeed suggest a close connection. However, Küsters et al. [KTV11] demonstrated that the relationship between privacy and coercion-resistance is more subtle.

Among others, it turns out that improving the level of privacy of a protocol in a natural way (e.g., by changing the way honest voters fill out ballots) can lead to a *lower* level of coercion-resistance. Clearly, in general, one does not expect privacy to imply coercion-resistance. Still, the effect is quite surprising.

A maybe even more important and unexpected finding that comes out of the case studies in [KTV11] is that the level of privacy of a protocol can be much *lower* than its level of coercion-resistance.

The reason behind this phenomenon is basically that it may happen that the counter-strategy a coerced voter may carry out to defend against coercion hides the behavior of the coerced voter, including her vote, better than the honest voting program.

On the positive side, in [KTV11] Küsters et al. proved a theorem which states that under a certain additional assumption a coercion-resistant protocol provides at least the same level of privacy. This is the case when the counter-strategy does not "outperform" the honest voting program in the above sense. The theorem is applicable to a broad class of voting protocols.

In what follows, we explain the subtle relationships between coercion-resistance and privacy in more detail. The findings are based on the formal privacy and coercion-resistance definitions originally proposed in [KTV11] and [KTV10a, KTV12a], respectively (recall Section 2.5.1 for the formal privacy definition). The definition of coercion-resistance is built upon the same general protocol model as the one for verifiability and privacy, and

hence, it is applicable to all classes of voting systems. We only informally introduce the coercion-resistance definition in what follows and point to the reader to [KTV11, KTV10a, KTV12a] for the formal definition.

For the definition of coercion-resistance, the voter under observation considered for privacy is now replaced by a *coerced voter* and the observer is replaced by the *coercer*. We imagine that the coercer demands full control over the voting interface of the coerced users, i.e., the coercer wants the coerced voter to run a dummy strategy dum which simply forwards all messages between the coerced voter and the coercer. If the coerced voter in fact runs dum, the coercer can effectively vote on behalf of the coerced voter or decide to abstain from voting. Of course, the coercer is not bound to follow the specified voting procedure.

Now, informally speaking, a protocol is called *coercion-resistant* if the coerced voter, instead of running the dummy strategy, can run some *counter-strategy* such that (i) by running this counter-strategy, the coerced voter achieves her own goal γ (formally, again a set of runs), e.g., successfully votes for a specific candidate, and (ii) the coercer is not able to distinguish whether the coerced voter followed his instructions (i.e., run dum) or tried to achieve her own goal. Similarly to the privacy definition, the probability in (ii) is bounded by some constant δ (plus some negligible function). Again, δ is important in order to be able to measure the level of coercion-resistance a protocol provides: there is always a non-negligible chance for the coercer to know for sure whether the coerced voter followed his instructions or not (e.g., when all voters voted for the same candidate).

Improving Privacy Can Lower the Level of Coercion-Resistance. To illustrate this phenomenon, we consider the following variant of ThreeBallot (for details of ThreeBallot see Section 2.3.3). An honest voter is supposed to submit, according to her favorite candidate,

$$\text{either } \binom{x}{x}, \binom{x}{o}, \binom{o}{o} \text{ or } \binom{x}{x}, \binom{o}{x}, \binom{o}{o},$$

and always take the first single ballot $\binom{x}{x}$ as her receipt. The scheme is ideal in terms of privacy because the bulletin board and the receipts do not leak any information apart from the pure election result. However, this scheme does not provide any coercion-resistance. Assume that the coerced voter is instructed to cast

$$\binom{o}{x}, \binom{x}{x}, \binom{o}{o}$$

and take the first single ballot as receipt (which is allowed but never done by honest voters). If the coerced voter actually wants to vote for candidate A,

the voter would have to cast

$$\binom{o}{x}, \binom{x}{o}, \binom{x}{o}.$$

But then, as all the honest voters submit

$$\binom{x}{x}, \binom{x}{o}, \binom{o}{o} \text{ or } \binom{x}{x}, \binom{o}{x}, \binom{o}{o},$$

the coercer could easily detect that he was cheated, by counting the number of ballots of type $\binom{o}{o}$ on the bulletin board.

Coercion-Resistance Does Not Imply Privacy. For the original variant of ThreeBallot and the simple variant of VAV, Küsters et al. proved that the level of privacy is much lower than its level of coercion-resistance. The reason behind this phenomenon is basically that the counter-strategy hides the behavior of the coerced voter, including her vote, better than the honest voting program hides the vote. In these voting systems, a receipt an honest voter obtains indeed discloses more information than necessary (for details see [KTV11]).

The following simple, but unlike ThreeBallot and VAV, artificial example, carries this effect to extremes: Consider the ideal voting protocol which collects all votes and publishes the correct result. Now, imagine a voting protocol in which voters use the ideal voting protocol to cast their vote, but where half of the voters publish how they voted (e.g., based on a coin flip). Clearly, the privacy level this protocol provides is very low, namely $\delta \geq \frac{1}{2}$. However, a coerced voter can be more clever and simply lie about how she voted. This protocol indeed provides a high level of coercion-resistance.

As mentioned at the beginning of this section, it is shown in [KTV11] that if the counter-strategy does not "outperform" the honest voting program (or conversely, the honest voting program does not leak more information than the counter-strategy), then indeed if a voting system provides a certain level of coercion-resistance, then it provides the same level of privacy. Fortunately, in most systems which are supposed to provide coercion-resistance, the counter-strategy indeed does not outperform the honest program.

Chapter 3

sElect: A Lightweight Verifiable Remote E-Voting System

The design of practical remote e-voting systems is very challenging as many aspects have to be considered. In particular, one has to find a good balance between simplicity, usability and security. This in turn very much depends on various, possibly even conflicting requirements and constraints, for example: What kind of election is targeted? National political elections or elections of much less importance and relevance, e.g., within clubs or associations? Should one expect targeted and sophisticated attacks against voter devices and/or servers, or are accidental programming errors the main threats to the integrity of the election? Is it likely that voters are coerced, and hence, should the system defend against coercion? How heterogeneous are the computing platforms of voters? Can voters be expected to have/use a second (trusted) device and/or install software? Is a simple verification procedure important, e.g., for less technically inclined voters? Should the system be easy to implement and deploy, e.g., depending on the background of the programmers? Should authorities and/or voters be able to understand (to some extent) the inner workings of the system?

Therefore, there does not seem to exist a "one size fits all" remote e-voting system. In this chapter, we are interested in systems for *low-risk elections*, such as elections within clubs and associations, rather than national elections, where—besides a reasonable level of security—simplicity and convenience are important.

The goal of this chapter is to design a particularly lightweight remote system which (still) achieves a good level of security. The system is supposed to be lightweight both from a voter's point of view and a design/complexity point of view. For example, we do not want to require the voter to install software or use a second device. Also, verification should be a very simple

procedure for a voter or should even be completely transparent to the voter.

We present a new, particularly lightweight remote e-voting system, called *sElect* (secure and simple elections), for which we perform a detailed cryptographic security analysis w.r.t. privacy of votes as well as verifiability and accountability. sElect has also been implemented as a platform independent web application.[1] The system combines several concepts, such as verification codes (see, e.g., [DLM82]) and Chaumian mix nets [Cha81], in a novel way. sElect is not meant to defend against coercion and mostly tries to defend against untrusted or malicious authorities, including inadvertent programming errors or deliberate manipulation of servers, but excluding targeted and sophisticated attacks against voters' devices.

Overview. We start with an overview and discussion of the unique features of sElect in Section 3.1. In Section 3.2, we describe how sElect works, and then, in Section 3.3, we formally model sElect in the computation model introduced in Section 2.2. In Sections 3.4 to 3.6, we analyze verifiability, accountability, and privacy of sElect, with full proofs provided in Appendix C. In Section 3.7, we describe our implementation of sElect. We close this chapter with a discussion of related work in Section 3.8.

3.1 Features and Limitations

We sketch the main characteristics of sElect, including several novel and unique features and concepts which should be beneficial also for other systems. We also provide a general discussion on sElect, including its limitations.

Fully automated verification. One of the important unique features of sElect is that it supports fully automated verification. This kind of verification is carried out by the voter's browser. It does not require any voter interaction and is triggered as soon as a voter looks at the election result. This is meant to increase verification rates and ease the user experience. As voters are typically interested in the election results, combining the (fully automated) verification process with the act of looking at the election result in fact appears to be an effective way to increase verification rates as indicated by two small mock elections we performed with sElect (see Section 3.7). In a user study carried out in [AKBW14] for various voting systems, automated verification was pointed out to be lacking in the studied systems, including, for example, Helios. It seems that our approach of automated verification should be applicable and can be very useful for other remote e-voting systems,

[1] The implementation of sElect is covered in the Ph.D. thesis by Scapin [Sca18].

such as Helios, as well. Another important aspect of the automated verification procedure of sElect is that it performs certain cryptographic checks and, if a problem is discovered, it singles out a specific misbehaving party and produces binding evidence of the misbehavior. This provides a high level of accountability and deters potentially dishonest voting authorities.

Obviously, for fully automated verification we need to assume that (most of) the VSDs can be trusted. In Section 3.7, we will describe that in our implementation of sElect a VSD consists of the voter's computing platform (hardware, operating system, browser) and the voting booth (server), where the idea is that the voter can choose a voting booth she trusts among a set of voting booths.

Voter-based verification (human verifiability). Besides fully automated verification, sElect also supports a very easy to understand manual verification procedure: a voter can check whether a verification code she has chosen herself when casting her vote appears in the election result along with her choice. This simple procedure has several obvious benefits:

- It reduces trust assumptions concerning the voter's computing platform (for fully automated verification the voter's computing platforms needs to be fully trusted). With voter-based verification the voter does not have to trust any device or party, except that she should be able to look up the *actual* election outcome on a bulletin board, in order to make sure that her vote was counted (see also below). In particular, she does not have to trust the voting booth (she chose) at all, which is one part of her VSD.

- Also voters can easily grasp the procedure and its purpose, essentially without any understanding of the rest of the system, which should help to increase user satisfaction and verification rates. In other systems, such as Helios, voters have to have trust in the system designers and cryptographic experts in the following sense: when their ballots appear on the bulletin board, then some universally verifiable tallying mechanism—which, however, a regular voter does not understand—guarantees that her vote is actually counted. Also, other systems require the voter to perform much more complex and cumbersome actions for verifiability and they typically assume a second trusted device in order to carry out the cryptographic checks, which altogether often discourages voters from performing these actions in the first place.[2]

[2]For example, Helios demands voters (*i*) to perform Benaloh challenges and (*ii*) to check whether their ballots appear on the bulletin board. However, regular voters often have difficulties understanding these verification mechanisms and their purposes, as indicated

On the negative side, verification codes could be easily misused for coercion. A voter could (be forced to) provide a coercer with her verification code *before* the election result is published, and hence, once the result is published, a coercer can see how the voter voted. A mitigation for this problem has been considered in the Selene protocol [RRI16], but this approach assumes, among others, a public-key infrastructure for all voters. We note, however, that in any case, for most practical remote e-voting systems, including sElect and, for instance, Helios, there are also other simple, although perhaps not as simple, methods for coercion. Depending on the exact deployment of these systems, a coercer might, for example, ask for the credentials of voters, and hence, simply vote in their name. Also, voters might be asked/forced to cast their votes via a (malicious) web site provided by the coercer, or the coercer asks voters to run a specific software. So, altogether preventing coercion resistance is extremely hard to achieve in practice, and even more so if, in addition, the system should still be simple and usable. This is one reason that coercion-resistance was not a design goal for sElect.

Furthermore, since sElect employs Chaumian mix nets (see Section 3.2), a single server could refuse to perform its task, and hence, block the tallying. Clearly, those servers who deny their service could be blamed, which in many practical situations should deter them from misbehaving. Therefore, for low-risk elections targeted in this work, we do not think that such a misbehavior of mix servers is a critical threat in practice. Other systems use different cryptographic constructions to avoid this problem, namely, threshold schemes for distributed decryption and (universally verifiable) reencryption mix nets.

Simple cryptography and design. Unlike other modern remote voting systems, sElect uses only the most basic cryptographic operations, namely, public key encryption and digital signatures. And, as can been seen from Section 3.2, the overall design and structure of sElect is simple as well. In particular, sElect does *not* rely on any more sophisticated cryptographic operations, such as zero-knowledge proofs, verifiable distributed decryption, universally verifiable mix nets, etc. Our motivation for this design choice is twofold. Firstly, we wanted to investigate what level of security (privacy, verifiability, and accountability) can be obtained with only the most basic

by several usability studies (see, e.g., [AKBW14, KOKV11, KKO+11, NORV14, OBV13, WH]). Therefore, many voters are not motivated to perform the verification, and even if they attempt to verify, they often fail to do so. Furthermore, the verification process, in particular the Benaloh challenge, is quite cumbersome in that the voter has to copy/paste the ballot (a long randomly looking string) to another, *then trusted*, device in which cryptographic operations need to be performed. If this is done at all, it is often done merely in a different browser window (which assumes that the voter's platform and the JavaScript in the other window is trusted), instead of a different platform.

cryptographic primitives (public-key encryption and digital signatures) and a simple and user-friendly design, see also below. Secondly, using only the most basic cryptographic primitives has several advantages:

- The implementation can use standard cryptographic libraries and does not need much expertise on the programmers side. In fact, simplicity of the design and implementation task is valuable in practice in order to avoid programming errors, as, for example, noted in [AKBW14].
- The implementation of sElect is also quite efficient (see Section 3.2).
- sElect does not rely on setup assumptions. In particular, unlike other remote voting systems, we do not need to assume common reference strings (CRSs) or random oracles.[3] We note that in [KZZ15b, KZZ15a] very complex non-remote voting systems were recently proposed to obtain security without such assumptions.
- Post-quantum cryptography could easily be used with sElect, because one could employ appropriate public key encryption schemes and signature schemes.
- In sElect, the space of voters' choices can be arbitrarily complex since, if hybrid encryption is employed, arbitrary bit strings can be used to encode voters' choices; for systems that use homomorphic tallying (such as Helios) this is typically more tricky, and requires to adjust the system (such as certain zero-knowledge proofs) to the specific requirements.

On the downside, with such a very simple design one does not achieve certain properties one can obtain with more advanced constructions. For example, sElect, unlike for instance Helios, does not provide universal verifiability (by employing, for example, verifiable distributed decryption or universally verifiable mix nets). Universal verifiability can offer more robustness as it allows one to check (typically by verifying zero-knowledge proofs) that all ballots on the bulletin board are counted correctly. Every voter still has to check, of course, that her ballot appears on the bulletin board and that it actually contains her choice (cast-as-intended and individual verifiability).

Rigorous cryptographic security analysis. We perform a rigorous cryptographic analysis of sElect w.r.t. end-to-end verifiability, accountability, and privacy. Since quite rarely implementations of practical e-voting systems

[3] We note that the underlying cryptographic primitives, i.e., the public key encryption scheme and the signature scheme, might use a random oracle, depending on the schemes employed.

come with a rigorous cryptographic analysis, this is a valuable feature by itself. Our cryptographic analysis, carried out in Sections 3.4 and 3.6, shows that sElect enjoys a good level of security, given the very basic cryptographic primitives it uses. Remarkably, the standard technique for achieving (some level of) end-to-end verifiability is to establish both so-called individual and universal verifiability.[4] In contrast, sElect demonstrates that one can achieve (a certain level of) end-to-end verifiability, as well as accountability, without universal verifiability. This is interesting from a conceptual point of view and may lead to further new applications and system designs.

Summary. Altogether, sElect is a remote e-voting system for low-risk elections which provides a new balance between simplicity, usability, and security, emphasizing simplicity and usability, and by this, presents a new option for remote e-voting. Also, some of its new features, such as fully automated verification and triggering verification when looking up the election result, could be used to improve other systems, such as Helios, and lead to further developments and system designs.

As mentioned above, we assume low-risk elections (e.g., elections in clubs and associations) where we do not expect targeted and sophisticated attacks against voters' computing platforms.[5] Also, as we will describe in Section 3.7, the idea is that several voting booth services are available, possibly provided by different organizations and independently of specific elections, among which a voter can choose one she trusts. So, for low-risk elections it is reasonable to assume that VSDs are trusted. In addition, voter-based verification provides some mitigation for dishonest VSDs (see also the discussion above and our analysis in Section 3.4).

Structure of the chapter. In Section 3.2, we describe sElect in detail on a conceptual level. Verifiability, accountability, and privacy of sElect are then analyzed in Sections 3.4 to 3.6, respectively, based on the model of sElect provided in Section 3.3. Details of our implementation of sElect are presented in Section 3.7, with a detailed discussion of related work provided in Section 3.8.

[4] As pointed out in Section 2.3.2, this combination does not guarantee end-to-end verifiability, though.

[5] For high-stake elections, such as national elections, untrusted VSD are certainly a real concern. This is in fact a highly non-trivial problem which has not been solved satisfactorily so far when both security and usability are taken into account (see, e.g., [GRCC15]).

3.2 Description

In this section, we present the sElect voting system on the conceptual level. Its implementation is described in Section 3.7.

Cryptographic primitives. sElect uses only basic cryptographic operations: public-key encryption and digital signatures. More specifically, the security of sElect is guaranteed for any IND-CCA2-secure public-key encryption scheme[6] and any EU-CMA-secure signature scheme, and hence, very standard and basic cryptographic assumptions. Typically, the public-key encryption scheme will employ hybrid encryption so that arbitrarily long messages and voter choices can be encrypted.

To simplify the protocol description, we use the following convention. Whenever we say that a party computes a signature on some message m, this implicitly means that the signature is computed on the tuple $(\mathsf{id}_{\mathsf{election}}, \mathsf{tag}, \mathsf{m})$ where $\mathsf{id}_{\mathsf{election}}$ is an election identifier (different for different elections) and tag is a tag different for signatures with different purposes (for example, a signature on a list of voters uses a different tag than a signature on a list of ballots). Similarly, every message encrypted by a protocol participant contains the election identifier.

Set of participants. The set of participants of the protocol consists of an append-only *bulletin board* B, n_{voters} *voters* $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$ and their *voter supporting devices* $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, an *authentication server* AS, n_{servers} *mix servers* $\mathsf{M}_1, \ldots, \mathsf{M}_{n_{\mathsf{servers}}}$, and a *voting authority* Auth. For sElect, a VSD is simply the voter's browser (and the computing platform the browser runs on).

We assume that there are authenticated channels from each VSD to the authentication server AS. These channels allow the authentication server to ensure that only eligible voters are able to cast their ballots. By assuming such authenticated channels, we abstract away from the exact method the VSDs use to authenticate to the authentication server; in practice, several methods can be used, such as one-time codes, passwords, or external authentication services.

We also assume that for each VSD there is one (mutual) authenticated and one anonymous channel to the bulletin board B (see below for details). Depending on the phase, the VSD can decide which channel to use in order to post information on the bulletin board B. In particular, if something went wrong, the VSD might want to complain anonymously (e.g., via a proxy) by

[6] For the privacy property of sElect, we require that the public-key encryption scheme for every public-key and any two plaintexts of the same length always yields ciphertexts of the same length. This seems to be satisfied by all practical schemes.

posting data on the bulletin board B that identifies the misbehaving party.

A protocol run consists of the following phases: the *setup phase* (where the parameters and public keys are fixed), the *voting phase* (where voters choose their candidate and let their VSDs create and submit the ballots), the *mixing phase* (where the mix servers shuffle and decrypt the election data), and the *verification phase* (where the voters verify that their ballots were counted correctly). These phases are now described in more detail.

Setup phase. In this phase, all the election parameters (the election identifier, list of candidates, list of eligible voters, opening and closing times, etc.) are fixed and posted on the bulletin board by Auth.

Every server (i.e., every mix server and the authentication server) runs the key generation algorithm of the digital signature scheme to generate its public/private (verification/signing) keys. Also, every mix server M_j runs the key generation algorithm of the encryption scheme to generate its public/private (encryption/decryption) key pair $(\mathsf{pk}_j, \mathsf{sk}_j)$. The public keys of the servers (both encryption and verification keys) are then posted on the bulletin board B; proofs of possession of the corresponding private keys are not required.

Voting phase. In this phase, every voter V_i can decide to abstain from voting or to vote for some candidate cand_i. In the latter case, the voter indicates her candidate cand_i to the VSD. In addition, for verification purposes, a *verification code* code_i is generated (see below), which the voter is supposed to write down/store. At the end of the election, the candidate/verification code pairs of all voters who cast a vote are supposed to be published so that every voter can check that her candidate/verification code pair appears in the final result, and hence, that her vote was actually counted. The verification code is a concatenation $\mathsf{code}_i = \mathsf{code}_i^{\mathsf{voter}} \| \mathsf{code}_i^{\mathsf{vsd}}$ of two nonces. The first nonce, $\mathsf{code}_i^{\mathsf{voter}}$, which we call the *voter chosen nonce*, is provided by the voter herself, who is supposed to enter it into her VSD (in our implementation, see Section 3.7, this nonce is a nine character string chosen by the voter). For verifiability, it is not necessary that these nonces are chosen uniformly at random. What matters is only that it is sufficiently unlikely that different voters choose the same nonce. The second nonce, $\mathsf{code}_i^{\mathsf{vsd}}$, is generated by the VSD itself, the *VSD generated nonce*. Now, when the verification code is determined, the VSD encrypts the voter's candidate cand_i and the verification code code_i, i.e., the candidate/verification code pair $\alpha^i_{n_{\mathsf{servers}}} = (\mathsf{cand}_i, \mathsf{code}_i)$, under the last mix server's public key $\mathsf{pk}_{n_{\mathsf{servers}}}$ using random coins $r^i_{n_{\mathsf{servers}}}$, resulting in the ciphertext

$$\alpha^i_{n_{\mathsf{servers}}-1} = \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{servers}}}, (\mathsf{cand}_i, \mathsf{code}_i); r^i_{n_{\mathsf{servers}}}).$$

Then, the VSD encrypts $\alpha^i_{n_{\mathsf{servers}}-1}$ under $\mathsf{pk}_{n_{\mathsf{servers}}-1}$ using the random coins $r^i_{n_{\mathsf{servers}}-1}$, resulting in the ciphertext

$$\alpha^i_{n_{\mathsf{servers}}-2} = \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{servers}}-1}, (\mathsf{cand}_i, \mathsf{code}_i); r^i_{n_{\mathsf{servers}}-1}),$$

and so on. In the last step, it obtains

$$\alpha^i_0 = \mathsf{Enc}(\mathsf{pk}_1, \ldots \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{servers}}}, (\mathsf{cand}_i, \mathsf{code}_i); r^i_{n_{\mathsf{servers}}}) \ldots ; r^i_1).$$

The VSD submits α^i_0 as V_i's ballot to the authentication server AS on an authenticated channel. If the authentication server receives a ballot in the correct format (i.e., the ballot is tagged with the correct election identifier), then AS responds with an acknowledgement consisting of a signature on the ballot α^i_0; otherwise, it does not output anything. If the voter/VSD tried to re-vote and AS already sent out an acknowledgement, then AS returns the old acknowledgement only and does not take into account the new vote.

If a VSD does not receive a correct acknowledgement from the authentication server AS, the VSD tries to re-vote, and, if this does not succeed, it files a complaint on the bulletin board using the authenticated channel. If such a complaint is posted, it is in general impossible to resolve the dispute and decide who is to be blamed: AS who might not have replied as expected (but claims, for instance, that the ballot was not cast) or the VSD 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 VA to resolve the problem.

When the voting phase is over, AS publishes two lists on the bulletin board, both in lexicographic order and without duplicates and both signed by the authenticated server: the list $\vec{\mathsf{b}}$ containing all the cast valid ballots and the list $\vec{\mathsf{id}}$ containing the identifiers of all voters who cast a valid ballot. It is expected that the list $\vec{\mathsf{id}}$ is at least as long as $\vec{\mathsf{b}}$ (otherwise AS will be blamed for misbehavior).

Mixing phase. The list of ciphertexts $\vec{\mathsf{c}}_0 = \vec{\mathsf{b}}$ posted by the authentication server is the input to the first mix server M_1, which processes $\vec{\mathsf{c}}_0$, as described below, and posts its signed output $\vec{\mathsf{c}}_1$ on the bulletin board. This output is the input to the next mix server M_1, and so on. We will denote the input to the j-th mix server by M_{j-1} and its output by M_j. The output $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ of the last mix server $\mathsf{M}_{n_{\mathsf{servers}}}$ is the output of the mixing stage and, at the same time, the output of the election. It is supposed to contain the plaintexts $(\mathsf{cand}_1, \mathsf{code}_1), \ldots, (\mathsf{cand}_{n_{\mathsf{voters}}}, \mathsf{code}_{n_{\mathsf{voters}}})$ (containing voters' candidates along with their verification codes) in lexicographic order.

The steps taken by a mix server M_j are as follows:

1. *Input validation.* M_j checks whether $\vec{\mathsf{c}}_{j-1}$ has the correct format, is correctly signed, arranged in lexicographic order, and does not contain any duplicates. If this is not the case, it sends a complaint to the bulletin board and stops its process (this in fact aborts the whole election process and the previous server is blamed for misbehaving). Otherwise, M_j continues with the second step.

2. *Processing.* M_j decrypts all entries of $\vec{\mathsf{c}}_{j-1}$ under its private key sk_j, removes duplicates, and orders the result lexicographically. If an entry in $\vec{\mathsf{c}}_{j-1}$ cannot be decrypted or is decrypted to a message in an unexpected format, then this entry is discarded and not further processed. The sequence of messages obtained in such a way is then signed by M_j and posted on the bulletin board as the output $\vec{\mathsf{c}}_j$.

Verification phase. After the final result $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ has been published on the bulletin board B, the verification phase starts. As mentioned in the introduction, a unique feature of sElect is that it supports the following two forms of verification, explained next: *(pure) voter-based verification*, and hence human verifiability, and *(fully automated) VSD-based verification*.

The first form is carried out by the voter herself and does not require any other party or any device, and in particular, it does not require any trust in any other party or device, except that the voter needs to be able to see the published result on the bulletin board. As we will see below, the verification procedure is very simple. As proven in Section 3.4, voter-based verification ensures verifiability even in the threat scenario that all VSDs are corrupted.

VSD-based verification is carried out fully automatically by the voter's VSD and triggered automatically as soon as the voter takes a look at the final result, as further explained in Section 3.7. It does not need any input from the voter. This is supposed to result in high verification rates and further ease the user experience, as verification is performed seamlessly from the voter's point of view and triggered automatically. Under the assumption that VSDs are honest, it yields verifiability, and even a high-level of accountability (see Section 3.5).

We now describe how these two forms of verification work in detail.

Voter-based verification. For voter-based verification, the voter simply checks whether her verification code, which in particular includes the voter chosen nonce $\mathsf{code}_i^{\mathsf{voter}}$, appears next to her candidate in the final result list.

If this is the case, the voter would be convinced that her vote was counted (see also Section 3.4). A voter V_i who decided to abstain from voting may check the list $\vec{\mathsf{id}}$ to make sure that her name (identifier) is not listed there.[7]

[7] Variants of the protocol are conceivable where a voter signs her ballot and the authen-

When checks fail, the voter would file a complaint.

VSD-based verification. For VSD-based verification, the voter's VSD performs the verification process fully automatically. In particular, this does not require any action or input from the user. In our implementation, as further explained in Section 3.7, the VSD-based verification process is triggered automatically whenever the voter goes to see the election result. Clearly, this kind of verification provides security guarantees only if the VSD is honest, and hence, for this kind of verification, the voter needs to trust her device. Making use of the information available to the VSD, the VSD can provide evidence if servers misbehaved, which can then be used to rightfully blame misbehaving parties. The VSD-based verification process works as follows. VSD_i checks whether the originally submitted plaintext $(\mathsf{cand}_i, \mathsf{code}_i)$ appears in $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$. If this is not the case, the VSD determines the misbehaving party, as described below. Recall that a VSD which did not obtain a valid acknowledgment from the authenticating server was supposed to file a complaint already in the voting phase. The following procedure is carried out by VSD_i which obtained such an acknowledgement and cannot find the plaintext $(\mathsf{cand}_i, \mathsf{code}_i)$ in $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$. First, VSD_i checks whether the ballot α_0^i is listed in the published result $\vec{\mathsf{c}}_0$ of the authentication server AS. If this is not the case, VSD_i anonymously publishes the acknowledgement obtained from AS on the bulletin board B which proves that AS misbehaved (recall that such an acknowledgement contains a signature of AS on the ballot α_0^i). Otherwise, i.e., if α_0^i is in $\vec{\mathsf{c}}_0$, the VSD checks whether α_1^i is listed in the published result $\vec{\mathsf{c}}_1$ of the first mix server M_1. If $\vec{\mathsf{c}}_1$ contains α_1^i, VSD_i checks whether α_2^i can be found in the published result $\vec{\mathsf{c}}_2$ of the second mix server M_2, and so on. As soon as VSD_i gets to the first mix server M_j which published a result $\vec{\mathsf{c}}_j$ that does not contain α_j^i (such a mix server has to exist), the VSD anonymously sends (j, α_j^i, r_j^i) to the bulletin board B. This triple demonstrates that M_j misbehaved: the encryption of α_j^i under pk_j with randomness r_j^i yields α_{j-1}^i, and hence, since α_{j-1}^i is in the input to M_j, α_j^i should have been in M_j's output, which, however, is not the case. The reason that an anonymous channel is necessary to submit the triple is the fact that it might reveal how the voter voted, for example, if M_j is the last mix server and thus α_j^i contains the voter's candidate as a plaintext. In practice, the voter could, for example, use a trusted proxy server, the Tor network, or some anonymous e-mail service.

tication server presents such a signature in case of a dispute. This solution is conceptually simple. On the pragmatic side, however, it is not always reasonable to expect that voters maintain keys and, therefore, here we consider the simpler variant without signatures. Note that this design choice was also made in several existing and prominent systems, such as Helios.

We say that a voter V_i *accepts* the result of an election if neither the voter V_i nor VSD_i output a complaint. Otherwise, we say that V_i *rejects* the result.

Remark 2. *Note that the procedures for ballot casting and mixing are very simple. In particular, a mix server needs to carry out only n_{voters} decryptions. Using standard hybrid encryption based on RSA and AES, it amounts to n_{voters} RSA decryption steps (n_{voters} modular exponentiations) and n_{voters} AES decryptions. This means that the mixing step is very efficient and the system is practical even for very big elections: mixing 100000 ballots takes about 3 minutes and mixing one million ballots takes less than 30 minutes with 2048-bit RSA keys on a standard computer/laptop.*

Remark 3. *Observe that the VSD-based verification technique cannot be applied to a re-encryption mix net because the VSD would not know the trace of its input ciphertext through the re-encryption mix net. Therefore, this technique can only be applied to Chaumian/decryption mix nets, as described above.*

3.3 Formal Protocol Model

The sElect system can be modeled in a straightforward way as a protocol $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{vsd}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ in the above sense, as detailed next.

- By n_{voters} we denote the number of voters and their voter supporting devices, and by n_{servers} the number of mix servers.
- By μ we denote a probability distribution on the set of choices, including abstention. An honest voter makes her choice according to this distribution.[8] Note that the set of valid choices is implicitly given by μ. The set of valid choices C in sElect is the product space of the set of candidates and the set of possible voter verification codes, including abstention. We assume that the choices are represented by messages of the same length. The choice provided by the voter to her VSD is called the *actual choice* of the voter.
- By $p^{\mathsf{vote}}_{\mathsf{verif}} \in [0, 1]$ we denote the probability that an honest voter who does not abstain from voting verifies the result, i.e., performs the voter-based verification procedure.

[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.

- By $p^{\mathsf{vsd}}_{\mathsf{verif}} \in [0,1]$ we denote the probability that an honest VSD of a voter who does not abstain from voting is triggered to verify the result.
- By $p^{\mathsf{abst}}_{\mathsf{verif}} \in [0,1]$ we denote the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $\vec{\mathsf{id}}$ output by the authentication server.
- By f_{sElect} we denote the result function of sElect which takes as input a list of valid choices, i.e., pairs of the form $(\mathsf{cand}, \mathsf{code}^{\mathsf{voter}})$ or abstain, and outputs the same list in lexicographic order.

The set of agents of P_{sElect} consists of all agents described in Section 3.2, i.e., the bulletin board B, the voters $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$ and their VSDs $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, the authentication servers AS, the mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_{\mathsf{servers}}}$, and in addition, a scheduler S. The latter party will play the role of the voting authority Auth and schedule all other agents in a run according to the protocol phases. Also, it will be the master program in every instance of P_{sElect}. All agents are connected via channels with all other agents; honest agents will not use all of these channels, but dishonest agents might. The honest programs of all agents are defined in the obvious way according to the description of the agents in Section 3.2 (see below for details). We assume that the scheduler and the bulletin board are honest. All other agents can possibly be dishonest and run any probabilistic polynomial-time program. We note that the scheduler is only a modeling tool. It does not exist in real systems. The assumption that the bulletin board is honest is common; Helios makes this assumption too, for example (see also Section 3.1). In addition to the participants listed above, we also have a judge J in order to model/analyze verifiability and accountability of sElect (recall Section 2.3 for details on verifiability).

Next, we describe the honest program of every agent a in P_{sElect}.

Scheduler S. In every instance of P_{sElect}, the scheduler S plays the role of the master program (in the sense of Section 2.2). We assume that it is given information about which agents are honest and which are dishonest in order to be able to schedule the agents in the appropriate way. In what follows, we implicitly assume that the scheduler triggers the adversary (any dishonest party) at the beginning of the protocol run and at the end of this run. Also, the adversary is triggered each time an honest party finishes its computations (after being triggered by the scheduler in some protocol step). This keeps the adversary up to date and allows it to output its decision at the end of the run. By this, we obtain stronger security guarantees. Similarly, we assume that the judge is triggered each time any other party (honest or dishonest) finishes its computation (after being triggered by the scheduler).

This gives the judge the chance to output its verdict after each protocol step. If the judge posts a message on the bulletin board B which indicates to stop the whole protocol (see Section 3.5), then the scheduler triggers once more the adversary (to allow it to output its decision) and then halts the whole system. This means that no participants are further triggered.

In the remaining part of the section, we precisely describe the honest program of the scheduler depending on the voting phase.

Scheduling the setup phase. At the beginning of the election, the scheduler generates a random number id, the election identifier, with the length of the security parameter ℓ and sends it to the bulletin board B which publishes id. After that, the scheduler first triggers all the honest servers, which are supposed to generate their signing/verification key pairs and publish the public (verification) keys on the bulletin board B, and then all the dishonest ones. The analogous process is carried out for generation and publishing of encryption keys.

Scheduling the voting phase. The scheduler first triggers all the honest voters and then the dishonest ones, allowing them to cast their ballots to the authentication server AS using their VSDs. After each such step (when the computations of a voter, her VSD and the authentication server are finished), the scheduler triggers the VSD again, to allow the VSD to post a complaint, if it does not get a valid acknowledgment from the authentication server. Recall that the authentication server AS is modeled in such a way that it provides all collected ballots (even before AS publishes them on the bulletin board B) to an arbitrary participant who requests these ballots. Afterwards, the scheduler triggers the authentication server which is supposed to publish the lists $\vec{\mathsf{id}}$ (containing the names of those eligible voters who cast a valid ballot) and the list $\vec{\mathsf{c}}_0$ (containing the (first) valid ballot cast by each eligible voter) on the bulletin board B.

Scheduling the mixing phase. In this phase, the scheduler triggers all the mix servers, from M_1 to $\mathsf{M}_{n_{\mathsf{servers}}}$ (recall that the judge and the adversary are triggered after each such step).

Scheduling the verification phase. Similar to the voting phase, the scheduler triggers first the honest voters and their VSDs who are supposed to verify the result. Recall that, if a voter abstained, she is supposed to verify with probability $p^{\mathsf{abst}}_{\mathsf{verif}}$ whether her name appears in the list $\vec{\mathsf{id}}$, and, if this is not the case, to file a complaint as described in the description (Section 3.2). If the voter did not abstain, she and her VSD are supposed to verify with probability $p^{\mathsf{vote}}_{\mathsf{verif}}$ and $p^{\mathsf{vsd}}_{\mathsf{verif}}$, respectively, whether the voter's submitted choice appears in the final result $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$, and, if this is not the case, to file a complaint as described in the description, (Section 3.2). Afterwards,

the scheduler triggers all the dishonest voters.

Bulletin board B. Running its honest program, the bulletin board B accepts messages from all agents. If the bulletin board B receives a message via an authenticated channel, it stores the message in a list along with the identifier of the agent who posted the message. Otherwise, if the message is sent anonymously, it only stores the message. On request, the bulletin board sends its stored content to the requesting agent.

Voter V_i. A voter V_i, when triggered by the scheduler in the voting phase, picks a candidate-code pair $(\mathsf{cand}_i, \mathsf{code}_i^{\mathsf{voter}})$ according to the probability distribution μ. A choice may be either a distinct value $\mathsf{abstain}$, which expresses abstention from voting, or a real choice. If $\mathsf{ch}_i = \mathsf{abstain}$, then the voter program stops. Otherwise, if ch_i is a real choice, the program sends $\mathsf{ch}_i = (\mathsf{cand}_i, \mathsf{code}_i^{\mathsf{voter}})$ to her voter supporting device VSD_i. The voter V_i, when triggered by the scheduler in the verification phase, carries out the following steps, depending on whether her choice ch_i was $\mathsf{abstain}$ or not. If ch_i was $\mathsf{abstain}$, the voter, with probability $p_{\mathsf{verif}}^{\mathsf{abst}}$, verifies that her name is not listed in the list $\vec{\mathsf{id}}$ of names output by the authentication server. She files a complaint if this is not the case, as described in Section 3.2. If $\mathsf{ch}_i \neq \mathsf{abstain}$, the voter, with probability $p_{\mathsf{verif}}^{\mathsf{vote}}$, follows the verification procedure to check that her candidate/verification code-pair is listed in the final result. If this is not the case, she files a complaint as described in Section 3.2.

Voter supporting device VSD_i. When the voter supporting device VSD_i receives a tuple $\mathsf{ch} = (\mathsf{cand}, \mathsf{code}^{\mathsf{voter}})$ by V_i, it produces and casts a ballot as described in Section 3.2. The voter supporting device expects to get back an acknowledgement (a signature of AS on the submitted ballot). When this happens, the voter supporting device verifies the acknowledgement. If the acknowledgement is incorrect, the voter supporting device posts a complaint on the bulletin board via her authenticated channel. Note that the program of the voter supporting device may not get any response from AS in case AS is dishonest. To enable the voter supporting device in this case to post a complaint on the bulletin board, the scheduler triggers the voter supporting device again (still in the voting phase). The voter supporting device VSD_i, when triggered by the scheduler in the verification phase, carries out the following steps. If it did not receive an input by V_i in the voting phase, its program stops. Otherwise, the voter supporting device, with probability $p_{\mathsf{verif}}^{\mathsf{vsd}}$, follows the verification procedure to check that V_i's candidate/verification code-pair is listed in the final result. If this is not the case, it files a complaint as described in Section 3.2.

Authentication server AS. The honest authentication server AS carries out the steps described in Section 3.2, with one additional step: when AS

is asked for the ballots of the voters, AS provides all ballots collected so far to the requester (even before AS published them on B). This models the assumption that the channel from the voter to AS is authenticated, but does not necessarily provide secrecy.

Mix server M_j. The honest program of M_j carries out the procedure described in Section 3.2.

Judge J. The honest program of J carries out the following procedure. We note that this program, as defined below, uses only the publicly available information, and therefore every party, including the voters as well as external observers, can run the judging procedure. The judge J, whenever triggered by the scheduler, reads data from the bulletin board and verifies its correctness, including correctness of posted complaints. The judge outputs its verdicts (as described below) on a distinct tape. More precisely, the judge outputs verdicts in the following situations:

(J1) If a server S does not publish data when expected or the published data is not in the expected format, this server is blamed by the judge, i.e., the judge outputs the verdict $\mathsf{dis}(S)$, and the whole election process is halted.

(J2) If a voter V_i posts an authenticated complaint in the voting phase that the authentication server has not responded with a valid acknowledgement, then the judge outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$, which means that (the judge believes that) either V_i or 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, the judge outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

(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 α, while α is not listed in the output $\vec{\mathsf{c}}_0$ of AS, then the judge blames AS outputting the verdict $\mathsf{dis}(\mathsf{AS})$.

(J5) If, in the verification phase, a valid complaint of the form (j, α, r) is (anonymously) posted, i.e., $\mathsf{Enc}^r_{\mathsf{pk}_j}(\alpha)$ is in $\vec{\mathsf{c}}_{j-1}$, but α is not in $\vec{\mathsf{c}}_j$, then the judge blames M_j outputting the verdict $\mathsf{dis}(\mathsf{M}_j)$.

3.4 Verifiability

In this section, we formally establish the level of verifiability provided by sElect. We show that sElect enjoys a good level of verifiability based on the generic definition of end-to-end verifiability presented in Section 2.3.2.[9] Importantly, verifiability is ensured without having to trust any of the VSDs, mix servers, or voting authorities. Verifiability is provided by the simple voter-based verification mechanism (human verifiability), and the only assumption we have to make is that each voter has access to the final result in order to check whether her voter-generated verification code appears next to her chosen candidate (see also the discussion in Section 3.1).

The verifiability level of sElect depends on whether or not *clashes* occur, i.e., whether two or more honest voters chose the same pair $(\mathsf{cand}, \mathsf{code}^{\mathsf{voter}})$. We denote the probability of having at least one clash by p_{clash} and define $p_{\mathsf{noclash}} = 1 - p_{\mathsf{clash}}$. Under certain conditions, clashes allow collaborating malicious participants, such as the VSDs or the servers, to drop the vote of one of the affected honest voters and replace it by a different vote without being detected: If two honest voters happened to choose the same voter chosen nonce and vote for the same candidate and the VSDs of both voters are malicious, the adversary (controlling both VSDs) could inject another vote by making sure that the two honest voters obtain the same choice/verification code pairs. The adversary can then just output one such pair in the final result list, and hence, he could possibly inject another choice/verification code. Such attacks are called *clash attacks* [KTV12b].

We now state the verifiability level provided by sElect. Recall that $p^{\mathsf{vote}}_{\mathsf{verif}}$ denotes the probability that an honest voter who does not abstain from voting verifies the final result, and that $p^{\mathsf{abst}}_{\mathsf{verif}}$ denotes the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $\vec{\mathsf{id}}$ output by the authentication server.

As mentioned above, we use the generic end-to-end verifiability definition presented in Section 2.3.2 which is an instantiation of the generic verifiability definition (Definition 1) with the goal $\gamma(k, \varphi)$. Recall that, roughly speaking, the goal $\gamma(k, \varphi)$ is the set of protocol runs in which the distance between the "real" input (where choices of honest voters are possibly manipulated by the adversary) and the "ideal" input to the voting protocol is bounded by k under the assumption that φ holds true. More precisely, the distance increases by 1 if an honest voter's choice is dropped and by 2 if it is changed to a different choice (excluding abstain). In the case of sElect, we merely have to assume

[9] In [KMST16a], verifiability of sElect has been analyzed w.r.t. the goal γ_k (Section 5.2), while in this thesis, we use the refined goal $\gamma(k, \varphi)$ (Section 2.3.2) to analyze verifiability. For the reasons, see Section 5.2.3.

that the abstract modelling tools, i.e., the scheduler S and the judge J, are honest and that the bulletin board provides the correct election result to the voters (see Section 3.1 for a discussion on the trusted bulletin board); formally $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{B}) \wedge \mathsf{hon}(\mathsf{J})$.

Theorem 2 (Verifiability)**.** *Let* $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{B}) \wedge \mathsf{hon}(\mathsf{J})$ *be the trust assumptions. Then, the protocol* $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{vsd}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ *provides* δ_{sElect}*-verifiability w.r.t. the goal* $\gamma(k, \varphi)$*, where*

$$\delta_{\mathsf{sElect}} = p_{\mathsf{clash}} + p_{\mathsf{noclash}} \cdot \left(\max_{k_1+k_2+2\cdot k_3 \geq k+1} (1 - p^{\mathsf{abst}}_{\mathsf{verif}})^{k_1} (1 - p^{\mathsf{vote}}_{\mathsf{verif}})^{k_2+k_3} \right).$$

The theorem says that the probability that the distance between the "real" and the "ideal" voter input is more than k, but still no voter complains, and hence, the judge rejects the run, is bounded by δ_{sElect}.

The formal proof of Theorem 2 is provided in Appendix C.1. The intuition behind the definition of δ_{sElect} is as follows. If there are no clashes in a run, then the adversary can manipulate a vote of an honest voter only if this voter does not verify the final result. Recall that dropping or stuffing a vote increases the distance between the "real" and the "ideal" voter input by 1, while changing it from a candidate to a different candidate increases it by 2. The integer k_1 denotes the number of "stuffed" votes, k_2 denotes the number of "dropped" votes, and k_3 denotes the number of changed votes. So, assuming φ holds true and no clashes occur, in order to increase this distance by at least k, and hence, violate $\gamma(k, \varphi)$, at least k_1 abstaining and $k_2 + k_3$ non-abstaining voters such that $k_1 + k_2 + 2 \cdot k_3 \geq k + 1$ should not check the final result. The probability for this very quickly approaches 0 when k grows.

The other case is that a clash occurs. We note that the occurrence of a clash does not necessarily mean that the adversary can manipulate more than k votes. For this, there have to be sufficiently many clashes, and voters within a cluster of clashes have to vote for the same candidate. Also, the VSDs of all of these voters have to be dishonest since the probability for clashes among codes generated by honest VSDs is negligible. So, δ_{sElect} as stated in the theorem is not optimal and certainly smaller in practice, and hence, the actual level of verifiability offered by sElect is better than what is stated in the theorem. On the downside, the known results on user-generated passwords (see, e.g., [Bon12, BPA12]) suggest that the quality of "randomness" provided by users may be very weak. However, it remains to be determined in a systematic and sufficiently large user study how likely clashes are for voter-chosen verification codes.

3.5 Accountability

While verifiability requires that manipulation can be detected, accountability in addition requires that misbehaving parties can be blamed (recall Section 2.4).

As already described, sElect employs two-factor verification: voter-based verification/human verifiability and VSD-based verification. The verifiability result stated above says that the voters, using only the former kind of verification, i.e., voter-based verification, and without having to trust any component of the voting system, including their own devices (except that they need to be able to see the election result on the bulletin board), can check that their votes have been counted correctly. Since human voters are only asked to keep their verification codes but not the ciphertexts and the random coins used to encrypt the choice-code pairs, they do not hold enough information to single out possibly misbehaving parties and to prove the misbehavior of a specific participant to the judge. If such a dispute occurs, the judge cannot tell whether a voter makes false claims or some servers actually misbehaved, and hence, the judge cannot resolve the dispute.

Under the assumption that VSDs (of honest voters) are honest, we show, however, that with VSD-based verification sElect provides strong accountability. For this, we use the general definition of accountability described in Section 2.4, which we instantiate for sElect.

On a high level, our accountability result for sElect says that once an honest voter (VSD) has successfully cast a ballot and obtained a signed acknowledgement from the authentication server, then in case of manipulation of the ballot, and in particular, in case the voter's vote is not counted for whatever reason, the VSD, when triggered in the verification phase, can always produce valid evidence to (rightly) blame the misbehaving party.

Unification of voter and VSD. Since the (human) voter does not play a role for accountability, we will identify each voter with her voter supporting device and simply call the unification of these two participants the *voter*. As before, we denote the probability that an honest voter who does not abstain from voting verifies the result by $p^{\mathsf{vote}}_{\mathsf{verif}} \in [0,1]$, and the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $\vec{\mathsf{id}}$ output by the authentication server by $p^{\mathsf{abst}}_{\mathsf{verif}} \in [0,1]$.

In this model, we will state the level of accountability provided by sElect for the goal $\gamma(k,\varphi)$ that we have already applied to state the verifiability result for sElect (Theorem 2). In particular, the trust assumption φ remains the same: we merely have to assume that the scheduler, the judge, and the bulletin board are honest, i.e., $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$. To state the accountability result, we first have to define an accountability property Φ_k

which covers the goal $\gamma(k, \varphi)$.

Accountability property. As defined in Section 2.4, an accountability property is a collection of accountability constraints which describe who should be blamed in which situation. For sElect, we define the accountability constraints along the judging procedure that we have formally modeled in Section 3.3.

Let χ_i contain all runs of P_{sElect} where (J2) occurs, i.e., the voter V_i complains that she did not get a receipt from AS. Similarly, let χ'_i contain all runs of P_{sElect} where (J3) occurs, i.e., the voter V_i complains that she did not vote, but her identifier is listed in $\vec{\mathsf{id}}$ published by AS. Let $\chi = \bigcup_{i \in \{1,\ldots,n_{\mathsf{voters}}\}} \chi_i \cup \chi'_i$.

Now, we define Φ_k as the accountability property consisting of the following constraints (for $i \in \{1, \ldots, n_{\mathsf{voters}}\}$):

$$\begin{aligned} \chi_i &\Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS}) \\ \chi'_i &\Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS}) \\ \neg\gamma(k, \varphi) \wedge \neg\chi &\Rightarrow \mathsf{dis}(\mathsf{AS}) \mid \mathsf{dis}(\mathsf{M}_1) \mid \ldots \mid \mathsf{dis}(\mathsf{M}_{n_{\mathsf{servers}}}) \end{aligned}$$

Clearly, this accountability property covers $\neg\gamma(k, \varphi)$ by construction, i.e., if $\gamma(k, \varphi)$ is not satisfied, these constraints require that the judge has to blame some party.

Note also that, in the runs covered by the last constraint, all the verdicts are atomic. This means that Φ_k requires that except for the cases where χ holds, whenever the goal $\gamma(k, \varphi)$ is violated, an individual party is blamed. Conversely, if χ_i occurs, the judge cannot be sure whether AS or a voter V_i misbehaved. As already discussed in Section 3.1, this is a very general problem, which applies to virtually any remote voting protocol but for which there are pragmatic solutions. The case χ'_i is a common problem as well. This could be solved, for example, when voters have public/private keys. Then they could be required to sign their ballots, and hence, AS would have proof that a voter voted.

Accountability result. We are now able to precisely state and prove the accountability level of sElect. Recall that $p^{\mathsf{vote}}_{\mathsf{verif}} \in [0, 1]$ is the probability that an honest voter who does not abstain from voting verifies the result, and $p^{\mathsf{abst}}_{\mathsf{verif}} \in [0, 1]$ is the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $\vec{\mathsf{id}}$ output by the authentication server.

For the primitives used by sElect (see Section 3.2) we assume the following: the public-key encryption scheme should be correct; for verifiability and accountability, IND-CCA2-security is not needed. Also, the signature scheme $\mathcal{S}$ should be EUF-CMA-secure (see Appendix A.2).

Theorem 3 (Accountability)**.** *Under the assumptions stated above and the mentioned judging procedure run by the judge* J*, the protocol* $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ *provides* $(\Phi_k, \delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}))$*-accountability for* J*, where*

$$\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}) = \max_{k_1+k_2+2\cdot k_3 \geq k+1} \left((1-p^{\mathsf{abst}}_{\mathsf{verif}})^{k_1}(1-p^{\mathsf{vote}}_{\mathsf{verif}})^{k_2+k_3}\right).$$

Proof. See Appendix C.1. □

This theorem means that the probability that the distance between the "real" and the "ideal" voter input is more than k, but the judge nevertheless did not blame any party, is at most $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$. If, as discussed above, voters would sign their ballots, then one could get rid of $p^{\mathsf{abst}}_{\mathsf{verif}}$ in the definition of $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$, as in this case one could require AS to provide a signed ballot for every voter listed in $\vec{\mathsf{id}}$. In practice, the problem is that voters often do not have signing keys. This is why, for example, in Helios too such signatures are not required. So, if it cannot be proven that voters voted, the above accountability result really is the best one can hope for in general: if voters do not check the published data (with their receipt), manipulation might go undetected with the stated probability.

Note that the accountability level of sElect (Theorem 3) is a special case of its more general verifiability level (Theorem 2) for $p_{\mathsf{clash}} = 0$. That is because for the verifiability result, we did not assume that the VSD of an honest voter is honest as well. So, the probability that a clash occurs between the verification codes of two human voters (who vote for the same candidate) is typically not negligible as is the case for the verification codes of two honest VSD. Hence, while the accountability level of sElect is better than its verifiability level, we have to make a somewhat stronger trust assumption.

3.6 Privacy

In this section, we carry out a rigorous privacy analysis of sElect. We show that sElect has a high level of privacy for the class of adversaries which are not willing to take a high risk of being caught cheating. This level is in fact very close to ideal when measuring privacy of single voters.

In what follows, we first introduce the class of adversaries we consider. We then state the privacy result for sElect.

3.6.1 Risk-avoiding Adversaries

The privacy definition we use (Definition 3) 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 in sElect. Assuming that the voters' VSDs are honest (which we need to assume for privacy anyway, see further below), the probability of being caught grows exponentially in the number k of honest votes that are dropped or changed (see Section 3.5). 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*.[10][11]

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) accountability results obtained in Section 3.5, we know that whenever an adversary decides to break $\gamma(k, \varphi)$ his risk of being caught is at least δ_{sElect}: 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 δ_{sElect}. That is, such an adversary knows upfront that he will be caught with a probability of at least δ_{sElect} which converges exponentially fast to 1 in k (assuming that the VSDs are honest, which we need to assume for privacy anyway). Therefore, an adversary not willing to take a risk of being caught higher than δ_{sElect} 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 (w.r.t. φ)* 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 (w.r.t. φ) over the set of all runs of π.

[10] In [KMST16a], such adversaries are called k-semi-honest. However, this term is misleading since these adversaries do not have to follow the protocol.

[11] In the field of secure MPC (see, e.g, [AL07]), such adversaries are sometimes called *covert adversaries*.

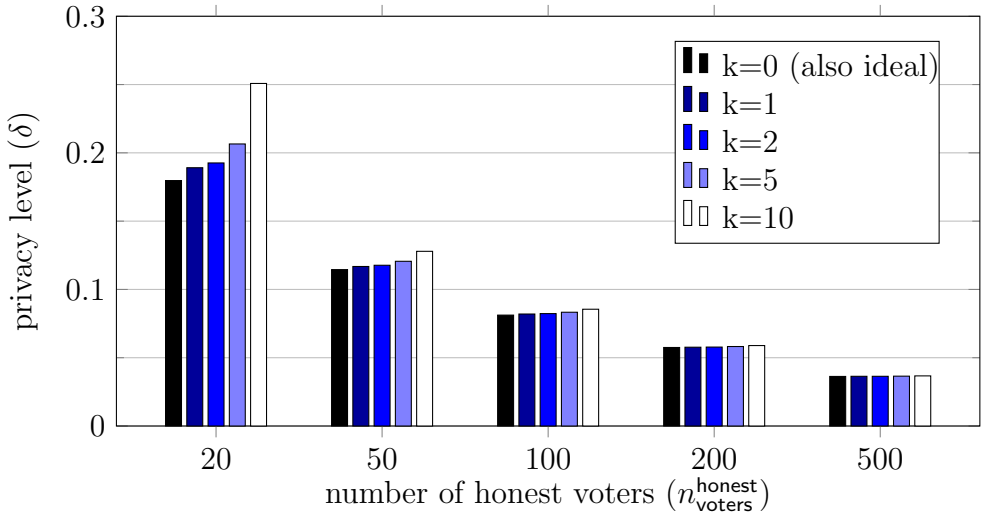

Figure 3.1: Privacy level for sElect with k-risk-avoiding adversary, for different number of honest voters $n_{\mathsf{voters}}^{\mathsf{honest}}$ and different k. The honest voters vote for two candidates, with probabilities 0.4 and 0.6. Note that the case $k = 0$ also equals the ideal case.

3.6.2 Analysis

We now prove that sElect provides a high level of privacy w.r.t. k-risk-avoiding adversaries and in the case that at most $n_{\mathsf{servers}} - 1$ mix servers are dishonest. Clearly, if n_{servers} servers were dishonest, privacy cannot be guaranteed because an adversary could then trace all ballots through the mix net. Obviously, we also need to assume that the VSD of each honest voter is honest since the device receives the chosen candidate and the verification code of the voter in plaintext. In our formal analysis of privacy below, we therefore consider the voter and the VSD to be one entity. By "high level of privacy" we mean that sElect provides δ-privacy for a δ that is very close to the ideal one.

More specifically, the privacy result for sElect is formulated w.r.t. the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{sElect}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ (Section 2.5.2). 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_{sElect}.

The level of privacy clearly depends on the number of cast ballots by honest voters. In our analysis, to have a guaranteed number of honest voters casting their ballots, we therefore in what follows assume that honest voters

do not abstain from voting. Note that the adversary would know anyway which voters abstained and which did not. Also abstaining voters can be simulated as dishonest voters by the adversary. Technically, our assumption means that in the distribution μ the probability of abstention is zero.

We prove the privacy theorem for sElect for adversaries that (*i*) are k-risk-avoiding, (*ii*) do not corrupt more than $n_{\mathsf{servers}} - 1$ mix servers, and (*iii*) leave at least $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters uncorrupted, i.e., given an instance, the set of runs for which these conditions are not met should be negligible. The mapping $\mathcal{A}$ to sets of admissible adversaries is defined accordingly (recall Definition 3). Now, the privacy theorem for sElect says that the level of privacy of sElect 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 4 (Privacy)**.** *Under the assumptions stated above and with the mapping $\mathcal{A}$ as defined above, the protocol $P_{\mathsf{sElect}}(n_{\mathsf{voters}}^{\mathsf{honest}}, n_{\mathsf{servers}}, \mu, p_{\mathsf{verif}}^{\mathsf{vote}}, p_{\mathsf{verif}}^{\mathsf{abst}}, f_{\mathsf{sElect}})$ achieves $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}-k, \mu)}(f_{\mathsf{sElect}})$-privacy w.r.t. $\mathcal{A}$.*

Proof. See Appendix C.2. □

In Figure 3.1, we present some selected values of $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}-k, \mu)}(f_{\mathsf{sElect}})$ which, by the above theorem, express the privacy level of sElect when k-risk-avoiding adversaries are considered. Since the probability distribution of the (human) voter verification codes can, in general, hardly be modeled, the result function in Figure 3.1 is simplified such that it only reveals the chosen candidates cand but not the respective verification codes $\mathsf{code}^{\mathsf{voter}}$. As can be seen from Figure 3.1, the privacy level for different k's changes only very little for 20 honest voters and almost nothing for more honest voters. Conversely, the risk of the adversary being caught increases dramatically with increasing k, i.e., the number of dropped votes. For example, if we take $p = 0.5$ similar to our mock elections, we obtain 75% for $k = 4$, 97% for $k = 10$, and $\approx 100\%$ for $k = 20$. This means that unless adversaries do not care being caught at all, privacy cannot be broken.

3.7 Implementation

In this section, we shortly describe our prototypical implementation of sElect.[12] We also briefly report on two small mock elections we carried out with sElect, with the main intention to get a first feedback on the verification rates for our fully automated VSD-based verification mechanism.

[12] As noted in Section 1.2, Enrico Scapin has implemented the sElect voting system. For more details, see [Sca18].

We have implemented sElect as a platform independent web application. Voters merely need a browser to vote and to verify their votes. In order to vote, voters go to a web site that serves what we call a *voting booth*. More precisely, a voting booth is a web server which serves a collection of static HTML/CSS/JavaScript files. There otherwise is no interaction between the voter's browser and the voting booth server: ballot creation, casting, and verification are then performed within the browser, as explained below (of course for ballot casting, the voter's browser communicates with the authentication server). The idea is that the voter can choose a voting booth, i.e., a web server, among different voting booths that she trusts and that are independent of the election authority. Voting booths might be run by different organizations as a service and independently of a specific election (see also the discussion in Section 3.1). So what abstractly was called a VSD in the previous sections, in our implementation comprises the voter's computing platform, including her browser, as well as some voting booth server which the voter picks and which serves the static JavaScript files to be executed. The JavaScript code performs the actual actions of the VSD described in Section 3.2 within the browser and without further interaction with the voting booth server.[13]

A voter enters her vote in the browser (on the voting booth's web site) and then ballot creation and verification of acknowledgments are carried out locally within the voters' browser. Votes only leave the browser encrypted (as ballots), to be submitted to the authentication server. Full receipts, i.e., all the information required for the VSD-based verification process, are saved using the browser's local storage (under the voting booth's origin); other web sites cannot access this information. When the election is over, the voter is prompted to go to her voting booth again in order to check the election result. When the voter opens the voting booth in this phase, it automatically fetches all the necessary data and carries out the automated verification procedure; if the voter's ballot has not been counted correctly, cryptographic evidence against a misbehaving server is produced, as described in Section 3.2 (see also Section 3.5). In addition to this fully automated check, the voter is given the opportunity to visit the bulletin board (web site), where she can see the result and manually check that her verification code is listed next to her choice.

Two small mock elections. To obtain user feedback and, in particular, get

[13]On a mobile device one could, for example, also provide an app to the voter which performs the task of the VSD; again there might be more apps from which the voter could choose. This of course assumes that the voter installs such an app on her device. Since the idea is that a voting booth can be used independently of a specific election, this is reasonable as well.

a first estimate of the verification ratio for the fully automated verification, we carried out two mock elections. We used a slightly modified version of the voting booth which allowed us to gather statistics concerning the user behavior. We emphasize that these field tests were not meant to be full-fledged and systematic usability studies, which we leave for future work.

The participants of these mock elections were students of our department and researchers of a national computer science project. In the former case, out of 52 cast ballots, 30 receipts were checked automatically; in the latter case, out of 22 cast ballots, 13 were checked automatically. As one can see, the verification ratio was quite high in both cases (57.5% and 59.1%). In fact, with such a high ratio, the dropping or manipulation of even a very small number of votes is detected with very high probability, according to our results in Sections 3.4, 3.5, and 3.6. Moreover, we can expect that some number of verification codes were checked manually, so the overall verification ratio might be even higher (we do not have, however, reliable data about voter-based verification).

We believe that for real elections one might obtain similar ratios: voters might be even more interested in the election outcome than in a mock election and, hence, they would tend to check the result and trigger the automated verification procedure.

3.8 Related Work

As already mentioned in the introduction, existing remote electronic voting systems range from Helios, which so far was one of the simplest systems, to very complex systems, such as Civitas. In addition, there is the idea of code voting, where voters, via an additional channel, e.g., mail, are provided with codes which they use to vote (see, e.g., [HRT10, NFSF14]). sElect is in the spirit of the former mentioned systems.

There have been several usability studies which show that for the existing remote voting systems, in particular Helios, voters often have problems with the ballot casting and verification procedure. In particular, they are confused about the concept of and the motivation for verification in Helios (see, e.g., [OBV13]). While we have not performed a systematic usability study, our case studies indicate that this seems to be less problematic in sElect.

To the best of our knowledge, all practical remote e-voting systems rely on specific cryptographic schemes, such as ElGamal encryption in combination with specific zero-knowledge proofs etc. In contrast, sElect can be used with any IND-CCA2-secure public-key encryption scheme and any EUF-CMA-secure digital signatures scheme, which, as already pointed out in the

introduction, has several advantages (arbitrary ballots can be supported, post-quantum cryptography could be used, standard cryptographic libraries). In [KMW12], the intention was to built a universally verifiable mix net (a full fledged e-voting system was not considered) from any IND-CCA2-secure public-key encryption scheme. However, even here the construction needed specific requirements (the scheme needed to allow for proofs of correct decryption), which are not necessary for sElect.

Chapter 4

Ordinos: A Verifiable Tally-Hiding Remote E-Voting System

With only very few exceptions (see below), in all existing e-voting systems which are designed to provide privacy and verifiability/accountability, the *full* election result, consisting of the number of votes per candidate or even all single votes, is revealed. Revealing everything, however, sometimes is undesirable, for example, in the following situations:

- 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 results, 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, who then move on to the runoff election.

- Elections are often carried out among a small set of voters (e.g., in boardroom or jury votes). Even in an ideal voting system, revealing the complete result leads to a low level of privacy because a single voter's vote is "hidden" behind only a low number of other votes. Therefore, in such an election, a voter may not vote for her actual preference knowing that it does not really remain private.

- In some elections, for example, within companies, associations, sports clubs, or in boardroom elections, it might be embarrassing for the loosing candidates to publish the (possibly low) number of votes they received.

- For some elections, regulations might forbid to reveal the full result to anybody or at least to the public.

These examples illustrate that, for some elections, it is desirable to hide parts of the election result. Following [SP15], we call e-voting systems that hide part of the tally *tally-hiding*. There is of course a large variety of election result functions (which determine the kind of result to be published) such systems could realize and these functions can be more or less tally-hiding. For example, one might want to reveal only the winner of an election with or without the number of votes he/she received, only the set of the first k candidates, which might be up for a runoff election, 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 they received.

So, while tally-hiding e-voting is desirable in many situations, it has received only very little attention so far. In fact, as to the best of our knowledge, only three tally-hiding e-voting systems have been proposed in the literature to date: a quite old one by Benaloh [Ben86] and two more recent ones by Szepieniec and Prenell [SP15] and by Canard et al. [CPST18]. As further discussed in Section 4.8, among other shortcomings, none of these systems comes with a rigorous cryptographic security proof and only one of these systems has been implemented. Hence, in this largely unexplored field, it remains an open problem to develop and implement a provably secure verifiable tally-hiding e-voting system and to obtain a deeper understanding of tally-hiding voting in general. This is the main goal of this chapter.

4.1 Contributions

We present Ordinos, the first provably secure verifiable tally-hiding e-voting system. Ordinos is a generic extension of the prominent Helios remote e-voting system, where Helios reveals always the full result. Ordinos supports several tally-hiding result functions, including revealing only the winner of an election, the k best/worst candidates, or the overall ranking, with or without disclosing the number of votes per candidate, respectively.

We start with describing Ordinos in Section 4.2 and then formally model Ordinos in Section 4.3. In Sections 4.4 and 4.5, we carry out a detailed cryptographic analysis proving that Ordinos provides privacy, verifiability, and accountability, with full proofs provided in Appendix C. More specifically, we show that Ordinos preserves the same level of verifiability and accountability as Helios, independently of the tally-hiding result function considered. More generally, these results demonstrate that the common definitions for

verifiability and accountability are independent of the specific result functions considered. Conversely, with result functions that hide most of the full election result, the level of privacy Ordinos provides might be much better than Helios. 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. We then show that the privacy of Ordinos can be reduced to the ideal protocol. These general results about properties of tally-hiding voting systems are of independent interest.

Our cryptographic analysis of Ordinos is based on generic properties of the employed cryptographic primitives. Hence, they can be instantiated by arbitrary cryptographic constructions satisfying these properties. In Section 4.6, we propose one such instantiation using among others Paillier public-key encryption, an MPC protocol for greater-than by Lipmaa and Toft [LT13], as well as NIZKPs by Schoenmakers and Veeningen [SV15]. We implemented Ordinos based on this instantiation and evaluated its performance, demonstrating its practicability in Section 4.7. We close this chapter with a discussion of Ordinos and its related work in Section 4.8.

4.2 Description

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 is tally-hiding. More specifically, the generic version of Ordinos, which we prove secure, supports arbitrary (tally-hiding) result functions evaluated over the aggregated votes per candidate. Our concrete instantiation (see Section 4.6) then realizes many practically relevant such 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. We use the following cryptographic primitives (for more details, see Appendix A.1, A.3, B, and A.2, respectively):

- A homomorphic, IND-CPA-secure $(t, n_{\mathsf{trustees}})$-*threshold public-key encryption scheme* $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$, for example, exponential ElGamal or Paillier.

- A *non-interactive zero-knowledge proof* (NIZKP) π^{Enc} for proving correctness and knowledge 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.

- A *multi-party computation (MPC) protocol* P_{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 P_{MPC} has to satisfy to achieve privacy, verifiability, and accountability for the overall system, Ordinos, are made precise in the following sections.

- An EUF-CMA-secure *signature scheme* $\mathcal{S}$.

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

We assume the existence of the following authenticated channels:[1] An authenticated channel from each voter V_i to the authentication server AS. These channels allow AS to ensure that only eligible voters are able to cast their ballots. An authenticated channel from each voter V_i to the bulletin board B. The voter can use the channel in order to post information on the bulletin board B, for example, a complaint in case her ballot is not published by AS (see below).

Protocol overview. A protocol run consists of the following phases:

1. *Setup phase*: Parameters and key shares are fixed/generated. The public key shares are published.

[1] 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.

2. *Voting phase*: Voters pick their choices, encrypt them, and submit their ballots.[2]

3. *Voter verification phase*: Voters verify (now or later) that their ballots have been published by the authentication server.

4. *Tallying phase*: The trustees homomorphically aggregate the ballots and run P_{MPC} in order to secretly compute and publish the election result according to the tally-hiding result function f_{tally} so that not even the trustees learn anything beyond the final tally-hiding result.

5. *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 $\mathsf{C} \subseteq \{0,\ldots,n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}} \cup \{\mathsf{abstain}\}$ of valid choices where n_{cand} denotes the number of options/candidates, n_{vpc} the number of admissible votes per option/candidate, and $\mathsf{abstain}$ models that a voter abstains from voting. For example, 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 $\mathsf{KeyShareGen}$ 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 $\mathsf{ch} \in \mathsf{C} \cap \{0,\ldots,n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}}$. In the latter

[2]We note that in the original Helios voting scheme, a voter can use a second trusted device to challenge her VSD in order to verify whether or not her VSD has in fact encrypted the correct choice (*Benaloh challenge*). This verification procedure is orthogonal to the rest of the protocol. As such, this mechanism could easily be included in Ordinos as well. For the sake of simplicity, we do not take Benaloh challenges into account.

case, the voter 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. In addition to $\vec{\mathsf{c}}_i$, the voter creates a NIZKP π_i^{Enc} in order to prove that she knows which choice ch the vector $\vec{\mathsf{c}}_i$ encrypts and that $\mathsf{ch} \in \mathsf{C} \setminus \{\mathsf{abstain}\}$.

The voter V_i submits $\mathsf{b}_i = (\mathsf{id}, \vec{\mathsf{c}}_i, \pi_i^{\mathsf{Enc}})$ as her ballot to the authentication server AS on an authenticated channel, where $\mathsf{id} \in \vec{\mathsf{id}}$ is the voter's identifier. 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.[3] The voter stores the ballot b_i as well as the acknowledgement 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 a voter does not receive a correct acknowledgement from AS, the voter tries to re-vote, and, if this does not succeed, she files a complaint on the bulletin board using the authenticated channel. If such a complaint is posted, it is in general impossible to resolve the dispute and decide who is to be blamed: AS who might not have replied as expected (but claims, for instance, that the ballot was not cast) or 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 (e.g., sElect, see Section 3.2). In practice, the voter could ask the voting authority Auth to resolve the problem.

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 [CS11]. 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 verify 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

[3] 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., [KTV12b]).

dishonestly?[4]

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., [KOKV11]). In our security analysis of Ordinos, we therefore assume that voters perform the verification procedures with a certain probability.

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 (as described in the voting phase above). 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 $\mathsf{c}_{\mathsf{unsorted}}$ with n_{cand} 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 P_{MPC} with input $\mathsf{c}_{\mathsf{unsorted}}$ 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}.[5]

Public verification phase. In this phase, every participant, including the voters or external observers, can verify the correctness of the tallying procedure, in particular, the correctness of all NIZKPs.

Instantiation and implementation. As already mentioned, in Section 4.6 we show how to efficiently instantiate Ordinos with concrete primitives. In particular, we provide an efficient realization of a relevant class of tally-hiding result functions, e.g., for publishing only the winner of an election or certain subsets or rankings of candidates. In Section 4.7, 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.

[4] As for sElect, 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., [CGGI14]). This also helps in preventing so-called ballot stuffing.

[5] π^{MPC} will typically consist of several NIZKPs, e.g., NIZKPs of correct decryption, etc. See also our instantiation in Section 4.6.

4.3 Formal Protocol Model

The Ordinos voting protocol can be modeled in a straightforward way as a voting protocol $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$ in the above sense, as detailed next.

- By n_{voters} we denote the number of voters and their voter supporting devices, and by n_{trustees} the number of trustees.
- By μ we denote a probability distribution on the set of choices, including abstention. An honest voter makes her choice according to this distribution.[6] Note that the set of valid choices is implicitly given by μ.
- By $p_{\mathsf{verify}} \in [0, 1]$ we denote the probability that an honest voter V_i performs the check described in Section 4.2, voter verification phase.[7]
- By f_{tally} we denote the (tally-hiding) result function.

The set of agents of P_{Ordinos} consists of all agents described in Section 4.2, i.e., the bulletin board B, voters $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$, the authentication server AS, trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$, and in addition, a scheduler S. As in sElect, the latter party plays the role of the voting authority Auth and schedules all other agents in a run according to the protocol phases. Also, it is the master program in every instance of P_{Ordinos}. All agents are connected via channels with all other agents; honest agents will not use all of these channels, but dishonest agents might. The honest programs $\hat{\pi}_{\mathsf{a}}$ of honest agents a are defined below. We assume that the scheduler S and the bulletin board B are honest. All other agents can possibly be dishonest. These agents can run arbitrary probabilistic (polynomial-time) programs. We note that the scheduler is only a modeling tool. It does not exist in real systems. The assumption that the bulletin board is honest is common; Helios makes this assumption too, for example. In reality, the bulletin board should be

[6] 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.

[7] 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 sElect protocol (Section 3.3). 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 4).

implemented in a distributed way (see, e.g., [CS14, KKL+18]). In addition to the participants listed above, we also have a judge J in order to model/analyze verifiability and accountability of sElect (recall Section 2.3 for details on verifiability).

Scheduler S. In every instance of P_{Ordinos}, the honest program $\hat{\pi}_{\mathsf{S}}$ of S plays the role of the master program. We assume that it is given information about which agents are honest and which are dishonest in order to be able to schedule the agents in the appropriate way. In what follows, we implicitly assume that the scheduler triggers the adversary (any dishonest party) at the beginning of the protocol run and at the end of this run. Also, the adversary is triggered each time an honest party finishes its computations (after being triggered by the scheduler in some protocol step). This keeps the adversary up to date and allows it to output its decision at the end of the run. By this, we obtain stronger security guarantees. Similarly, we assume that the judge is triggered each time any other party (honest or dishonest) finishes its computation (after being triggered by the scheduler). This gives the judge the chance to output its verdict after each protocol step. If the judge posts a message on the bulletin board B which indicates to stop the whole protocol, then the scheduler triggers once more the adversary (to allow it to output its decision) and then halts the whole system. This means that no participants are further triggered. We also let the scheduler create common reference strings (CRSs) for all the required NIZKPs, by calling the setup algorithm of the non-interactive zero-knowledge proof systems used in the protocol, and provide them to all parties.

In the remaining part of the section, we precisely describe the honest program of the scheduler depending on the voting phase.

Scheduling the setup phase. At the beginning of the election, the scheduler determines the set of possible choice $\mathsf{C} \subseteq \{0,\dots,n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}} \cup \{\mathsf{abstain}\}$ of valid choices where n_{cand} denotes the number of options/candidates, n_{vpc} the number of admissible votes per option/candidate, and $\mathsf{abstain}$ models that a voter abstains from voting. Then, the scheduler generates a random number $\mathsf{id}_{\mathsf{election}}$, the election identifier, with the length of the security parameter ℓ and sends it to the bulletin board B which publishes $\mathsf{id}_{\mathsf{election}}$ and C.[8]

After that, the scheduler first triggers all honest trustees T_k, which are supposed to generate their verification/signing key pairs $(\mathsf{Verify}_k, \mathsf{sign}_k)$ and

[8] Whenever we say that a party computes a signature on some message m, this implicitly means that the signature is computed on the tuple $(\mathsf{id}_{\mathsf{election}}, \mathsf{tag}, \mathsf{m})$ where $\mathsf{id}_{\mathsf{election}}$ is an election identifier (different for different elections) and tag is a tag different for signatures with different purposes (for example, a signature on a list of voters uses a different tag than a signature on a list of ballots).

publish the public (verification) keys Verify_k on the bulletin board B, and then all the dishonest ones. In what follows, we implicitly assume that each trustee T_k is supposed to sign all of its messages to the bulletin board under sign_k.

Afterwards, the scheduler triggers all honest trustees, and then all dishonest ones, to run the key share generation algorithm $\mathsf{KeyShareGen}$ of the public-key encryption scheme scheme $\mathcal{E}$. As a result, each trustee publishes a public key share pk_k (together with a NIZKP of correctness and knowledge of the respective secret key share sk_k), so that the public key pk can be obtained by running $\mathsf{PublicKeyGen}$ on the published public key shares.

Scheduling the voting phase. The scheduler first triggers all the honest voters and then the dishonest ones, allowing them to cast their ballots to the authentication server AS. After each such step (when the computations of a voter and the authentication server are finished), the scheduler triggers the voter again in order to allow the voter to post a complaint, if she does not get a valid acknowledgement from the authentication server. As specified below, the authentication server AS is modeled in such a way that it provides all collected ballots (even before AS publishes them on the bulletin board B) to an arbitrary participant who requests these ballots. Afterwards, the scheduler triggers the authentication server which is supposed to publish the list of ballots $\vec{\mathsf{b}}$ (containing the (first) valid ballot cast by each eligible voter) on the bulletin board B.

Scheduling the voter verification phase. Similarly to the voting phase, the scheduler triggers first the honest voters who are supposed to verify (with probability p_{verify}) the input to the tallying phase. See below for details. Afterwards, the scheduler triggers all the dishonest voters.

Scheduling the tallying phase. The scheduler runs the scheduling procedure of the given MPC protocol.

Authentication Server AS. The honest program of the authentication server AS in Ordinos is the same as the one of AS in sElect (see Section 3.3).

Bulletin Board B. The honest program of the bulletin board B in Ordinos is the same as the one of B in sElect (see Section 3.3).

Voter V_i. A voter V_i, when triggered by the scheduler S in the voting phase, picks ch_i from C according to the probability distribution μ. A choice may be either a distinct value $\mathsf{abstain}$, which expresses abstention from voting, or an integer vector from $\{0,\dots,n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}}$. If $\mathsf{ch}_i = \mathsf{abstain}$, then the voter program stops. Otherwise, if $\mathsf{ch}_i = (\mathsf{m}_{i,1},\dots,\mathsf{m}_{i,n_{\mathsf{cand}}}) \in (\{0,\dots,n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}} \cap \mathsf{C})$, the program continues as follows. The voter encrypts each integer $\mathsf{m}_{i,j}$ under the public key pk to obtain a ciphertext $\mathsf{c}_{i,j}$. Afterwards, the voter creates a NIZKP π^{Enc} of knowledge and correctness for the n_{cand}-ary relation over the

plaintext space which holds true if and only if $(\mathsf{m}_1, \ldots, \mathsf{m}_{n_{\mathsf{cand}}}) \in \mathsf{C} \setminus \{\mathsf{abstain}\}$. The ballot

$$\mathsf{b}_i = \left(\mathsf{id}_i, (\mathsf{c}_{i,1}, \ldots, \mathsf{c}_{i,n_{\mathsf{cand}}}), \pi_i^{\mathsf{Enc}}\right)$$

is then sent to the authentication server AS. The voter expects to get back an acknowledgement (a signature of AS on the submitted ballot). When this happens, the voter verifies the acknowledgement. If the acknowledgement is incorrect, the voter posts a complaint on the bulletin board via her authenticated channel. Note that the program of the voter may not get any response from AS in case AS is dishonest. To enable the voter in this case to post a complaint on the bulletin board, the scheduler triggers the voter again (still in the voting phase).

The voter V_i, when triggered by the scheduler S in the verification phase, carries out the following steps with probability p_{verify}. If ch_i was $\mathsf{abstain}$, the voter verifies that her id is not listed in the list of ballots $\vec{\mathsf{b}}$ output by the authentication server. She files a complaint if this is not the case. If $\mathsf{ch}_i \neq \mathsf{abstain}$, the voter checks that her id and her ballot b_i appear in the list of ballots $\vec{\mathsf{b}}$, output by the authentication server. As before, she files a complaint if this is not the case.

Trustee T_k. A trustee T_k, when triggered by the scheduler S in the key generation phase for the signature scheme, runs the key generation algorithm KeyGen of $\mathcal{S}$ to obtain a verification/signing key pair $(\mathsf{Verify}_k, \mathsf{sign}_k)$. Then, the trustee sends the verification key to the bulletin board B.

When triggered by the scheduler S in the key generation phase for the encryption scheme, the trustee T_k runs the key share generation algorithm $\mathsf{KeyShareGen}$ of $\mathcal{E}$ to obtain a secret key share sk_k and a public key share pk_k. Then, the trustee T_k creates a NIZKP $\pi_k^{\mathsf{KeyShareGen}}$ for proving correctness of the public key share pk_k including knowledge of an adequate secret key share sk_k. The trustee signs $(\mathsf{pk}_k, \pi_k^{\mathsf{KeyShareGen}})$ with the signing key sign_k and sends it, together with signature, to the bulletin board B.

When triggered by the scheduler S in the tallying phase, the trustee T_k reads the list of ballots $\vec{\mathsf{b}}$ published and signed by the authentication server AS from the bulletin board B. If no such list exists or if the signature is not correct or if the list is not correct (see above), the trustee aborts. Otherwise, T_k calculates

$$\mathsf{c}_{\mathsf{unsorted}} \leftarrow \left(\sum_i \mathsf{ch}_{i,1}, \ldots, \sum_i \mathsf{ch}_{i,n_{\mathsf{cand}}}\right),$$

where $\sum_i \mathsf{ch}_{i,j}$ encrypts the total number of valid votes for candidate j. Up to this step, Ordinos is completely identical to Helios.

Then, the trustees run the MPC protocol P_{MPC} with input $\mathsf{c}_{\mathsf{unsorted}}$. The output of P_{MPC} is the overall election result of Ordinos, plus some NIZKP of correct evaluation π^{MPC}.

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 the authentication server AS has not responded with a valid acknowledgement, then the judge outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$, which means that (the judge believes that) either V_i or 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 P_{MPC} the judge runs the judging procedure $\mathsf{J}_{\mathsf{MPC}}$ of P_{MPC}. If $\mathsf{J}_{\mathsf{MPC}}$ outputs a verdict, then J also outputs this verdict.

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

4.4 Verifiability and Accountability

In this section, we formally establish the level of verifiability and accountability provided by Ordinos. We show that Ordinos inherits the level of verifiability and accountability 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/accountability.

Our analysis of Ordinos in terms of verifiability and accountability uses the generic KTV framework introduced in Section 2.3. 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.

Assumptions. We prove the verifiability result for Ordinos for the goal $\gamma(k, \varphi)$, with $\gamma(k, \varphi)$ as defined in Section 2.3.2, and under the following assumptions:

(V1) The public-key encryption scheme $\mathcal{E}$ is correct (for verifiability, IND-CPA-secure 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, and the judge J are honest, i.e., $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$.

(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. We now state the verifiability and accountability level offered by Ordinos according to Definitions 1 and 2. Both levels depend on the voter verification rate p_{verify}, as described in Section 4.3.

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 $\delta_k(p_{\mathsf{verify}})$.

Theorem 5 (Verifiability)**.** *Under the assumptions (V1), (V2) and (V3) stated above, the protocol* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$ *is* $(\gamma(k, \varphi), \delta_k(p_{\mathsf{verify}}))$*-verifiable by the judge* J *where* $\delta_k(p_{\mathsf{verify}}) = (1 - p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$.

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 verification procedure. The probability for this is $(1-p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$: the exponent is not $k+1$, as one might expect, but $\lceil \frac{k+1}{2} \rceil$ because, according to the formal definition of $\gamma(k,\varphi)$ (see Section 2.3.2), 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.

In [KTV10b], it was shown that accountability implies verifiability. The verifiability theorem therefore follows immediately from the stronger accountability result presented next.

Accountability. For the accountability theorem, we make the same assumptions (V1) to (V3) as for the verifiability theorem above, 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) above as follows so that Ordinos guarantees accountability.

(V3) The MPC protocol P_{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 4.2 fails. (Our instantiation presented in Section 4.6 fulfills this assumption.)

Now, in order to state our theorem for accountability of Ordinos, we use the following accountability property Φ_k which covers the goal $\gamma(k,\varphi)$.[9]

Let χ_i denote the set of runs of an instance of P_{Ordinos} where voter V_i complains that she did not get a receipt from AS. In such runs the judge cannot be sure who to blame individually (V_i or AS?). But he does know that at least one of them is dishonest (recall the discussion in Section 4.2). This is captured by the accountability constraint $\chi_i \Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

Similarly, let χ_i' contain all runs of P_{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 $\chi_i' \Rightarrow \mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

The accountability theorem for Ordinos (see below) states that if the adversary breaks the goal $\gamma(k,\varphi)$ in a run of P_{Ordinos} but neither 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

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

[9] Observe that the accountability properties for Ordinos and sElect (Section 3.5) are essentially the same.

where $\chi = \bigcup_{i \in \{1,\ldots,n_{\mathsf{voters}}\}} (\chi_i \cup \chi'_i)$.

For P_{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 (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 (individual accountability).

Theorem 6 (Accountability)**.** *Under the assumptions (V1), (V2) and (V3) stated above and the mentioned judging procedure of the judge* J*, the protocol* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$ *is* $(\Phi_k, \delta_k(p_{\mathsf{verify}}))$*-accountable for* J*, where* $\delta_k(p_{\mathsf{verify}}) = (1 - p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$

Proof. See Appendix C.3. □

Note that, possibly surprisingly, our results show that the level of verifiability and accountability provided by Ordinos is independent of the result function f_{tally}, and hence, independent of how much of the full tally is hidden by 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).

4.5 Privacy

In this section, we carry out a rigorous analysis of the vote privacy of Ordinos. We show that Ordinos has a high level of privacy for the class of adversaries which are not willing to take a high risk of being caught cheating. More specifically, we show that for this class of adversaries the level of privacy is ideal, i.e., coincides with the level of privacy an ideal voting protocol achieves, where merely the election result according to the (tally-hiding) result function considered is published.

After that, to better understand the relationship between the privacy level of Ordinos and the (tally-hiding) result function used, we provide a detailed comparison between the "traditional" result function (number of votes per candidate) and two important tally-hiding result functions (only the winner(s) or the overall ranking). These results demonstrate that Ordinos

can dramatically improve vote privacy, depending on the tally-hiding result function.

Privacy result. As in sElect (or Helios), an adversary who controls the authentication server in Ordinos 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 4.4). 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. Therefore, as for sElect, we consider k-risk-avoiding adversaries (Section 3.6.1) to analyze the privacy level of Ordinos.

We now state 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.

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 4.2): the public-key encryption scheme $\mathcal{E}$ is IND-CPA-secure, the signatures are EUF-CMA-secure, the proofs $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} are NIZKPs, and the MPC protocol realizes an ideal functionality which essentially takes a vector of ciphertexts and returns f_{tally} evaluated on the corresponding plaintexts (see Appendix A.1, A.3 and B for details).

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, our assumption means that in the distribution μ the probability of abstention is 0.

We prove the privacy theorem for Ordinos for adversaries that (i) are k-risk-avoiding, (ii) do not corrupt more than $t-1$ trustees, and (iii) leave at least $n^{\mathsf{honest}}_{\mathsf{voters}}$ voters uncorrupted, i.e., given an instance, the set of runs for which these conditions are not met should be negligible. The mapping $\mathcal{A}$ to sets of admissible adversaries is defined accordingly. Now, the privacy

theorem for Ordinos says that the level of privacy of Ordinos for this class of adversaries is the same as the privacy level $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{res}})$ for the ideal protocol with $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters (see Section 2.5.2 for the definition of the ideal voting protocol and its privacy level).

Theorem 7 (Privacy)**.** *Under the assumptions stated above and with the mapping $\mathcal{A}$ as defined above, the protocol $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$ achieves $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{res}})$-privacy w.r.t. $\mathcal{A}$ where the result function f_{res} first counts the number of votes per candidate and then evaluates f_{tally}.*[10]

Proof. See Appendix C.4. □

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 of weakening privacy. In fact, as illustrated below, 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 (tally-hiding) result function can very well have a big effect on the level of privacy of the ideal protocol, and hence, also on Ordinos: whether only the winner of an election is announced or the complete result is published typically has a big effect on the level of privacy provided by the system.

Privacy comparison of result functions. In the following, we compare the optimal privacy levels for the following result functions for which, among others, we instantiated and implemented the generic Ordinos voting protocol (see Section 4.6 and 4.7):

- 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)

[10] 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}.

We denote the corresponding privacy levels by $\delta^{\mathsf{ideal}}_{\mathsf{rank}}$, $\delta^{\mathsf{ideal}}_{\mathsf{win}}$, and $\delta^{\mathsf{ideal}}_{\mathsf{complete}}$, respectively.

1. *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 4.1 for an example. Another, more extreme example is given in Figure 4.2.

2. *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 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; the latter is illustrated by Figure 4.3.

3. *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 ch_0 or ch_1 is to choose ch_0 if ch_0 gets more votes than ch_1, and this strategy is applicable even if only the ranking is published. We note that f_{win} is still better, i.e., $\delta^{\mathsf{ideal}}_{\mathsf{win}} < \delta^{\mathsf{ideal}}_{\mathsf{complete}} = \delta^{\mathsf{ideal}}_{\mathsf{rank}}$. A concrete example is given in Figure 4.4.

We note that these results yield a lower bound for privacy for tally-hiding systems in general.

4.6 Instantiation

In this section, we provide an instantiation of the generic Ordinos protocol with concrete cryptographic primitives; in Section 4.7, we then describe our

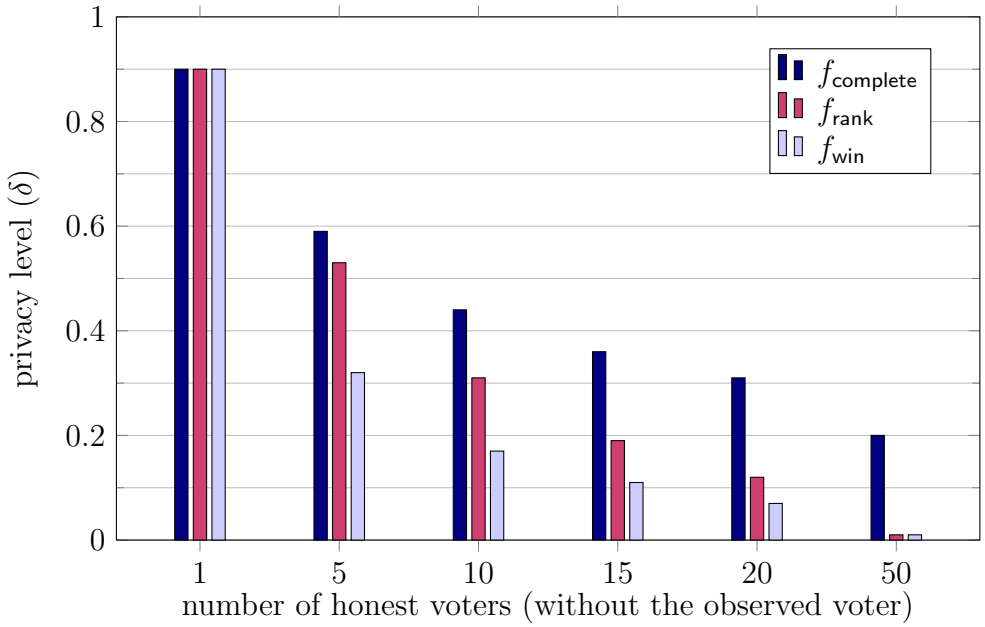

Figure 4.1: Level of privacy (δ) for the ideal protocol with three candidates, $p_1 = 0.6$, $p_2 = 0.3$, $p_3 = 0.1$ and no dishonest voters.

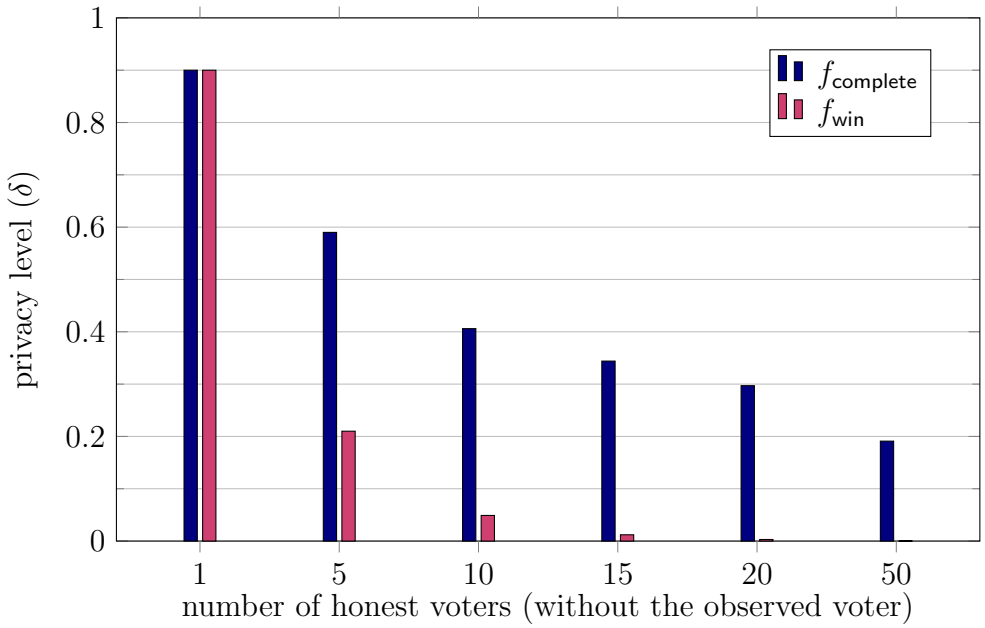

Figure 4.2: Level of privacy (δ) for the ideal protocol with two candidates and no dishonest voters. Probability for abstention: 0.3, $p_1 = 0.1, p_2 = 0.6$.

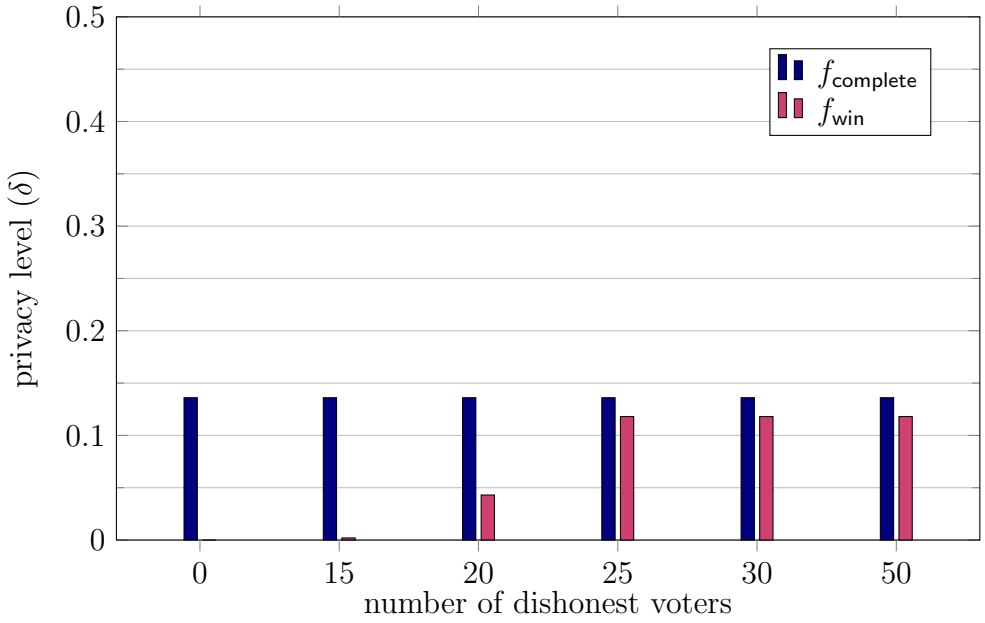

Figure 4.3: Level of privacy (δ) for the ideal protocol with two candidates and $n = 100$ honest voters. Probability for abstention: 0.3, $p_1 = 0.1, p_2 = 0.6$

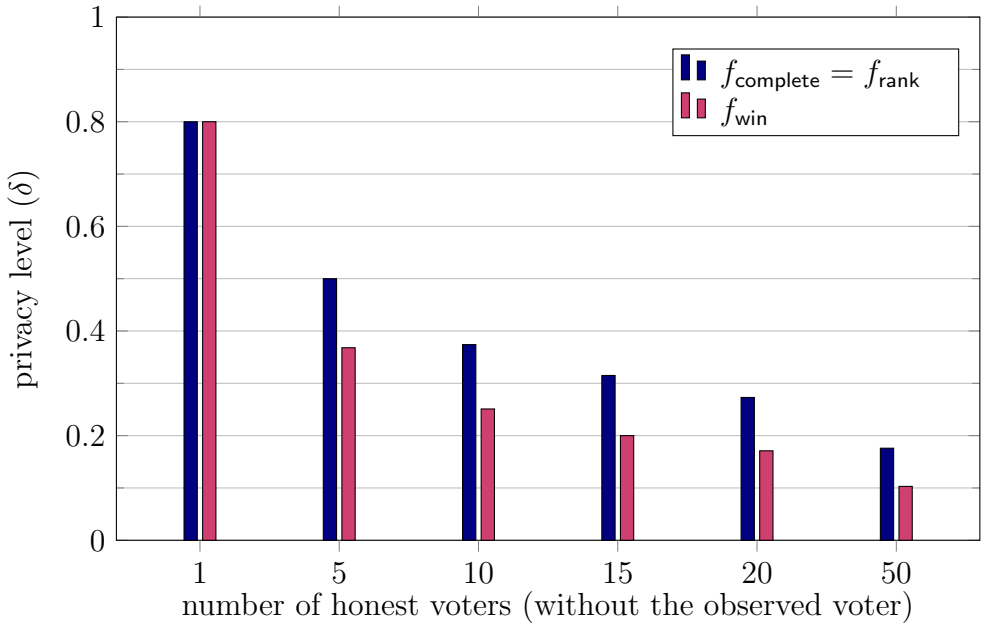

Figure 4.4: Level of privacy (δ) for the ideal protocol with 5 candidates and a uniform distribution on the candidates.

implementation of this instantiation of Ordinos and provide several benchmarks, demonstrating its practicability.

Tally-hiding result functions. Our instantiation can be used to realize many different practically relevant tally-hiding result functions. 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 three 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 [Pai99] as the $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E}$. The main reason for choosing Paillier instead of exponential ElGamal [Gam84] (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 [SV15], in combination with [CDS94].

The NIZKP $\pi^{\mathsf{KeyShareGen}}$ depends on the way public and private keys are shared among the trustees. One could, for example, employ the protocol by Algesheimer et al. [ACS02], 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 4.7), 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 P_{MPC} for evaluating practically relevant tally-hiding result functions, which at the same time satisfies the conditions for verifiability/accountability (see Section 4.4) as well as privacy (see Section 4.5). We now describe such a construction.

Overview of P_{MPC}. The cornerstone of our instantiation of P_{MPC} is a secure MPC protocol $P^{\mathsf{gt}}_{\mathsf{MPC}}$ 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 $P^{\mathsf{gt}}_{\mathsf{MPC}}$ with the "greater-than" MPC protocol by Lipmaa and Toft [LT13] 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 [SV15]. Lipmaa and Toft demonstrated that their protocol is secure in the malicious setting. Due to the NIZKPs this protocol employs, it even provides individual accountability in our specific instantiation in the sense of Section 2.3, i.e., if the outcome of the protocol is incorrect, everyone can identify the misbehaving trustee(s). 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 4.7).

We also use the secure MPC protocol $P^{\mathsf{eq}}_{\mathsf{MPC}}$ by Lipmaa and Toft [LT13] which secretely evaluates equality of two secret integers. Similarly to $P^{\mathsf{gt}}_{\mathsf{MPC}}$, this protocol, in our instantiation, also provides individual accountability.

Now, P_{MPC} is carried out in two phases in Ordinos:

1. Given the vector $\vec{\mathsf{c}}_{\mathsf{unsorted}}$ of the encrypted number of votes per candidate (see Section 4.2), the trustees collaboratively run several instances of the greater-than-test $P^{\mathsf{gt}}_{\mathsf{MPC}}$ in order to obtain a ciphertext vector $\vec{\mathsf{c}}_{\mathsf{comp}}$ which encrypts the overall ranking of the candidates.

2. The resulting ciphertext vector (plus possibly $\vec{\mathsf{c}}_{\mathsf{unsorted}}$) is used to realize the desired tally-hiding result function.

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, $\vec{\mathsf{c}} = (\mathsf{c}[1], \ldots, \mathsf{c}[n_{\mathsf{cand}}])$ 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 $\vec{\mathsf{c}}_{\mathsf{unsorted}} = (\mathsf{c}_{\mathsf{unsorted}}[1], \ldots, \mathsf{c}_{\mathsf{unsorted}}[n_{\mathsf{cand}}])$ where $\mathsf{c}_{\mathsf{unsorted}}[i]$ encrypts the total number of votes for the ith candidate.

In the second step, we essentially apply the direct sorting algorithm by Çetin et al. [ÇDSS15] to $\vec{\mathsf{c}}_{\mathsf{unsorted}}$.

More precisely, in what follows we denote by $\mathsf{Dec}(c)$ 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 $P^{\mathsf{eq}}_{\mathsf{MPC}}$ with input $(\mathsf{c}_{\mathsf{unsorted}}[i], \mathsf{c}_{\mathsf{unsorted}}[j])$ and output $\mathsf{c}_{\mathsf{eq}}[i, j]$ which then decrypts to 1

if $\mathsf{Dec}(\mathsf{c}_{\mathsf{unsorted}}[i])$ equals $\mathsf{Dec}(\mathsf{c}_{\mathsf{unsorted}}[j])$ and to 0 otherwise. Clearly, the trustees need to run the protocol $P^{\mathsf{eq}}_{\mathsf{MPC}}$ only $\frac{(n_{\mathsf{cand}}-1)n_{\mathsf{cand}}}{2}$ many times because $\mathsf{c}_{\mathsf{eq}}[i,i]$ always decrypts to 1 and $\mathsf{c}_{\mathsf{eq}}[j,i] = \mathsf{c}_{\mathsf{eq}}[j,i]$. 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-test $P^{\mathsf{gt}}_{\mathsf{MPC}}$ with input $(\mathsf{c}_{\mathsf{unsorted}}[i], \mathsf{c}_{\mathsf{unsorted}}[j])$ and output $\mathsf{c}_{\mathsf{gt}}[i,j]$ which decrypts to 1 if we have $\mathsf{Dec}(\mathsf{c}_{\mathsf{unsorted}}[i]) \geq \mathsf{Dec}(\mathsf{c}_{\mathsf{unsorted}}[j])$ and to 0 otherwise. Thanks to the previous step, the trustees need to run the $P^{\mathsf{gt}}_{\mathsf{MPC}}$ protocol only $\frac{(n_{\mathsf{cand}}-1)n_{\mathsf{cand}}}{2}$ 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 $n_{\mathsf{cand}} \times n_{\mathsf{cand}}$ *comparison matrix* M_{comp}:

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

Based on this matrix, everyone can compute an encrypted overall ranking of the candidates: for each column i of M_{comp}, the homomorphic sum

$$\vec{\mathsf{c}}_{\mathsf{comp}}[i] = \sum_{j=1}^{n_{\mathsf{cand}}} \mathsf{c}_{\mathsf{gt}}[i,j]$$

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 $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{comp}}[i]) = 1$ because $\mathsf{Dec}(\mathsf{c}_{\mathsf{gt}}[i,i]) = 1$, and if it has received the most votes, then $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{comp}}[i]) = n_{\mathsf{cand}}$. We collect all of these ciphertexts in a *ranking vector*

$$\vec{\mathsf{c}}_{\mathsf{comp}} = (\vec{\mathsf{c}}_{\mathsf{comp}}[1], \dots, \vec{\mathsf{c}}_{\mathsf{comp}}[n_{\mathsf{cand}}])\,.$$

Second phase: Calculating the election result. Using $\vec{\mathsf{c}}_{\mathsf{comp}}$ and $\vec{\mathsf{c}}_{\mathsf{unsorted}}$, we can, for example, realize the following families of tally-hiding result functions and combinations thereof. First note that, for example, $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{comp}}) = (6,6,6,3,3,3)$ 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, $\mathsf{Dec}(\vec{\mathsf{c}}_{\mathsf{comp}}) = (6,6,6,3,3,2)$ is not a possible plaintext ranking vector.

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

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

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

3. *Including the number of votes*: The trustees decrypt the ciphertext $\mathsf{c}_{\mathsf{unsorted}}[i]$ 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 $P^{\mathsf{lt}}_{\mathsf{MPC}}$ from the results of the equality tests $P^{\mathsf{eq}}_{\mathsf{MPC}}$ and the greater-than tests $P^{\mathsf{gt}}_{\mathsf{MPC}}$ 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_{comp} 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 $P^{\mathsf{gt}}_{\mathsf{MPC}}$ with input $(\mathsf{c}_{\mathsf{unsorted}}[i], \mathsf{Enc}(\tau))$ 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. 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}.

Accountability of our Instantiation of Ordinos. Our instantiations of $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$ provide individual accountability, i.e., everyone can tell whether a trustee misbehaved, mainly due to the NIZKPs employed. More precisely, in $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$, 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 P_{MPC} provides individual accountability w.r.t. the goal $\gamma(0, \varphi)$ and accountability tolerance 0 up to the point where $\vec{\mathsf{c}}_{\mathsf{comp}}$ 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 P_{MPC}, again $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$ are used as well as distributed *verifiable* decryption (which anyway is part of $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$). 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 8 (Accountability (MPC))**.** *Let* $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{B})$. *Then, the protocol* P_{MPC}*, as defined in Section 4.6, provides individual accountability for the goal* $\gamma(0, \varphi)$ *and accountability level 0.*

With this, assumption (V3) for Theorem 6 is satisfied. Since the distributed Paillier public-key encryption scheme is correct, and 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 1 (Accountability)**.** *The instantiation of* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{Ordinos}})$ *presented above is* $(\Phi_k, \delta_k(p_{\mathsf{verify}}))$*-accountable w.r.t. the judge* J*, where* $\delta_k(p_{\mathsf{verify}}) = (1 - p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$.

Privacy of our Instantiation of Ordinos. Lipmaa and Toft [LT13] showed that $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$ 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 [SV15] and under the assumption that at least the threshold of t trustees are honest. Now, it is easy to show that, given $P^{\mathsf{gt}}_{\mathsf{MPC}}$ and $P^{\mathsf{eq}}_{\mathsf{MPC}}$, the sorting algorithm that in the end yields $\vec{\mathsf{c}}_{\mathsf{comp}}$ does not leak any information (the same operations are performed on all ciphertexts and all results are encrypted). Similarly, the evaluation of f_{Ordinos} as discussed above also does not leak any information except for the final result according to f_{Ordinos}. From this we can conclude that our instantiation of P_{MPC} realizes the ideal functionality $\mathcal{I}_{\mathsf{MPC}}$ defined in Appendix B, which given a vector of encrypted integers (in our case $\mathsf{c}_{\mathsf{unsorted}}$) returns the result of f_{Ordinos} evaluated on the (plaintext) integers.

Theorem 9 (Privacy (MPC))**.** *The protocol* P_{MPC}*, as defined above, realizes* $\mathcal{I}_{\mathsf{MPC}}$ *for tally-hiding result functions* f_{Ordinos} *as described above.*

With this, our instantiation of the generic Ordinos system satisfies all assumptions made in Theorem 7, and hence, as an immediate corollary of this theorem we obtain that this instantiation essentially provides the same level of privacy as the ideal voting protocol for tally-hiding result functions f_{Ordinos}.

Corollary 2 (Privacy)**.** *The above instantiation of* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{Ordinos}})$ *with* $n_{\mathsf{voters}}^{\mathsf{honest}}$ *honest voters achieves* $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}-k, \mu)}(f_{\mathsf{res}})$*-privacy w.r.t. the mapping* $\mathcal{A}$ *to sets of admissible adversaries, with* $\mathcal{A}$ *and* f_{res} *as in Theorem 7.*

4.7 Implementation

We implemented Ordinos for the instantiation described in Section 4.6. The main purpose of our implementation was to be able to evaluate the performance of the system in the tallying phase, which is the most time critical part. Our benchmarks therefore concentrate on the tallying phase. In particular, we generated the offline material for $P^{\mathsf{eq}}_{\mathsf{MPC}}$ and $P^{\mathsf{gt}}_{\mathsf{MPC}}$ in a trusted way (see Section 4.6 for alternatives).

Recall that the tallying phase consists of two parts. In the first part, the trustees generate $\vec{\mathsf{c}}_{\mathsf{comp}}$ for input $\vec{\mathsf{c}}_{\mathsf{unsorted}}$. In the second part, the trustees evaluate a specific tally-hiding result function with input $\vec{\mathsf{c}}_{\mathsf{comp}}$ (and possibly $\vec{\mathsf{c}}_{\mathsf{unsorted}}$). The first part, in particular constructing M_{rank}, 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 tally-hiding 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; see below for details of the machines.

We first note that the length of encrypted integers to be compared by $P^{\mathsf{gt}}_{\mathsf{MPC}}$ determines the number of recursive calls of $P^{\mathsf{gt}}_{\mathsf{MPC}}$ from [LT13]. This protocol, in a nutshell, 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 $P^{\mathsf{gt}}_{\mathsf{MPC}}$. 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.

Benchmarks and results 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 $\vec{\mathsf{c}}_{\mathsf{comp}}$) and then briefly discuss the second part.

Figure 4.5 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/lower graph in Figure 4.6 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 feasible in this figure. This is due to the logarithmic online complexity of $P^{\mathsf{gt}}_{\mathsf{MPC}}$.

Figure 4.6 also demonstrates that the parameter that determines the running time is the number of candidates, as $P^{\mathsf{gt}}_{\mathsf{MPC}}$ needs to be invoked $\mathcal{O}(\mathsf{n}^2_{\mathsf{cand}})$ times to construct M_{rank} (see Section 4.6).

Furthermore, Figure 4.6 shows that the running time is quite independent of the specific networks over which the trustees communicate. In the local network, where we run each trustee (up to 8) on an ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM), the running time is essentially the same as when we run 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[11] 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 ESPRIMOs perform a greater-than test locally

[11]One machine is as above, the second is an Intel Pentium G4500 (64bit, 2x3.5 GHz Dualcore, 8 GB RAM, running Windows 10), and the third is an Intel Core i7-6600U (CPU @ 2.60GHz, 2801 Mhz, 2 Cores 8 GB RAM, running Windows 10).

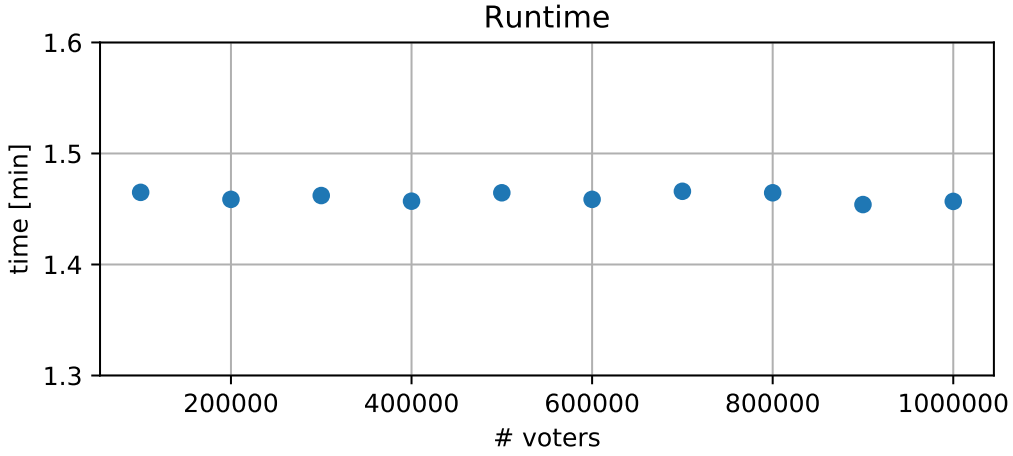

Figure 4.5: Three trustees on a local network and five candidates; 32-bit integers for vote counts.

	3 candidates		5 candidates	
# voters	[CPST18]	Ordinos	[CPST18]	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 4.1: Comparison to [CPST18] (three trustees, time in minutes).

in about 8.5 seconds, the slowest machine in this setup needs 10.5 seconds. (*ii*) The Internet connections from the home networks are slower than those in *Internet 1*.

Finally, we benchmarked a tally-hiding function for the second part of the tallying phase, namely the one which reveals the set (without ranking) of the candidates on the first n positions (see Section 4.6). This is in fact the most costly function among the functions listed in Section 4.6 as it requires to perform a greater-than test with every entry of $\vec{c}_{\mathsf{comp}}$; equality-tests are much cheaper. 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.

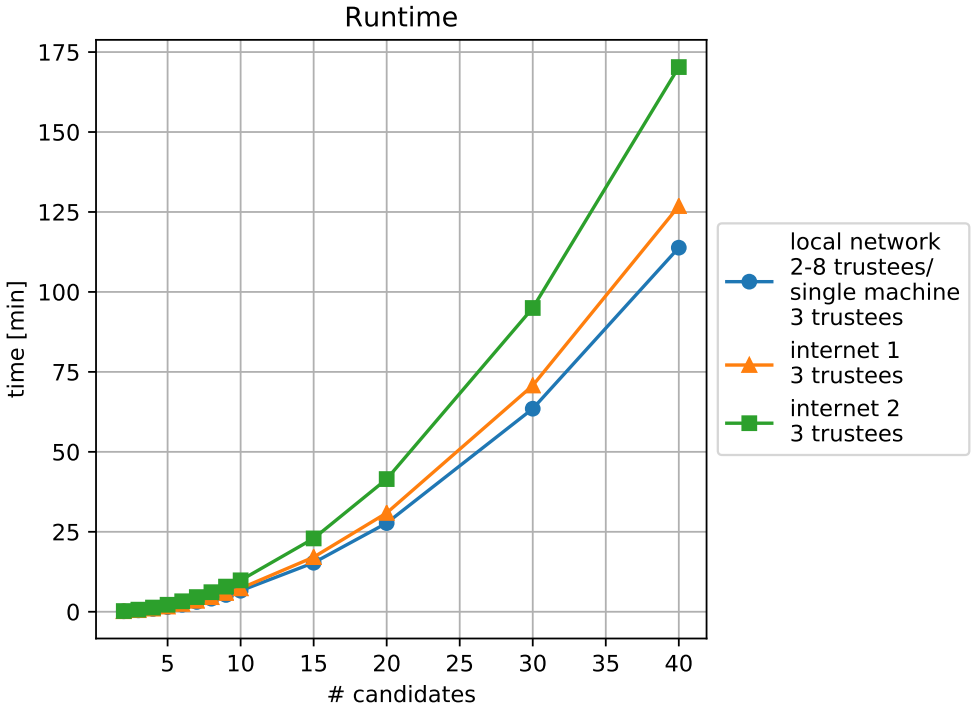

Figure 4.6: Trustees on a single machine, local network and on the Internet; 16-bit integers for vote counts.

4.8 Related Work and Discussion

In this section, we compare Ordinos with the only three tally-hiding voting protocols [Ben86, SP15, CPST18] that have been proposed so far, and a further voting protocol [CPRT18] that employs secure MPC for improving privacy and coercion-resistance, but without being fully tally-hiding.

Benaloh [Ben86] 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.

Szepieniec and Prenell [SP15] 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 [SP15], Appendix A for details). Furthermore, as discussed in Section 5.11, the verifiability definition proposed by the authors is problematic. Finally, just as the protocol by Benaloh, this protocol has not been implemented.

Canard et al. [CPST18] 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 and analyzing its security was not in the scope of the paper. In Table 4.1, we briefly compare the performance of our implementation with theirs, using the only available benchmarks published in [CPST18], 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 4.1, 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 [ST06] 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 [CPST18] would actually be connected over a network. As demonstrated in Section 4.7, in our case a network does in fact not cause much overhead.

Also very recently, Culnane et al. [CPRT18] 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.

Alternatively to all of these protocols and to Ordinos, one could, of course, also try to employ a generic secure MPC protocol for tally-hiding voting (e.g., [DPSZ12]). However, to the best of our knowledge, there is no MPC protocol in the literature that provides what we require for Ordinos all at the same time: accountability (sometimes called *identifiable abort* in the context of secure MPC), privacy, practicality, and a suitable threshold structure. Furthermore, our instantiation of Ordinos seamlessly extends Helios, which is, as mentioned, one of the most prominent e-voting systems.

Chapter 5

Verifiability Notions for E-Voting Protocols

For the systematic security analysis of verifiable e-voting systems, one challenge has been to formally and precisely capture the meaning of verifiability. While the first attempts at a formal definition stem from the late 1980s [Ben87], new definitions are still being put forward, with many definitions having been proposed in the last few years [CFP+10, KRS10, JCJ10, KTV10b, CGGI14, KZZ15b, KZZ15b, KZZ15a, SFC15, SP15].

The definitions differ in many aspects, including the classes of protocols they capture, the underlying models and assumptions, the notation, and importantly, the formulations of the very core of the meaning of verifiability.

This is an unsatisfying state of affairs, which leaves the research on the verifiability of e-voting protocols and systems in a fuzzy state and raises many questions, such as: What are the advantages, disadvantages, problems, and limitations of the various definitions? How do the security guarantees provided by the definitions compare? Are they similar or fundamentally different? Answering such questions is non-trivial. It requires some common basis on which the definitions can be discussed and compared.

5.1 Contributions

First, we show that the essence of all formal definitions of verifiability proposed in the literature so far can be cast in one framework. We choose the framework proposed by Küsters, Truderung, and Vogt [KTV10b] for this purpose (Section 2.3). The generic definition of verifiability in this framework (Definition 1) is applicable to essentially any kind of protocol, with a flexible way of dealing with various trust assumptions and types of corruption. Most

importantly, it allows us to capture many kinds and flavors of verifiability.

The casting of the different definitions in one framework is an important contribution by itself as it yields a uniform treatment of verifiability. This uniform treatment enables us to provide a detailed and systematic comparison of the different formal definitions of verifiability proposed in the literature until now. We present thorough discussions of all relevant definitions and models concerning their advantages, disadvantages, problems, and limitations, resulting in various new insights concerning the definitions themselves and their relationships. Among others, it turns out that while the definitions share a common intuition about the meaning of verifiability, the security guarantees that are actually captured and formalized often vary, with many technical subtleties involved. Cast in tailored models, different, sometimes implicit, and often unnecessary assumptions about the protocol structure or the trust assumptions are made. For some definitions, we point out severe limitations and weaknesses.

Finally, we distill these discussions and insights into detailed guidelines that highlight several aspects any verifiability definition should cover. Based on the KTV framework, we provide a solid, general, and flexible verifiability definition that covers a wide range of protocols, trust assumptions, and voting infrastructures. Even if alternative frameworks are used, for example in order to leverage specific proof techniques or analysis tools, our guidelines provide insights on which parameters may be changed and what the implications of such modifications are.

This lays down a common, uniform, and yet general basis for all design and analysis efforts of existing and future e-voting protocols. As such, our work offers a well-founded reference point for future research on the verifiability of e-voting systems and protocols.

Structure of this chapter. In Sections 5.2 to 5.11, we cast various definitions in this framework and based on this we carry out detailed discussions on these definitions. The mentioned definitions and guidelines we distill from our discussions, together with various insights, are presented in Section 5.12.

5.2 A Specific Verifiability Goal by Küsters et al.

In [KTV10b], Küsters et al. also propose a specific family of goals for e-voting protocols that they used in [KTV10b] as well as subsequent works [KTV14, KTV12b, KTV11]. We present this family of goals below, as well as the way Küsters et al. have instantiated the model when applied to concrete

protocols. Since this is a specific instantiation of the KTV framework, we can omit the casting of their definition in this framework.

5.2.1 Model

When applying the KTV framework in order to model specific e-voting protocols, Küsters et al. model static corruption of parties. That is, it is clear from the outset whether or not a protocol participant (and in particular a voter) is corrupted. An honest voter V runs her honest program π_{V} with her choice $\mathsf{ch} \in \mathsf{C}$. This choice is called the *actual choice* of the voter, and says how the voter intends to vote.

5.2.2 Verifiability

While just as in Definition 1, the verifiability definition proposed by Küsters et al. does not require to fix a specific goal, for e-voting they propose a family $\{\gamma_k\}_{k\geq 0}$ of goals, which has been applied to analyze various e-voting protocols and mix nets [KTV10b, KTV14, KTV12b, KTV11].

Roughly speaking, for $k \geq 0$, the goal γ_k contains exactly those runs of the voting protocol in which *all but up to* k votes of the honest voters are counted correctly *and* every dishonest voter votes at most once.

Before recalling the formal definition of γ_k from [KTV10b], we first illustrate γ_k by a simple example. For this purpose, consider an election with five eligible voters, two candidates, with the result of the election simply being the total number of votes per candidate. Let the result function f_{res} (see Section 2.1) be defined accordingly. Now, let r be a run with three honest and two dishonest voters such that A, A, B are the actual choices of the honest voters in r and the published election result in r is the following: one vote for A and four votes for B. Then, the goal γ_1 is satisfied because the actual choice of one of the honest voters choosing A can be changed to B and at the same time the choice of each dishonest voter can be B. Hence, the result is equal to $f_{\mathsf{res}}(A, B, B, B, B)$, which is the published result. However, the goal γ_0 is not satisfied in r because in this case, all honest voters' choices (A, A, B) have to be counted correctly, which, in particular, means that the final result has to contain at least two votes for A and at least one vote for B. In particular, a final result with only two votes for A but none for B would also not satisfy γ_0, but it would satisfy γ_1. (Recall from Section 2.1 that abstention is a possible choice.)

Definition 4 (Goal γ_k)**.** *Let r be a run of an e-voting protocol. Let $n_{\mathsf{voters}}^{\mathsf{honest}}$ be the number of honest voters in r and $n_{\mathsf{voters}}^{\mathsf{dishonest}} = n_{\mathsf{voters}} - n_{\mathsf{voters}}^{\mathsf{honest}}$ be the*

number of dishonest voters in r. Let $\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}}$ be the actual choices of the honest voters in this run, as defined above. Then γ_k is satisfied in r *if there exist valid choices $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}}$ such that the following conditions hold true:*

1. *The multiset $\{\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}}\}$ contains at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ elements of the multiset $\{\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}}\}$.*

2. *If a result is published in r, it is equal to $f_{\mathsf{res}}(\{\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}}\})$.*

If no election result is published in r, then γ_k is not satisfied in r.

With this goal, Definition 1 requires that if more than k votes of honest voters were dropped/manipulated or the number of votes cast by dishonest voters (which are subsumed by the adversary) is higher than the number dishonest voters (*ballot stuffing*), then the judge should not accept the run. More precisely, the probability that the judge nevertheless accepts the run should be bounded by δ.

We note that the definition of γ_k does not require that choices made by dishonest voters in r need to be extracted from r in some way and that these extracted choices need to be reflected in $\{\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}}\}$: the multiset $\{\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}}\}$ of choices is simply quantified existentially. It has to contain $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ honest votes but no specific requirements are made for votes of dishonest voters in this multiset. They can be chosen fairly independently of the specific run r (except for reflecting the published result and the requirement that there is at most one vote for every dishonest voter). This is motivated by the fact that, in general, one cannot provide any guarantees for dishonest voters, since, for example, their ballots might be altered or ignored by dishonest authorities without the dishonest voters complaining (see also the discussion in [KTV10b]).

5.2.3 Discussion

The goal γ_k makes only very minimal assumptions about the structure of a voting system. Namely, it requires only that, given a run r, it is possible to determine the actual choice (intention) of an honest voter and the actual election result. Therefore, the goal γ_k can be used in the analysis of a wide range of e-voting protocols.

One drawback of the goal γ_k is that it assumes static corruption. Another disadvantage of γ_k (for $k > 0$) is the fact that it does not distinguish between honest votes that are dropped and those that are turned into different valid votes, although the impact on the final result by the second kind of manipulation is stronger than the one by the first kind. To illustrate this issue,

consider two voting protocols P_1 and P_2 (with the result function f_{res} being the counting function). In P_1, the adversary might not be able to turn votes by honest voters into different valid votes, e.g., turn a vote for candidate A into a vote for B. This can be achieved if voters sign their ballots. In this case, the adversary can only drop ballots of honest voters. In P_2, voters might not sign their ballots, and hence, the adversary can potentially manipulate honest votes. Now, P_1 obviously offers stronger verifiability because in P_1 votes of honest voters can only be dropped, but not changed: while in P_2 the adversary could potentially turn five honest votes, say for candidate A, into five votes for B, in P_1 one could at most drop the five honest votes, which is less harm. Still, both protocols might achieve the same level of verifiability in terms of the parameters γ_k and δ. If γ_k distinguished between dropping of votes and manipulation, one could distinguish the security levels of P_1 and P_2.

In Section 2.3.2, we have introduced a refinement of γ_k which solves this problem.

5.3 Verifiability by Benaloh

In this section, we study the verifiability definition by Benaloh [Ben87]. This definition constitutes the first formal verifiability definition proposed in the literature, and hence, the starting point for the formal treatment of verifiability. This definition is close in its essence to the one discussed in Section 5.2.

5.3.1 Model

Following [Ben87], an $(t, n_{\mathsf{trustees}}, n_{\mathsf{voters}})$-*election system* E is a synchronous system of communicating processes (probabilistic Turing machines), consisting of n_{trustees} trustees, n_{voters} voters and further participants. Each process of an election system controls one *bulletin board*. Each bulletin board can be read by every other process, but only be written by the owner.

The intended (honest) behavior of the system participants is specified by an election schema. An $(t, n_{\mathsf{trustees}}, n_{\mathsf{voters}})$-*election schema* S consists of a collection of programs to be used by the participants of an $(t, n_{\mathsf{trustees}}, n_{\mathsf{voters}})$-election system and an efficiently computable function Verify, which, given the security parameter ℓ and the messages posted to the public bulletin boards, returns either accept or reject. The election schema S describes a program π_{T} for each trustee process and two possible programs for each voter: π_{yes} to be used to cast a "yes" vote and program π_{no} to be used to cast a "no" vote. At the end of the election, each trustee T_k releases a value

τ_k.

Any process which follows (one of) its program(s) prescribed by S is said to be *proper*. We say that a voter *casts a valid "yes" vote*, if the messages it posts are consistent with the program π_{yes}, and similarly for a "no" vote. Note that a proper voter, by definition, always casts a valid vote; an improper voter may or may not cast a valid vote, and if it does not cast a valid vote, that fact may or may not be detectable by others.

The result res of an election is the pair (t_{yes}, t_{no}), where t_{yes} and t_{no} are the numbers of voters who cast valid "yes" and "no" votes, respectively. Note that this pair expresses the expected result corresponding to the cast valid votes. The result of the election is said to be *correct* if $\rho(\tau_1, \ldots, \tau_m) = (t_{yes}, t_{no})$, where ρ is a pre-determined function. The expression $\rho(\tau_1, \ldots, \tau_m)$ describes the actual result, that is the result of the election as jointly computed by the trustees (and combined using the function ρ).

5.3.2 Verifiability

Now, in [Ben87], verifiability is defined as follows.

Definition 5 (Verifiability)**.** *Let δ be a function of ℓ. The $(t, n_{\mathsf{trustees}}, n_{\mathsf{voters}})$-election schema S is said to be* verifiable with confidence $1 - \delta$ *if, for any election system E, Verify satisfies the following properties for random runs of E using security parameter ℓ:*

1. *If at least t trustees are proper in E, then, with probability at least $1 - \delta(\ell)$, Verify returns accept and the election result is correct.*

2. *The joint probability that Verify returns accept and the election result is not correct is at most $\delta(\ell)$.*

The election schema S is said to be verifiable *if δ is negligible.*

Condition (1) of Definition 5 expresses a fairness condition (see Section 2.3), where to guarantee the successful (and correct) run of a protocol, it is enough to only assume that t trustees are honest.

Condition (2) of Definition 5 is the core of Definition 5. Roughly speaking, it corresponds to Definition 1 with the goal γ_0 defined by Küsters et al. (see Section 5.2.2). As discussed below, there are, however, subtle differences, resulting in a too strong definition.

5.3.3 Discussion

As mentioned before, Benaloh's definition constitutes the first formal verifiability definition, mainly envisaging an entirely computer-operated process based on trusted machines and where, for example, voters were not asked to perform any kind of verification. Given this setting, the definition has some limitations from a more modern point of view.

Similarly to the definition in Section 5.2, this definition is fairly simple and general, except that only yes/no-votes are allowed, trustees are explicitly required in this definition, and *every* participant has her own bulletin board. These restrictions, however, are not necessary in order to define verifiability, as illustrated in Section 5.2. This definition also focuses on static corruption. The main problem with this definition is that it is too strong in settings typically considered nowadays, and hence, it would exclude most e-voting protocols, even those that intuitively should be considered verifiable.

As already mentioned, Condition (2) of Definition 5 is related to the goal γ_0. The goal γ_0 is, however, typically too strong because, for example, not all honest voters perform the verification process, e.g., check whether their ballots actually appear on the bulletin board. Hence, there is a non-negligible chance that the adversary is not caught when dropping or manipulating ballots. This is why Küsters et al. (Section 5.2) considered relaxed goals γ_k for $k \geq 0$.

Moreover, the goal considered here is even stronger (see Section 5.3.4). Condition (2) in Definition 5 is concerned not only with honest voters, but also with dishonest ones who post messages consistent with honest programs. Now, the problem is that a dishonest voter could simply cast a vote just like an honest one. The dishonest voter may, however, never complain even if dishonest trustees (who might even team up with the dishonest voter) drop or manipulate the ballot of the dishonest voter. Hence, it cannot be guaranteed that votes of such dishonest voters are counted, unlike what Condition (2) in Definition 5 requires. So, Definition 5 would deem almost all e-voting protocols in settings typically considered nowadays insecure, even completely reasonable ones.

Also, Condition (1) of Definition 5 may be too strong in many cases. It says that the threshold of t trustees is enough to guarantee that a protocol run is correct, i.e., in terms of the KTV framework, the judge would accept the run. It might not always be possible to resolve disputes, for example, when voters complain (possibly for no reason). For the sake of generality of the definition, it would therefore be better to allow for a more flexible fairness condition, as the one sketched in Section 5.2.

5.3.4 Casting in the KTV Framework

We now cast Definition 5 in the KTV Framework. To this end, we have to define the class of protocols considered in [Ben87] in terms of the KTV Framework and the goal γ.

Protocol P_B. The set of agents Σ consists of the voters, the trustees, the judge J, one bulletin board for each of these participants, and the remaining participants. Since static corruption is considered, the agents accept a $\mathsf{corrupt}$ message only at the beginning of an election run. The bulletin boards and the judge do not accept $\mathsf{corrupt}$ message at all. As usual, we consider an additional honest party, the scheduler. The honest programs are defined as follows:

- The scheduler behaves in the expected way: it triggers all the parties in every protocol step. The judge is triggered in the final phase, after the trustees are supposed to output their (partial) tallying.
- The honest behavior of the bulletin boards is as described in Section 2.1, with the only difference that a bulletin board owned by some party accepts messages posted only by this party; it serves its content to all parties, though.
- When a voter V runs her honest program π_{V}, she first expects "yes" or "no" as input (if the input is empty, she stops). If the input is "yes", she runs π_{yes}, and otherwise π_{no}. She sends the result to her bulletin board $\mathsf{B}(\mathsf{V})$; π_{V} might later be triggered again to perform verification steps.
- When the judge J runs π_{J} and is triggered in the final phase, it reads the content of all the bulletin boards and computes the result of the function Verify on this content.
- The honest program π_{T} of T depends on the concrete election system that is used.

The goal. We define the goal γ_0^* to be γ_0 (see Definition 4), with the difference that, instead of considering the multiset $\{\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}}\}$ of choices of honest voters only, we now consider the multiset of choices of all voters who cast a valid vote. This, as explained, includes not only honest voters, but might also include some dishonest voters.

Verifiability. Now, it should be clear that the notion of verifiability defined by Benaloh can be characterized in terms of Definition 1 as (γ_0^*, δ)-

verifiability.[1] As discussed before, the goal γ_0^* is too strong for several reasons.

5.4 E2E Verifiability by Kiayias et al.

In this section, we study the end-to-end verifiability definition by Kiayias et al. [KZZ15b, KZZ15a].

5.4.1 Model

According to Kiayias et al., an e-voting scheme Π is a tuple (Setup, Cast, Tally, Result, Verify) of probabilistic polynomial-time (ppt) algorithms where Cast and Tally are interactive. The entities are the election authority Auth, the bulletin board B, the trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ and the voters. The algorithm Cast is run interactively between B and a voter V_i where the voter operates a *voter supporting device* VSD on the following inputs: public parameters $\mathsf{prm}_{\mathsf{pub}}$, her choice ch_i, and her credentials cred_i. Upon successful termination, V_i obtains a receipt α_i. The algorithm Tally is run between Auth, the trustees and B. This computation updates the public transcript τ. The algorithm $\mathsf{Verify}(\tau, \alpha_i)$ denotes the individual verification of the public transcript τ by voter V_i, while $\mathsf{Verify}(\tau, \mathsf{st}_i)$ denotes the verification of τ by trustee T_i on her private state st_i; the output of Verify is a bit. The algorithm Setup is run for setting up an election, and the algorithm Result, given τ, outputs the result of the election, if any.

5.4.2 E2E Verifiability

The E2E-verifiability game by Kiayias et al. [KZZ15b, KZZ15a] is given in Figure 5.1. The adversary A can corrupt voters and trustees, and he controls Auth and the VSDs of voters. The bulletin board is assumed to be honest, but the adversary can determine the content τ of it. The set $\mathsf{V}_{\mathsf{cast}}$ contains all voters who successfully terminated their protocol, and hence, obtained a receipt. However, they might not have verified their receipts. The adversary wins the game if (i) $|\mathsf{V}_{\mathsf{cast}}| \geq \theta$, i.e., not to few voters successfully terminated, and (ii) if all of these voters verified their receipt, would verify successfully, and (iii) the published result of the election $\mathsf{Result}(\tau)$ deviates by at least k from the actual result $f_{\mathsf{res}}(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}})$ obtained according to the actual votes of voters. More specifically, for the last condition, i.e., Condition (iii), Kiayias et al. postulates the existence of a vote extractor algorithm Extr (not

[1] Recall that here we do not consider the fairness conditions.

necessarily running in polynomial time) which is supposed to determine the votes of all voters not in $\mathsf{V}_{\mathsf{cast}}$, where Extr is given the transcript and the receipt of voters in $\mathsf{V}_{\mathsf{cast}}$ as input. Note that the adversary wins the game if Extr fails to return these votes (Condition (b)).

Definition 6 (E2E-verifiability)**.** *Let* $n_{\mathsf{voters}}, n_{\mathsf{choices}}, k, n_{\mathsf{trustees}}, \theta \in \mathbb{N}$ *with* $k > 0$ *and* $0 < \theta \leq n_{\mathsf{voters}}$*. Let* $0 < \delta < 1$*. The election protocol* Π *w.r.t. election function* f_{res} *achieves E2E verifiability with error* δ*, for a number of at least* θ *honest successful voters and tally deviation* k *if there exists a vote-extractor* Extr *such that for any adversary* A *controlling less than* $n_{\mathsf{voters}} - \theta$ *voters and* n_{trustees} *trustees, the* Auth *and all* VSD*'s holds:*

$$\Pr\left[G^{\mathsf{A},\mathsf{Extr},k,\theta}(1^{\ell}, n_{\mathsf{choices}}, n_{\mathsf{voters}}, n_{\mathsf{trustees}}) = 1\right] \leq \delta.$$

We note that [KZZ15b] considers a fairness condition (named *perfect correctness)* similarly to the one in Section 2.3.

5.4.3 Discussion

We first note that the definition is too specific in some situations due to the use of the extractor in the definition. Indeed, it does not seem to apply to voting protocols where ballots published on the bulletin board hide the choices of voters information-theoretically, such as [CPP13]. In this case, the adversary could, for example, corrupt some voters but just follow the protocol honestly. For these voters and those in $\mathsf{V}_{\mathsf{cast}}$ the extractor could not determine their votes, and hence, it would be very likely that the adversary wins the game in Figure 5.1: if the extractor outputs votes, then it would be very likely that Condition (a) is satisfied, and otherwise Condition (b) would be satisfied.

This problem can be fixed by providing the extractor with the votes of the voters in $\mathsf{V}_{\mathsf{cast}}$, not only with their receipts. In this case, the extractor could simply compute $\mathsf{Result}(\tau)$ and choose $(\mathsf{ch}_i)_{\mathsf{V}_i \notin \mathsf{V}_{\mathsf{cast}}}$ such that the distance $\mathrm{d}_1(\mathsf{Result}(\tau), f_{\mathsf{res}}(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}}))$ is minimal. This would be the best extractor, i.e., the one that makes it the hardest for the adversary to win the game. Note that this extractor does not have to actually extract votes from τ, or even look closely at τ, except for computing $\mathsf{Result}(\tau)$.

Conditions (a) and (b) could therefore be replaced by the following one:

(a*) For any combination of choices $(\mathsf{ch}_i)_{\mathsf{V}_i \notin \mathsf{V}_{\mathsf{cast}}}$:

$$\mathrm{d}_1(\mathsf{Result}(\tau), f_{\mathsf{res}}(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}})) \geq k.$$

E2E Verifiability Game $G^{\mathsf{A},\mathsf{Extr},k,\theta}(1^\ell, n_{\mathsf{choices}}, n_{\mathsf{voters}}, n_{\mathsf{trustees}})$

1. A chooses a list of choices $\mathsf{C} = \{\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{choices}}}\}$, a set of voters $\{\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}\}$, and a set of trustees $\{\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}\}$. It provides the challenger Ch with these sets along with information $\mathsf{prm}_{\mathsf{pub}}$ and voter credentials $\{\mathsf{cred}_i\}_{1 \leq i \leq n_{\mathsf{voters}}}$. Throughout the game, Ch plays the role of B.

2. A and Ch engage in an interaction where A schedules the Cast protocols of all voters. For each voter V_i, A can either completely control the voter or allow Ch operate on V_i's behalf, in which case A provides ch_i to Ch. Then, Ch engages in the Cast protocol with the adversary A, so that A plays the roles of Auth and VSD. Provided the protocol terminates successfully, Ch obtains a receipt α_i on behalf of V_i.

3. Finally, A posts the election transcript τ to B.

The game returns a bit which is 1 if the following conditions hold true:

1. $|\mathsf{V}_{\mathsf{cast}}| \geq \theta$, (i.e., at least θ honest voters terminated)

2. $\forall \mathsf{V}_i \in \mathsf{V}_{\mathsf{cast}} \colon \mathsf{Verify}_{\mathsf{V}}(\tau, \alpha_i) = 1$ (i.e. the honest voters that terminated verified successfully)

and either one of the following two conditions:

   (a) If $\perp \neq (\mathsf{ch}_i)_{\mathsf{V}_i \notin \mathsf{V}_{\mathsf{cast}}} \leftarrow \mathsf{Extr}(\tau, \{\alpha_i\}_{\mathsf{V}_i \in \mathsf{V}_{\mathsf{cast}}})$, then $\mathrm{d}_1(\mathsf{Result}(\tau), f_{\mathsf{res}}(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}})) \geq k$ (where d_1 is a metric).

   (b) $\perp \leftarrow \mathsf{Extr}(\tau, \{\alpha_i\}_{\mathsf{V}_i \in \mathsf{V}_{\mathsf{cast}}})$

Figure 5.1: E2E-verifiability by Kiayias et al.

This is then similar to Definition 4 where votes of dishonest voters are quantified existentially. (Note that (a)* talks about when verifiability is broken, while Definition 4 talks about the goal, i.e., what verifiability should achieve, hence the switch from existential quantification in Definition 4 to universal quantification in (a)*). As explained in Section 5.2, the existential quantification is very reasonable because, for several reasons, it is often not possible to extract votes of dishonest voters.

Our second observation is that the definition (even the version with the fix above) is too weak in the following sense. To see this, consider runs

where honest voters cast their votes successfully, and hence, obtain a receipt, but do not verify their receipt, and where the verification would even fail. Because of the second condition, the adversary would right away loose the game in these runs. However, these runs are realistic threats (since often voters do not verify), and hence, guarantees should be given even for such runs. The game in Figure 5.1 simply discards such runs. Therefore, instead of the second condition one should simply require that the judge (looking at τ and waiting for complaints from voters, if any) accepts the run. Note that if the judge does not accept the run, then the election is invalid.

5.4.4 Casting in the KTV Framework

Protocol P_{KZZ}. The set of agents Σ consists of the voters, the bulletin board B, the voting authority Auth, the judge J, the trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ and the remaining participants.

When a voter V runs her honest program π_{V} in the casting phase, she expects a choice ch, a credential and the public parameters of the election (if her input is empty, she stops). Then, she runs Cast in interaction with B, and expects a receipt α (if she does not receive a receipt, she stops). When the voter is triggered by the judge in the verification phase, the voter reads the election transcript τ from the bulletin board B (if she does not receive τ, she outputs "reject") and runs $\mathsf{Verify}(\tau, \alpha)$. If $\mathsf{Verify}(\tau, \alpha)$ evaluates to "false" or "true", respectively, she sends "reject" or "accept" to the judge J. The definition of Kiayias et al. is not explicit about whether voters always verify when triggered or not. Here one could also model that they decide whether they verify according to some probability distribution.

When a trustee T runs its honest program π_{T} in the setup phase, it interacts with the remaining trustees, Auth and B. It expects as output its secret state st (otherwise, it stops). In the tallying phase, on input st and the contents of B (if any input is empty, it stops), it runs Tally in interaction with B and Auth, and outputs a partial tally τ that is sent to Auth.

When the election authority Auth runs its honest program π_{Auth}, it expects a security parameter 1^ℓ in the setup phase (if the input is empty, it stops). Then, it runs Setup in interaction with B and the trustees, and outputs the election parameters, which are published in B, and the voters' credentials $(\mathsf{cred}_1, \ldots, \mathsf{cred}_{n_{\mathsf{voters}}})$, which are sent to the corresponding voters $(\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}})$. In the tallying phase, Auth runs Tally in interaction with B and the trustees, and publishes the partial tally data $\tau_1, \ldots, \tau_{n_{\mathsf{trustees}}}$ produced by each trustee at the end of the interaction.

When the judge J runs its honest program π_{J} and is triggered in the

verification phase, it reads the election transcript τ. It performs whatever check prescribed by the protocol. If one of these checks fails, J outputs "reject". Otherwise, J iteratively triggers all voters and asks about their verification results (if any). If one of the voters rejects, J outputs "reject", and otherwise, "accept".

E2E verifiability. We define the goal $\gamma_{\theta,k,\mathsf{Extr}}$, which is parameterized by θ, k, and Extr as in Figure 5.1, to be the set of runs of P_{KZZ} (with some adversary A) such that at least one of the Conditions in Figure 5.1 is not satisfied. With this, Definition 6, corresponds to the notion of $(\gamma_{\theta,k,\mathsf{Extr}}, \delta)$-verifiability according to Definition 1 when the same extractors are used and one quantifies over the same set of adversaries.

As already discussed above, this definition on the one hand is too specific (due to the use of the extractor) and on the other hand too weak (due to the second condition). Therefore, as mentioned, the definition would be improved if Conditions (a) and (b) were replaced by (a)* and the second condition was replaced by the condition that the judge accepts the run. If one set $\theta = 0$ in addition, then Definition 6 would closely resemble γ_k from Definition 4.

5.5 Computational Election Verifiability by Cortier et al.

In this section, we study the verifiability definition by Cortier et al. [CGGI14], which can be seen as an extension of a previous verifiability definition by Catalano et al. [JCJ10], whereby the bulletin board may act maliciously, and thus it could potentially perform ballot stuffing (i.e., stuff itself with self-made ballots on behalf of voters who did not vote) or erase ballots previously cast by voters.

5.5.1 Model

Cortier et al. [CGGI14] model an e-voting scheme Π as a tuple (Setup, $\mathsf{Credential}$, Vote,$\mathsf{Verify}_\mathsf{V}$,$\mathsf{Valid}$, Board, Tally, Verify) of ppt algorithms where $\mathsf{Verify}_\mathsf{V}$ and Verify are non-interactive. The entities are the registrar Reg, the bulletin board B, the trustee T and the voters. The algorithm $\mathsf{Setup}(\ell)$ is run by the trustee T and outputs the public parameters of the election $\mathsf{prm}_\mathsf{pub}$ and the secret tallying key sk. The procedure $\mathsf{Credential}$ is run by Reg with the identity id_i of voter V_i and outputs a public/secret credential pair $(\mathsf{upk}_i, \mathsf{usk}_i)$. The algorithms discussed next implicitly take $\mathsf{prm}_\mathsf{pub}$ as input. The algorithm Vote is run interactively between B and a voter V_i who

receives as inputs $\mathsf{prm}_{\mathsf{pub}}$, her choice ch_i and her credentials $(\mathsf{upk}_i, \mathsf{usk}_i)$. Upon successful termination, a ballot b_i is appended to the public transcript τ of the election. The procedure $\mathsf{Valid}(\mathsf{b})$ outputs 1 or 0, depending on whether b is well-formed. The algorithm that B must run to update τ is denoted by Board. The algorithm Tally is run at the end of the election by T, given the content of B and the secret key sk as input, and outputs tallying proofs π and the final election result, denoted by res. The algorithm $\mathsf{Verify}_{\mathsf{V}}(\tau, \mathsf{upk}_i, \mathsf{usk}_i, \mathsf{b}_i)$ is run by voter V_i for checking whether or not ballot b_i appears in τ. The algorithm $\mathsf{Verify}(\tau, \mathsf{res}, \pi)$ denotes the verification of the election result, while $\mathsf{Verify}_{\mathsf{V}}(\tau, \mathsf{upk}_i, \mathsf{b}_i)$ denotes the verification that ballot b_i from voter V_i was included in the final transcript of the election as published by B.

5.5.2 Verifiability Against Malicious Bulletin Board

In the e-voting system Helios [Adi08], a dishonest bulletin board B may add ballots, since it is the sole entity checking the eligibility of voters. If B is corrupted, then it might stuff itself with ballots on behalf of voters that in fact did not vote. This problem, as already mentioned in Section 5.2.2, is called *ballot stuffing*. Cortier et al. [CGGI14] give a definition of verifiability in the computational model to account for a malicious bulletin board. To defend voters against a dishonest B, a registration authority Reg is required. Depending on whether both B and Reg are required to be honest, Cortier et al. [CGGI14] define *weak verifiability* (B *and* Reg are honest) or *strong verifiability* (B *or* Reg are honest).

In Figure 5.2, we give a snapshot of the cryptographic game used in [CGGI14] to define verifiability in case B is dishonest. The adversary has oracles to register voters, corrupt voters, and let honest voters vote. The condition for winning the game is explained below. Note that Cortier et al. assume that the result function admits *partial counting*, namely $f_{\mathsf{res}}(S_1 \cup S_2) = f_{\mathsf{res}}(S_1) \star_{\mathsf{R}} f_{\mathsf{res}}(S_2)$ for any two lists S_1, S_2 containing sequences of elements $\mathsf{ch} \in \mathsf{C}$ and where $\star_{\mathsf{R}} \colon \mathsf{R} \times \mathsf{R} \to \mathsf{R}$ is a commutative operation. For example, the result function that counts the number of votes per candidate admits partial counting, while the result function that only reveals the winner or overall ranking does not admit partial counting.

Definition 7 (Verifiability against malicious bulletin board)**.** *An election scheme achieves* verifiability against the bulletin board *if the success rate* $\mathsf{Succ}(\mathsf{Exp}^{\mathsf{verb}}_{\mathsf{A},\Pi}) = \Pr[\mathsf{Exp}^{\mathsf{verb}}_{\mathsf{A},\Pi}(\ell) = 1]$ *of any ppt adversary* A *is negligible as a function of* ℓ*, where* $\mathsf{Exp}^{\mathsf{verb}}_{\mathsf{A},\Pi}$ *is defined as in Figure 5.2.*

Roughly speaking, this definition declares a protocol verifiable if, in the presence of a malicious bulletin board (which can erase previous cast ballots

Experiment $\mathsf{Exp}^{\mathsf{verb}}_{\mathsf{A},\Pi}$

Adversary A has access to the following oracles:

- $\mathcal{O}\mathsf{reg}(id)$ which creates voters' credentials via $(\mathsf{upk}_{id}, \mathsf{usk}_{id}) \leftarrow \mathsf{Credential}(id)$, stores them as $\mathcal{U} \leftarrow \mathcal{U} \cup \{(id, \mathsf{upk}_{id}, \mathsf{usk}_{id})\}$, and returns upk_{id} to the attacker.
- $\mathcal{O}\mathsf{corrupt}(id)$ which checks if an entry $(\mathsf{id}, *, *)$ appears in $\mathcal{U}$; if not, it stops. Else, it gives $(\mathsf{upk}_{id}, \mathsf{usk}_{id})$ to A, updates a list of corrupted voters $\mathcal{CU} \leftarrow \mathcal{CU} \cup \{(\mathsf{id}, \mathsf{upk}_{id})\}$ and the list of honest cast ballots HVote by removing any occurrence $(id, *, *)$.
- $\mathcal{O}\mathsf{vote}(id, \mathsf{ch})$ which aborts if $(\mathsf{id}, *, *) \notin \mathcal{U}$, or $(\mathsf{id}, *) \in \mathcal{CU}$. Else, it returns $\mathsf{b} \leftarrow \mathsf{Vote}(\mathsf{id}, \mathsf{upk}_{id}, \mathsf{usk}_{id}, \mathsf{ch})$ and replaces any previous entry $(id, *, *)$ in HVote with $(\mathsf{id}, \mathsf{ch}, b)$. (The latter list is used to record the voter's intention.)

Let $\mathsf{Checked} = \{(\mathsf{id}_1^E, \mathsf{ch}_1^E, b_1^E), \ldots, (\mathsf{id}_{n_E}^E, \mathsf{ch}_{n_E}^E, b_{n_E}^E)\} \subseteq \mathsf{HVote}$ contain those id's who checked that their ballot appears in τ at the end of the election. The experiment outputs a bit as follows:

1. $(\tau, \mathsf{res}, \pi) \leftarrow \mathsf{A}^{\mathcal{O}\mathsf{reg}, \mathcal{O}\mathsf{corrupt}, \mathcal{O}\mathsf{vote}}$
2. If $\mathsf{Verify}(\tau, \mathsf{res}, \pi) = \mathsf{reject}$, return 0.
3. If $\mathsf{res} = \perp$, return 0.
4. If there exist $(\mathsf{id}_1^A, \mathsf{ch}_1^A, *), \ldots, (\mathsf{id}_{n_A}^A, \mathsf{ch}_{n_A}^A, *) \in \mathsf{HVote} \setminus \mathsf{Checked}$ and there exist $\mathsf{ch}_1^B, \ldots, \mathsf{ch}_{n_B}^B \in \mathsf{C}$ such that $0 \leq n_B \leq |\mathcal{CU}|$ and

$$\mathsf{res} = \rho\left(\{\mathsf{ch}_i^E\}_{i=1}^{n_E}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^A\}_{i=1}^{n_A}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^B\}_{i=1}^{n_B}\right),$$

   return 0.
5. Return 1.

Figure 5.2: Verifiability against bulletin board by Cortier et al. [CGGI14]

and/or cast ballots on behalf of absentee voters), voters who check that their ballot has not been removed are guaranteed that their choice has been counted in the final result. Also some of the votes of honest voters who did not check might also be contained in the final result. However, their votes

Experiment $\mathsf{Exp}^{\mathsf{verg}}_{\mathsf{A},\Pi}$

Adversary A has access to the oracles $\mathcal{O}\mathsf{vote}, \mathcal{O}\mathsf{corrupt}$ as in Figure 5.2, and additionally to

- $\mathcal{O}\mathsf{cast}(id, \mathsf{b})$ which runs $\mathsf{Board}(\tau, \mathsf{b})$. ($\mathcal{O}\mathsf{cast}$ allows A to cast ballots to B on behalf of corrupted voters.)

Let HVote and $\mathsf{Checked}$ be the lists defined in Figure 5.2. The experiment outputs a bit as follows:

1. $(\mathsf{res}, \pi) \leftarrow \mathsf{A}^{\mathcal{O}\mathsf{cast},\mathcal{O}\mathsf{corrupt},\mathcal{O}\mathsf{vote}}$
2. If $\mathsf{Verify}(\tau, \mathsf{res}, \pi) = \mathsf{reject}$, return 0.
3. If $\mathsf{res} = \perp$, return 0.
4. If there exist $(\mathsf{id}_1^A, \mathsf{ch}_1^A, *), \ldots, (\mathsf{id}_{n_A}^A, \mathsf{ch}_{n_A}^A, *) \in \mathsf{HVote} \setminus \mathsf{Checked}$ and there exist $\mathsf{ch}_1^B, \ldots, \mathsf{ch}_{n_B}^B \in \mathsf{C}$ such that $0 \leq n_B \leq |\mathcal{CU}|$ and
$$\mathsf{res} = \rho\left(\{\mathsf{ch}_i^E\}_{i=1}^{n_E}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^A\}_{i=1}^{n_A}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^B\}_{i=1}^{n_B}\right),$$
return 0.
5. Return 1.

Figure 5.3: Verifiability against registrar by Cortier et al. [CGGI14]

may as well have been dropped (but not altered to other votes). Voters under adversarial control can only vote once, with choices belonging to the choice space. The bulletin board cannot stuff itself with additional ballots without getting caught.

5.5.3 Verifiability Against Malicious Registrar

In Helios, the bulletin board B accepts only ballots cast by eligible voters. The bulletin board B can tell apart eligible from ineligible voters generally by using some kind of authentication mechanism. In this situation, one might hope to enjoy verifiability against a dishonest registrar Reg, which is defined in Figure 5.3.

Definition 8 (Verifiability against malicious registrar)**.** *An election scheme achieves* verifiability against the registrar *if the success rate* $\mathsf{Succ}(\mathsf{Exp}^{\mathsf{verg}}_{\mathsf{A},\Pi}) =$

$\Pr[\mathsf{Exp}^{\mathsf{verg}}_{\mathsf{A},\Pi}(\ell) = 1]$ *of any ppt adversary* A *is negligible as a function of* ℓ*, where* $\mathsf{Exp}^{\mathsf{verg}}_{\mathsf{A},\Pi}$ *is defined as in Figure 5.3.*

The intuition behind and the guarantees provided by Definition 8 are similar to those of Definition 7 except that instead of a malicious bulletin board a malicious registrar is considered, which thus can handle credentials for voters in a malicious way, i.e., provide invalid credentials or make several users share the same credentials.

5.5.4 Strong Verifiability

A protocol is said to have *strong verifiability* if it enjoys verifiability against a dishonest registrar and verifiability against a dishonest bulletin board. Intuitively, this allows one to give verifiability guarantees under a weaker trust assumption than that used in Section 5.4 since for strong verifiability we do not need the bulletin board and the registrar to be simultaneously honest; in Section 5.3, it was assumed that every party has its own bulletin board, and in Sections 5.2, no specific trust assumptions were fixed or assumed.

We note that Cortier et al. also consider a fairness (correctness) condition similar to the ones mentioned above: the result corresponds to the votes of honest voters whenever all parties $\mathsf{Reg}, \mathsf{T}, \mathsf{B}$ and the voters are honest.

5.5.5 Weak Verifiability

For weak verifiability, the trust assumptions are stronger: both the registrar Reg and the board B are assumed to be honest. This means, in particular, that B does not remove ballots, nor stuffs itself; and that Reg faithfully distributes credentials to the eligible voters. The verifiability game is given in Figure 5.4.

Intuitively, weak verifiability guarantees that all votes that have been successfully cast are counted and that dishonest voters can only vote once; additionally only choices belonging to the choice space can be cast and counted.

5.5.6 Tally Uniqueness

As part of their definitional framework, Cortier et al. [CGGI14] and Juels et al. [JCJ10], require the notion of *tally uniqueness*. Roughly speaking, tally uniqueness of a voting protocol ensures that the tally of an election is unique, even if all the players in the system are malicious.

More formally, the goal of the adversary against tally uniqueness is to output public election parameters $\mathsf{prm}_{\mathsf{pub}}$, a public transcript τ, two results

Experiment $\mathsf{Exp}^{\mathsf{verw}}_{\mathsf{A},\Pi}$

Adversary A has access to the oracles $\mathcal{O}\mathsf{vote}, \mathcal{O}\mathsf{corrupt}, \mathcal{O}\mathsf{reg}$ and $\mathcal{O}\mathsf{cast}$ defined Figure 5.3 and 5.2. Let HVote the list containing the intended choices of the honest voters. The experiment outputs a bit as follows:

1. $(\mathsf{res}, \pi) \leftarrow \mathsf{A}^{\mathcal{O}\mathsf{cast},\mathcal{O}\mathsf{corrupt},\mathcal{O}\mathsf{vote},\mathcal{O}\mathsf{reg}}$
2. If $\mathsf{Verify}(\tau, \mathsf{res}, \pi) = \mathsf{reject}$, return 0.
3. If $\mathsf{res} = \perp$, return 0.
4. If there exist $(\mathsf{id}_1^A, \mathsf{ch}_1^A, *), \ldots, (\mathsf{id}_{n_A}^A, \mathsf{ch}_{n_A}^A, *) \in \mathsf{HVote} \setminus \mathsf{Checked}$ and there exist $\mathsf{ch}_1^B, \ldots, \mathsf{ch}_{n_B}^B \in \mathsf{C}$ such that $0 \leq n_B \leq |\mathcal{CU}|$ and
$$\mathsf{res} = \rho\left(\{\mathsf{ch}_i^A\}_{i=1}^{n_A}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^B\}_{i=1}^{n_B}\right),$$
return 0.
5. Return 1.

Figure 5.4: Weak verifiability by Cortier et al. [CGGI14]

$\mathsf{res} \neq \mathsf{res}'$, and corresponding proofs of valid tallying π and π' such that both pass verification, i.e. $\mathsf{Verify}(\tau, \mathsf{res}, \pi) = \mathsf{Verify}(\tau, \mathsf{res}', \pi') = 1$. A voting protocol Π has *tally uniqueness* if every ppt adversary A has a negligible advantage in this game.

Following [CGGI14], tally uniqueness ensures that, given a tally, there is at most one plausible instantiation (one-to-one property).

5.5.7 Discussion

Strong verifiability explicitly captures the situation where key players in an electronic election, such as the bulletin board or the registrar, might be corrupted and willing to alter the legitimate operation of the election. This is notably the case for Helios without identifiers (i.e., the transcript τ does not contain voters' identifiers), where a malicious B can stuff itself with ballots on behalf of absentee voters. Additionally, strong verifiability provides stronger guarantees, compared to previous definitions, to honest voters: ballots from honest voters that do not verify successfully at the end of the election can at worst be removed from the election's announced result, but *never changed*. In [CGGI14], sufficient properties for proving strong verifiability have been established.

A downside of the above definitions is that the voter's intent *is not* captured by the oracle $\mathcal{O}\mathsf{vote}(id, \mathsf{ch})$, as this oracle simply performs the honest voting algorithm. In reality, voters typically use some VSD, which might be corrupted. Additionally, since Cortier et al. require that the adversary wins the game (i.e., successfully cheats) with at most negligible probability, ballot audit checks, such as Benaloh's audits[2] [Ben06], are deemed non-verifiable as these checks may fail with non-negligible probability. Another weak point, although less important than the previous ones, is that this framework assumes that the result function ρ admits partial tallying, which is commonly the case, but it is, for example, not applicable to voting protocols which use the majority function as the result function.

5.5.8 Casting in the KTV Framework

Protocol P_{CGGI}. The set of agents Σ consists of the voters, the bulletin board B, the registrar Reg, the teller T, judge J, the scheduler, and the remaining participants. We assume that the judge and the scheduler cannot be corrupted (they ignore the $\mathsf{corrupt}$ message). As in the definition of Cortier et al., Reg and B can be corrupted statically, i.e., they accept the $\mathsf{corrupt}$ message at the beginning of a run only. Voters can be corrupted dynamically.

When the voter V runs her honest program π_{V}, she expects a choice ch, a credential pair $(\mathsf{upk}, \mathsf{usk})$ as input (if the input is empty, she stops). After that, she reads the election parameters $\mathsf{prm}_{\mathsf{pub}}$ and C from the bulletin board B (if she cannot find any election paramaters on B, she stops). Then, she runs $\mathsf{Vote}(\mathsf{prm}_{\mathsf{pub}}, \mathsf{ch}, \mathsf{upk}, \mathsf{usk})$ and sends the result b to the bulletin board B. Once the election is closed, she reads the content of the bulletin board and checks whether her ballot has been properly handled by the bulletin board by running $\mathsf{Verify}_{\mathsf{V}}(\tau, \mathsf{upk}, \mathsf{usk}, \mathsf{b})$. If not, the voter sends her complaint to the judge. The program of the judge accepts a run, if it does not receive any complaint from a voter and the procedure $\mathsf{Verify}(\tau, \mathsf{res}, \pi)$ returns 1.

When the registrar Reg runs the honest program π_{Reg}, it generates and distributes secret credentials to voters and registers the corresponding public credentials in the bulletin board.

When the teller T runs its honest program π_{T}, it reads the public transcript τ and runs $(\mathsf{res}, \pi) \leftarrow \mathsf{tally}(\tau, \mathsf{sk})$, with the election private key sk. The transcript is updated to $\tau' = \tau||\mathsf{res}||\pi$.

Strong verifiability. We define the goal γ_{SV} to be the set of all runs of P_{CGGI} in which either (a) both Reg and B are corrupted, (b) the result is not

[2] In these audits the voter can decide to cast or to audit a ballot created by her VSD. If she decides to audit the ballot, she can check whether it actually encodes her choice.

output, or (c) the result res of the election is defined and satisfies

$$\mathsf{res} = \rho\left(\{\mathsf{ch}_i^E\}_{i=1}^{n_E}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^A\}_{i=1}^{n_A}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^B\}_{i=1}^{n_B}\right)$$

for some n_E, n_A, n_B and some $\mathsf{ch}_i^E, \mathsf{ch}_i^A, \mathsf{ch}_i^B$ such that

- $\mathsf{ch}_1^E, \ldots, \mathsf{ch}_{n_E}^E$ are the choices read by honest voters that successfully checked their ballots at the end of the election (and report it to the judge),
- $w_1, \ldots, w_{m_A}$ are the candidates read by honest voters that did not check their ballots and $\{\mathsf{ch}_i^A\}_{i=1}^{n_A} \subseteq \{w_j\}_{j=1}^{m_A}$,
- $\mathsf{ch}_1^B, \ldots, \mathsf{ch}_{n_b}^B \in \mathsf{C}$ and n_b is smaller then the number of voters running a dishonest program.

Note that, according to the above definition, if both the registrar and the bulletin board are corrupted, then the goal is trivially achieved, as we do not expect to provide any guarantees in this case.

For the protocol P_{CGGI}, strong verifiability by Cortier et al. can essentially be characterized by the fact that it is (γ_{SV}, δ)-verifiable by the judge J in the sense of Definition 1, for $\delta = 0$.

Let us emphasize that this goal ensures that votes of honest voters who do not verify at the end of the election are at most dropped, but not changed. This is in contrast to the goals we have seen so far. In these goals, votes of honest voters who do not verify might have been tampered with.

Weak verifiability. We define the goal γ_{WV} to be the set of all runs of P_{CGGI} in which either (a) either Reg or B is corrupted, (b) the result is not output, or (c) the result res of the election is defined and satisfies

$$\mathsf{res} = \rho\left(\{\mathsf{ch}_i^A\}_{i=1}^{n_A}\right) \star_{\mathsf{R}} \rho\left(\{\mathsf{ch}_i^B\}_{i=1}^{n_B}\right)$$

for some n_A, n_B and some $\mathsf{ch}_i^A, \mathsf{ch}_i^B$ such that

- $\mathsf{ch}_1^A, \ldots, \mathsf{ch}_{n_A}^A$ are the candidates read by honest voters that cast their votes;
- $\mathsf{ch}_1^B, \ldots, \mathsf{ch}_{n_b}^B \in \mathsf{C}$ and n_b is smaller then the number of voters running a dishonest program.

For the protocol P_{CGGI}, weak verifiability by Cortier et al. can essentially be characterized by the fact that it is (γ_{WV}, δ)-verifiable in the sense of Definition 1.

Note that Item (c) of the goal γ_{WV} is stronger than the corresponding item of γ_{SV} (since all honest cast votes shall be counted). However, the latter is called *weak verifiability* in [CGGI14] because the trust assumptions (Item (a)) are stronger (both the bulletin board and the registrar shall be honest).

5.6 Computational Election Verifiability by Smyth et al.

This section focuses on the definitions of individual, universal and election verifiability by Smyth et al. [SFC15]. Smyth et al. consider two different verifiability settings, one for election schemes with *external* and the other one for election schemes with *internal* authentication (such as Helios and Civitas, respectively). For the sake of brevity, we focus on election schemes with external authentication because the issues discussed in Section 5.6.5 apply to both of them.

5.6.1 Model

According to Smyth et al., an election scheme Π is a tuple (Setup, Vote, Tally, Verify) of probabilistic polynomial-time algorithms. The algorithms Setup and Vote are defined as usual. The algorithm Tally is run by the tellers and receives the content of the bulletin board B and the parameters prm as input, and outputs the tally along with a non-interactive proof π for the correctness of the tally. The algorithm Verify describes the verification of the election result and receives the content of the bulletin board B, the public parameters $\mathsf{prm}_{\mathsf{pub}}$, the tally, denoted by tally, and a proof π, and outputs a bit. The algorithm Verify is deterministic.

5.6.2 Individual Verifiability

According to Smyth et al., an election scheme achieves individual verifiability if, for any two honest voters, the probability that their ballots are equal is negligible, which formally is expressed as follows.

Definition 9 (Individual verifiability)**.** *An election scheme* Π *achieves* individual verifiability *if the success rate* $\mathsf{Succ}(\mathsf{ExpIV}(\Pi, \mathsf{A}))$ *of any ppt adversary* A *in Experiment* $\mathsf{ExpIV}(\Pi, \mathsf{A})$ *(Fig. 5.5) is negligible as a function of* ℓ*.*

5.6.3 Universal Verifiability

According to Smyth et al., an election scheme achieves universal verifiability if no ppt adversary A can simulate a tallying phase such that, on the one hand, the verification algorithm Verify accepts the output (e.g., all zero-knowledge proofs are successful), and, on the other hand, the given output of the tallying phase does not agree with what Smyth et al. call the *correct tally*.

Experiment $\mathsf{ExpIV}(\Pi, \mathsf{A})$

1. $(\mathsf{prm}_{\mathsf{pub}}, \mathsf{ch}, \mathsf{ch}') \longleftarrow \mathsf{A}$
2. $\mathsf{b} \longleftarrow \mathsf{Vote}(\mathsf{ch}, \mathsf{prm}_{\mathsf{pub}})$
3. $\mathsf{b}' \longleftarrow \mathsf{Vote}(\mathsf{ch}', \mathsf{prm}_{\mathsf{pub}})$
4. If $\mathsf{b} = \mathsf{b}' \neq \perp$, return 1, else return 0.

Figure 5.5: Individual verifiability experiment by Smyth et al. [SFC15]

Experiment $\mathsf{ExpUV}(\Pi, \mathsf{A})$

1. $(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}}, \mathsf{tally}', \pi') \leftarrow \mathsf{A}$
2. $\mathsf{tally} \longleftarrow \mathsf{correct\ tally}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}})$
3. If $\mathsf{tally} \neq \mathsf{tally}'$ and $\mathsf{Verify}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}}, \mathsf{tally}', \pi')$, then return 1, else return 0.

Figure 5.6: Universal verifiability experiment by Smyth et al. [SFC15]

The function $\mathsf{correct\ tally}$, defined as follows, extracts the actual votes from the ballots on the bulletin board.

Definition 10 (Correct Tally)**.** *The function* $\mathsf{correct\ tally}$ *receives as input a tuple* $(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}})$ *and outputs a vector from* $\{0, \ldots, \mathsf{n}_{\mathsf{ballots}}\}^{\mathsf{n}_{\mathsf{cand}}}$ *such that for every choice* $\mathsf{ch} \in \{1, \ldots, \mathsf{n}_{\mathsf{cand}}\}$ *and every number* $l \in \{0, \ldots, \mathsf{n}_{\mathsf{ballots}}\}$*, we have that* $\mathsf{correct\ tally}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}})[\mathsf{ch}] = l$ *if and only if there are exactly* l *different ballots* b $(\neq \perp)$ *on the bulletin board* B *and for each of them there exists a random bit string* r *such that* $\mathsf{b} = \mathsf{Vote}(\mathsf{ch}, \mathsf{prm}_{\mathsf{pub}}; r)$*.*

Now, universal verifiability is defined as follows according to Smyth et al.

Definition 11 (Universal verifiability)**.** *An election scheme* Π *achieves* universal verifiability *if the success rate* $\mathsf{Succ}(\mathsf{ExpUV}(\Pi, \mathsf{A}))$ *of any ppt adversary* A *in Experiment* $\mathsf{ExpUV}(\Pi, \mathsf{A})$ *(Fig. 5.6) is negligible as a function of* ℓ*.*

5.6.4 Election Verifiability

The notion of verifiability proposed by Smyth et al. now combines the notions of individual and universal verifiability.

Definition 12 (Election Verifiability)**.** *An election scheme* Π *satisfies* election verifiability *if for any ppt adversaries* A*, there exists a negligible function* μ *such that for all security parameters* ℓ*, we have that*

$$\mathsf{Succ}(\mathsf{ExpIV}(\Pi, \mathsf{A})) + \mathsf{Succ}(\mathsf{ExpUV}(\Pi, \mathsf{A})) \leq \mu(\ell).$$

Smyth et al. also consider some soundness properties, including fairness and correctness, similar to the ones mentioned in previous sections.

5.6.5 Discussion

This definition has two main shortcomings. First, as stated by the authors [SFC15], their "definitions of verifiability have not addressed the issue of voter intent, that is, whether the ballot constructed by the Vote algorithm corresponds to the candidate choice that a voter intended to make." In general, it is not clear that the combination of the proposed definitions of verifiability along with additional soundness properties implies any form of end-to-end verifiability. More precisely, if all the verification procedures succeed, it is unclear whether the final outcome of an election corresponds to the voters' choices at least with reasonable probability. We think, however, that capturing such overall correctness and the voter's intent is at the very core of a meaningful notion of verifiability.

Second, the definition considers a restricted class of protocols (the authors themselves provide a list of protocols excluded by their definition), some of these restrictions, as pointed out before, also apply to some of the other definitions discussed in this paper: (1) The model captures "single-pass" protocols only: voters send a single ballot to the election server, without any further interaction. (2) The authors assume that the whole ballot is published. (3) The authors assume that the vote can be recovered directly from the ballot, which excludes protocols using information-theoretically hiding commitments. (4) There is no revote. (5) The bulletin board publishes the list of ballots, as received. And hence, voting schemes such as ThreeBallot [Smi07] cannot be modeled.

5.6.6 Casting in the KTV Framework

Protocol P_{SFC}**.** The set of agents Σ consists of the voters, the bulletin board B, the judge J, the scheduler, and the remaining participants. Since static corruption is considered, the agents only accept the corrupt message at the beginning of an election run. The bulletin board and the judge do not accept to be corrupted.

When a voter V runs her honest program π_{V}, she expects a choice ch as input (if the input is empty, she stops). After that, she reads the public election parameters $\mathsf{prm}_{\mathsf{pub}}$ from the bulletin board B (if she does not receive any election paramaters on B, she stops). Then, she runs $\mathsf{Vote}(\mathsf{ch}, \mathsf{prm}_{\mathsf{pub}})$ and sends the resulting ballot b to the bulletin board B. Although this is kept implicit in the discussed paper, we will assume here that V subsequently checks that her ballot is published on B.

When the judge J runs its honest program π_{J}, it reads the content from the bulletin board B, including the public paramaters $\mathsf{prm}_{\mathsf{pub}}$, the result tally, and the proof P (if the judge does not receive one of these inputs, it outputs "reject"). Then, the judge runs Verify and outputs "accept" or "reject", respectively, if $\mathsf{Verify}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}}, \mathsf{tally}, \pi)$ evaluates to "true" or "false".

Individual verifiability. We define the goal γ_{IV} to be the set of all runs of P_{SFC} in which all honest voters' ballots are pairwise different (if $\neq \perp$), i.e., no clashes occur. For the protocol P_{SFC}, *individual verifiability* according to Smyth et al. can essentially be characterized by the fact that the protocol P_{SFC} is $(\gamma_{IV}, 0)$-verifiable by the judge J in the sense of Definition 1.

To see this, observe that if a protocol achieves individual verifiability according to Definition 9, then we have that for all ppt adversaries π_{A} the probability

$$\mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{IV},\ (\mathsf{J}\colon \mathsf{accept})] \leq \mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{IV}]$$

is negligible for $\pi = \pi_P || \pi_{\mathsf{A}}$, where the latter probability is negligible if the protocol satisfies Definition 9.

For the implication in the opposite direction, let us assume that the probability $\mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{IV},\ (\mathsf{J}\colon \mathsf{accept})]$ is negligible for all adversaries. Now, for each adversary A from the game used in Definition 9, there is a corresponding adversary π_{A} which always produces a correct tally (note that A is not concerned with tallying). For this adversary, we have

$$\mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{IV},\ (\mathsf{J}\colon \mathsf{accept})] = \mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{IV}]$$

which, by the above assumption, is negligible. This implies individual verifiability (in the sense of Definition 9).

Universal verifiability. We define the goal γ_{UV} to be the set of all runs of P_{SFC} in which first $\mathsf{prm}_{\mathsf{pub}}$ and then a final result (tally, π) are published and $\mathsf{tally} = \mathsf{correct\ tally}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}})$ (recall that B is the content of the bulletin board that contains voters' ballots).

For the protocol P_{SFC}, *universal verifiability* according to Smyth et al. can essentially be characterized by the fact that the protocol P_{SFC} is $(\gamma_{UV}, 0)$-verifiable in the sense of Definition 1.

To see this, first observe that we have $\mathsf{Verify}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}}, \mathsf{tally}', \pi')$ for each adversary A in Experiment $\mathsf{ExpUV}(\Pi, \mathsf{A})$ (Fig. 5.6) is true if an honest judge J outputs "accept" (in the system π with the corresponding adversary), and false otherwise. Second, the adversary A in Experiment $\mathsf{ExpUV}(\Pi, \mathsf{A})$ produces a tuple $(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}}, \mathsf{tally}', \pi')$ for which $\mathsf{tally}' \neq \mathsf{correct\ tally}(\mathsf{B}, \mathsf{prm}_{\mathsf{pub}})$ holds true if and only if we have $\neg\gamma_{UV}$ (in the corresponding run of π).

Thus, essentially, for a voting protocol P achieving universal verifiability according to Definition 11 (which means that the success rate in Experiment $\mathsf{ExpUV}(\Pi, \mathsf{A})$ (Fig. 5.6) is negligible for every ppt adversary A) is equivalent to the statement that the goal γ_{UV} is 0-verifiable by the judge J according to Definition 1 (which means that the probability $\mathsf{Pr}[\pi(1^\ell) \mapsto \neg\gamma_{UV},\ (\mathsf{J}\colon \mathsf{accept})]$ is negligible in every instance $\pi_P||\pi_{\mathsf{A}}$).

Election verifiability. According to Smyth et al. the protocol P_{SFC} achieves *election verifiability* if it achieves individual and universal verifiability. Therefore this notion can be expressed in the language of Definition 1 using the goal $\gamma_{IV} \wedge \gamma_{UV}$.

5.7 Symbolic Verifiability by Kremer et al.

In this section, we study the definition of verifiability proposed by Kremer et al. [KRS10] who divide verifiability into three sub-properties.

- *Individual verifiability*: a voter should be able to check that her vote belongs to the bulletin board.
- *Universal verifiability*: anyone should be able to check that the result corresponds to the content of the bulletin board.
- *Eligibility verifiability*: only eligible voter may vote.

Since the proposed formal definition for eligibility verifiability is rather long and technical, we focus here on individual and universal verifiability.

5.7.1 Model

In symbolic models, messages are represented by terms. Kremer et al. model protocols as processes in the applied-pi calculus [AF01]. A *voting specification* is a pair (V, A) where V is a process that represents the program of a voter while A is an evaluation context that represents the (honest) authorities and the infrastructure. All voters are (implicitly) assumed to be honest.

5.7.2 Individual and Universal Verifiability

We can express the definitions of Kremer et al. independently of the execution model, which slightly extends their definitions.

The symbolic verifiability definition by Kremer et al. [KRS10] assumes that each voter V_i performs an individual test ϕ_i^{IV}, and that observers perform a universal test ϕ^{UV}. The individual test ϕ_i^{IV} takes as input the voter's vote and all her local knowledge (e.g. randomness, credentials, and public election data) as well as a partial view of the bulletin board (which should correspond to her ballot). The universal test ϕ^{UV} takes as input the outcome of the election, the public election data, the bulletin board, and possibly some extra data generated during the protocol used for the purposes of verification. These tests should satisfy the following conditions for any execution.

Definition 13 (Individual and Universal Verifiability)**.** *A voting specification* (V, A) *satisfies* individual and universal verifiability *if for all* $n \in \mathbb{N}$*,*

$$\forall i, j \colon \phi_i^{IV}(\mathsf{b}) \wedge \phi_j^{IV}(\mathsf{b}) \;\Rightarrow\; i = j \tag{5.1}$$

$$\phi^{UV}(\mathsf{B}, r) \wedge \phi^{UV}(\mathsf{B}, r') \;\Rightarrow\; r \approx r' \tag{5.2}$$

$$\bigwedge_{1 \le i \le n} \phi_i^{IV}(\mathsf{b}_i) \wedge \phi^{UV}(\mathsf{B}, r) \;\Rightarrow\; \mathsf{ch} \approx r \tag{5.3}$$

where $\mathsf{ch} = (\mathsf{ch}_1, \ldots, \mathsf{ch}_n)$ *are the choices of the voters,* b *is an arbitrary ballot,* B *is the (content of the) bulletin board,* r *and* r' *are possible outcomes, and* $\approx$ *denotes equality up to permutation.*

Intuitively, Condition (5.1) ensures that two distinct voters may not agree on the same ballot, i.e., no clash occurs. Condition (5.2) guarantees the unicity of the outcome: if the observers successfully check the execution, there is at most one outcome they may accept (up to permutation). Finally, Condition (5.3) is the key property: if all tests succeed, the outcome should correspond to the voters' intent. Observe that, since all voters are assumed to be honest, the implication $\mathsf{ch} \approx r$ in Condition (5.3) can be described by the goal γ_0 (see below).

5.7.3 Discussion

Definition (13) is tailored to a specific result function: the election result has to be the sequence of the votes. Moreover, the definition assumes that the ballot of the voter can be retrieved from the bulletin board, which does not apply to ThreeBallot for example. The main restriction is that all voters are assumed to be honest.

Observe that by Condition (5.3) the goal γ_0 is guaranteed only for protocol runs in which all voters successfully verify their ballots (and the universal test is positive). For the other runs, the outcome can be arbitrary. However, the assumption that all honest voters verify their ballot is unrealistically strong. Therefore, even though this definition uses the strong goal γ_0, this assumption makes the definition weak.

5.7.4 Casting in the KTV Framework

Protocol P_{KRS}. The set of agents Σ consists of the voters, the bulletin board B, the judge J, and the remaining participants. Only static corruption is considered. The voters, the bulletin board and the judge do not accept to be corrupted. The honest programs are defined as follows:

- When a voter V_i runs her honest program π_{V_i} and is triggered in order to cast a ballot, she runs the usual program. When V_i is triggered in order to verify her vote, she performs the individual test $\phi_i^{IV}(\mathsf{b})$ with her ballot b, and if this evaluates to "true", she outputs "accept", otherwise "reject".

- When the judge J runs its honest program π_{J}, it reads the content from the bulletin board B including the result r (if it does not receive any content, it outputs "reject"). Then the judge performs the universal test $\phi^{UV}(\mathsf{B}, r)$, and if this evaluates to "false", the judge outputs "reject". Otherwise, the judge iteratively triggers each voter V_i in order to verify her ballot. If every voter outputs "accept", the judge outputs "accept", and otherwise "false". (This models the requirement in the definition of Kremer et al. that all voters have to verify successfully in order for the run to be accepted. It also means that if not all voters verify, no guarantees are given.)

End-to-end honest verifiability. Let the goal γ_{IUV} be the sub-goal of γ_0 in which all voters produce pairwise different ballots. Then, *individual and universal verifiability* by Kremer et al. (Definition (13)) can essentially be characterized by the fact that the protocol P_{KRS} is $(\gamma_{IUV}, 0)$-verifiable by the judge.

To see this, first observe that the judge J (as defined above) outputs "accept" if and only if the condition $\bigwedge_{1\leq i\leq n} \phi_i^{IV}(\mathsf{b}_i) \wedge \phi^{UV}(\mathsf{B}, r)$ (Condition (5.3)) evaluates to true. As we already pointed out, the implication $\mathsf{ch} \approx r$ in Condition (5.3) describes the goal γ_0. Condition (5.1) stating that there are no clashes between the ballots of honest voters is also satisfied in γ_{IUV} by definition. Thus, for a protocol which achieves individual and universal verifiability

according to Definition 13, the probability that the judge J in P_{KRS} accepts a protocol run in which γ_{IUV} is not fulfilled, is negligible ($\delta = 0$), i.e., we have $\mathsf{Pr}[\pi^{(\ell)} \mapsto \neg\gamma_{IUV},\ (\mathsf{J}\colon \mathsf{accept})] \leq \delta = 0$ with overwhelming probability as in Definition 1.

5.8 Symbolic Verifiability by Cortier et al.

In this section, we study the symbolic verifiability definition by Cortier et al. [CEK+15]. Cortier et al. also define different notions of verifiability: individual, universal, and end-to-end verifiability. They prove that under the assumption of an additional property, called "no clash", individual and universal verifiability imply end-to-end verifiability in their symbolic model.

5.8.1 Model

As in [KRS10], the definitions of Cortier et al. are cast in a symbolic model. That is, messages are represented by terms and protocols are defined as symbolic processes. Additionally, Cortier et al. assume that voters reach several successive states denoted as follows:

- $\mathsf{Vote}(\mathsf{id}, \mathsf{ch}, \mathsf{cred})$: the voter with identity id owns some credential cred and is willing to cast a choice ch.
- $\mathsf{MyBallot}(\mathsf{id}, \mathsf{ch}, \mathsf{b})$: the voter id has prepared a ballot b corresponding to the choice ch.
- $\mathsf{VHappy}(\mathsf{id}, \mathsf{ch}, \mathsf{cred}, \mathsf{B})$: the voter id with credential cred has cast a choice ch and is happy with the content of the bulletin board B.

Cortier et al. [CEK+15] also assume the existence of a judge who checks whether or not a result res corresponds to a bulletin board B and reaches a state $\mathsf{JHappy}(\mathsf{B}, \mathsf{res})$ whenever this is the case.

After the casting and before the tallying, some ballots may be removed because they are invalid (e.g., due to flawed signatures or zero-knowledge proofs) or simply because some voters have voted several times and only the last vote counts. This yields a "sanitized" list of ballots $\mathsf{B}_{\mathsf{san}}$.

5.8.2 Individual Verifiability

Intuitively, individual verifiability by Cortier et al. holds true if whenever honest voters perform the checks prescribed by the protocol, then their ballots belong to the bulletin board.

Definition 14 (Individual Verifiability)**.** *A protocol guarantees* individual verifiability *if for every execution, and for every voter* $\mathsf{V_{id}}$*, choice* ch*, credentials* cred *and bull board* B*, whenever the state* $\mathsf{VHappy}(\mathsf{id}, \mathsf{ch}, \mathsf{cred}, \mathsf{B})$ *is reached, it follows that*

$$\mathsf{Vote}(\mathsf{id}, \mathsf{ch}, \mathsf{cred}) \wedge \exists \mathsf{b} \in \mathsf{B} \colon \mathsf{MyBallot}(\mathsf{id}, \mathsf{ch}, \mathsf{b}).$$

5.8.3 Universal Verifiability

The universal verifiability definition by Cortier et al. depends on certain predicates whose purpose is to formally define what it means that a ballot "contains" a vote and that the tallying proceeds correctly.

Wrap. To define that a vote is "contained" in a ballot, Cortier et al. introduce a predicate $\mathsf{Wrap}(\mathsf{ch}, \mathsf{b})$ that is left undefined, but has to satisfy the following properties:

1. Any well-formed ballot b corresponding to some choice ch satisfies the Wrap predicate:

$$\mathsf{MyBallot}(\mathsf{id}, \mathsf{ch}, \mathsf{b}) \Rightarrow \mathsf{Wrap}(\mathsf{ch}, \mathsf{b})$$

2. A ballot b cannot wrap two distinct choices ch_1 and ch_2:

$$\mathsf{Wrap}(\mathsf{ch}_1, \mathsf{b}) \wedge \mathsf{Wrap}(\mathsf{ch}_2, \mathsf{b}) \Rightarrow \mathsf{ch}_1 = \mathsf{ch}_2$$

For a given protocol, the definition of Wrap typically follows from the protocol specification.

Good sanitization. When the bulletin board B is sanitized, it is acceptable to remove some ballots but of course true honest ballots should not be removed. Therefore, Cortier et al. define the predicate $\mathsf{GoodSan}(\mathsf{B}, \mathsf{B_{san}})$ to hold true (implicitly *relatively to a run*) if the honest ballots of B are not removed from $\mathsf{B_{san}}$. This means that (i) $\mathsf{B_{san}} \subseteq \mathsf{B}$, and (ii) for any $\mathsf{b} \in \mathsf{B}$ such that $\mathsf{MyBallot}(\mathsf{id}, \mathsf{ch}, \mathsf{b})$ holds true for some voter $\mathsf{V_{id}}$ and some choice ch, it is guaranteed that $\mathsf{b} \in \mathsf{B_{san}}$.

Good counting. Cortier et al. define a predicate $\mathsf{GoodCount}(\mathsf{B_{san}}, \mathsf{res})$ in order to describe that the final result res corresponds to counting the votes of $\mathsf{B_{san}}$. This is technically defined in [CEK+15] by introducing an auxiliary bulletin board $\mathsf{B'_{san}}$ which is a permutation of $\mathsf{B_{san}}$ and from which the list rlist of votes (such that $\mathsf{res} = \rho(\mathsf{rlist})$ where ρ is the counting function) can be extracted line by line from $\mathsf{B'_{san}}$. More formally, $\mathsf{GoodCount}(\mathsf{B_{san}}, \mathsf{res})$ holds true if there exist $\mathsf{B'_{san}}, \mathsf{rlist}$ such that (i) $\mathsf{B_{san}}$ and rlist have the same size, and

(ii) $\mathsf{B}_{\mathsf{san}}$ and $\mathsf{B}'_{\mathsf{san}}$ are equal as multisets, and (iii) $\mathsf{res} = \rho(\mathsf{rlist})$, and (iv) for all ballots b with $\mathsf{B}'_{\mathsf{san}}[j] = \mathsf{b}$ for some index j, there exists a choice ch such that $\mathsf{Wrap}(\mathsf{ch}, \mathsf{b})$ as well as $\mathsf{rlist}[j] = \mathsf{ch}$ hold true. Note that the definition of $\mathsf{GoodCount}$ is parameterized by the counting function ρ of the protocol under consideration.

Then, universal verifiability is defined as follows.

Definition 15 (Universal Verifiability)**.** *A protocol guarantees* universal verifiability *if for every execution, and every bulletin board* B *and result* res*, whenever the state* $\mathsf{JHappy}(\mathsf{B}, \mathsf{res})$ *is reached, it holds that*

$$\exists \mathsf{B}_{\mathsf{san}} : \mathsf{GoodSan}(\mathsf{B}, \mathsf{B}_{\mathsf{san}}) \wedge \mathsf{GoodCount}(\mathsf{B}_{\mathsf{san}}, \mathsf{res}).$$

Intuitively, whenever the judge (some election authority) states that some result res corresponds to a bulletin board B, then res corresponds to the votes contained in a *subset* $\mathsf{B}_{\mathsf{san}}$ of B (some ballots may have been discarded because they were ill-formed for example) and this subset $\mathsf{B}_{\mathsf{san}}$ contains at least all ballots formed by honest voters that played the entire protocol (that is, including the final checks).

5.8.4 E2E Verifiability

Intuitively, end-to-end verifiability according to Cortier et al. holds true if, whenever no one complains (including the judge), then the election result includes all the votes corresponding to honest voters that performed the checks prescribed by the protocol.

Definition 16 (E2E Verifiability)**.** *A protocol guarantees* end-to-end verifiability *if for every execution, and every bulletin board* B *and result* res*, whenever a state is reached such that for some subset of the honest voters (indexed by some set* I*) with choices* $\mathsf{ch}_{\mathsf{id}}$ *and credentials* $\mathsf{cred}_{\mathsf{id}}$ *(*$\mathsf{id} \in I$*), we have*

$$\mathsf{JHappy}(\mathsf{B}, \mathsf{res}) \wedge \bigwedge_{\mathsf{id} \in I} \mathsf{VHappy}(\mathsf{id}, \mathsf{ch}_{\mathsf{id}}, \mathsf{cred}_{\mathsf{id}}, \mathsf{B}),$$

then there exist rlist *such that we have* $\mathsf{res} = \rho(\mathsf{rlist})$ *and* $\{\mathsf{ch}_{\mathsf{id}}\}_{\mathsf{id} \in I} \subseteq \mathsf{rlist}$ *(as multisets).*

5.8.5 No Clash

Finally, Cortier et al. define the notion of "no clash" as follows. Intuitively, "no clash" describes the property that two distinct honest voters may not build the same ballot.

Definition 17 (No Clash)**.** *A protocol guarantees* no clash *if for every execution, whenever a state is reached such that*

$$\mathsf{MyBallot}(i, \mathsf{ch}_i, \mathsf{b}) \wedge \mathsf{MyBallot}(j, \mathsf{ch}_j, \mathsf{b}),$$

then it must be the case that $i = j$ *and* $\mathsf{ch}_i = \mathsf{ch}_j$*.*

5.8.6 Discussion

Cortier et al. [CEK+15] showed that individual verifiability, universal verifiability, and the "no clash" property together imply end-to-end verifiability (all as defined above).

In order to be able to define their notions of individual and universal verifiability, Cortier et al. proposed a model in which it is possible to (*i*) extract single ballots from the bulletin board (implicit in the predicate VHappy), and to (*ii*) uniquely determine the content, i.e. the plain vote, of each single ballot (Wrap predicate). Therefore, these definitions can only be applied to a class of protocols which fulfill these requirements, and by this, for example, ThreeBallot [Smi07] as well as protocols in which ballots are information theoretically secure commitments (e.g. [CPP13]) can not be analyzed.

The notion of end-to-end verifiability (Definition 16) is rather weak since it only requires that honest votes are counted (for voters that checked). It does not control dishonest votes. In particular, this notion does not prevent ballot stuffing. The authors of [CEK+15] introduced this notion because the Helios protocol does not satisfy strong verifiability, as defined in [CGGI14] for example (see also Section 5.5). Moreover, the verification technique based on typing developed in [CEK+15] would probably require some adaption to also cover strong verifiability as it would need to *count* the number of votes, which is a difficult task for type-checkers.

5.8.7 Casting in the KTV Framework

Protocol P_{CEKMW}**.** The set of agents Σ consists of the honest voters, the bulletin board B, the judge J, and the remaining participants. Only static corruption is considered. The bulletin board and the judge do not accept to be corrupted. The honest programs are defined as follows:

- When a voter V runs her honest program π_{V}, and is triggered to cast her ballot, she expects an identity id and a choice ch (if not, she stops). Then, she runs Vote(ch) to build her ballot b and to submit it to the bulletin board. Afterwards, she reaches a state MyBallot(id, ch, b). When the voter is triggered to verify her vote, she reads the content of the

bulletin board B and reaches a state $\mathsf{VHappy}(\mathsf{id}, \mathsf{ch}, \mathsf{B})$ if her checks evaluate to true.

- When the judge J runs its honest program π_{J} and is triggered to verify the election run, it reads the content of the bulletin board B including the final result res (if not possible, J outputs "reject"). If the judge successfully performs some checks (which depend on the concrete voting protocol), then he outputs "accept" and reaches a state $\mathsf{JHappy}(\mathsf{B}, \mathsf{res})$.

Individual verifiability. We define the goal γ_{IV} to be the set of all runs of P_{CEKMW} in which whenever an honest voter V_{id} reaches the state $\mathsf{VHappy}(\mathsf{id}, \mathsf{ch}, \mathsf{B})$ for some choice ch and ballot b, then there exists a ballot $\mathsf{b} \in \mathsf{B}$ such that this voter started with $(\mathsf{id}, \mathsf{ch})$ as her input and reached $\mathsf{MyBallot}(\mathsf{id}, \mathsf{ch}, \mathsf{b})$ as intermediary state. Then, *individual verifiability* by Cortier et al. (Definition 14) can essentially be characterized by the fact that the protocol P_{CEKMW} is $(\gamma_{IV}, 0)$-verifiable by the judge J.

Universal verifiability. We define the goal γ_{UV} to be the set of all runs of P_{CEKMW} in which whenever a result res is obtained and the final content of the bulletin board is B then there exists $\mathsf{B}_{\mathsf{san}}$ such that $\mathsf{GoodSan}(\mathsf{B}, \mathsf{B}_{\mathsf{san}})$ and $\mathsf{GoodCount}(\mathsf{B}_{\mathsf{san}}, \mathsf{res})$ hold true (as defined above). Then, *universal verifiability* by Cortier et al. (Definition 15) can essentially be characterized by the fact that the protocol P_{CEKMW} is $(\gamma_{UV}, 0)$-verifiable by the judge J.

End-to-end verifiability. We define the goal γ_{E2E} to be the set of all runs of P_{CEKMW} in which the result res of the election satisfies $\mathsf{res} = \rho(\mathsf{rlist})$ for some rlist that contains (as multiset) all the choices $\mathsf{ch}_{\mathsf{id}}$ for which some honest voter V_{id} reached a state $\mathsf{VHappy}(\mathsf{id}, \mathsf{ch}_{\mathsf{id}}, \mathsf{cred}_{\mathsf{id}}\mathsf{B})$. Then, *end-to-end verifiability* by Cortier et al. (Definition 16) can essentially be characterized by the fact that the protocol P_{CEKMW} is $(\gamma_{E2E}, 0)$-verifiable by the judge J.

5.9 Publicly Auditable Secure MPC by Baum et al.

This section focusses on the definition of *publicly auditable secure multi-party computation* by Baum et al. [BDO14]. Baum et al. also present a game-based definition of auditable correctness which is, however, underspecified.[3] Therefore, we only analyze the definition of auditable correctness as implied by the ideal functionality.

[3] More precisely, in the game-based definition it is not stated by whom the input $x_1, \ldots, x_m$ is provided.

5.9.1 Model

The protocols are *client-server MPC protocols* in the Universal Composability Framework, where a set of parties provide input to the actual working parties, who run the MPC protocol among themselves and make the output public. The *input parties* are denoted by $\mathsf{V}_1, \ldots, \mathsf{V}_n$ and their inputs are denoted by $(\mathsf{ch}_1, \ldots, \mathsf{ch}_n)$. The *computing parties* are denoted by $\mathsf{T}_1, \ldots, \mathsf{T}_m$ and participate in the computation phase. Given a set of inputs $\mathsf{ch}_1, \ldots, \mathsf{ch}_n$, they compute an output $C(\mathsf{ch}_1, \ldots, \mathsf{ch}_n)$ for some circuit C over a finite field. After the protocol is executed, anyone acting as the judge J can retrieve the transcript τ of the protocol from the bulletin board and (using only the circuit C and the output res) determine whether the result is valid or not.

5.9.2 Auditable Correctness

Definition 18. *A client-server MPC protocol achieves* auditable correctness *if it realizes the ideal functionality $\mathcal{F}_{AuditMPC}$ (Fig. 5.7) in the UC framework.*[4]

In what follows, we describe the basic concept of the ideal functionality $\mathcal{F}_{AuditMPC}$ (Fig. 5.7). In the *initializing phase*, the adversary can determine which input parties and which computing parties are corrupted. In the *input phase*, each honest input party V provides the ideal functionality with input ch, and for each corrupted input party V, the adversary can provide the ideal functionality with an arbitrary input ch' which is then considered as the input of V. The inputs for honest and dishonest input parties are stored as $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_n$. In a voting protocol, $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_n$ denote the choices of the honest voters plus possible choices of the dishonest voters being provided by the adversary (at most one vote for each dishonest voter). In the *compute phase*, the ideal functionality first computes the correct result $\mathsf{res}' = C(\mathsf{ch}'_1, \ldots, \mathsf{ch}'_n)$. If no computing party is corrupted, we take $\mathsf{res} = \mathsf{res}'$. Otherwise (if at least one computing party is corrupted), the adversary is given res' and can determine the output res with the following restriction: if not all computing parties are corrupted, the adversary can output $\perp$ or res', and otherwise, he can choose an arbitrary result. In a voting protocol, res' denotes the *correct* output of the voting protocol run with input $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_n$, while res denotes

[4] In the ideal functionality $\mathcal{F}_{AuditMPC}$ as defined in the original work [BDO14], the variables $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_m$ that are used in the second step of the computing phase are not defined. It is reasonable to assume that these variables denote the inputs ch or ch', respectively, which are associated to the input parties $\mathsf{V}_1, \ldots, \mathsf{V}_n$ and stored in the first step of the input phase. Therefore, we added the third line of the input phase above to the ideal functionality from the original paper.

Functionality $\mathcal{F}_{AuditMPC}$

- **Initialize:** On input (Init, C) from all parties (where C is a circuit with n inputs and one output): Wait until A sends the sets $\mathsf{A}_{BV} \subseteq \{1, \ldots, n\}$ (corrupted input parties) and $\mathsf{A}_{BT} \subseteq \{1, \ldots, m\}$ (corrupted computing parties).
- **Input**: On input $(\mathsf{Input}, \mathsf{V}_i, varid_{\mathsf{ch}}, \mathsf{ch})$ from V_i and on input $(\mathsf{Input}, \mathsf{V}_i, varid_{\mathsf{ch}}, ?)$ from all parties T_k, with $varid_{\mathsf{ch}}$ a fresh identifier:
  1. Store $(varid_{\mathsf{ch}}, \mathsf{ch})$.
  2. If $|\mathsf{A}_{BT}| = m$, send $(\mathsf{Input}, \mathsf{V}_i, varid, \mathsf{ch})$ to all T_k.
  3. For all $i \in \{1, \ldots, n\}$ let ch'_i denote the input stored for V_i.
- **Compute**: On input $(\mathsf{Compute})$ from all parties T_k:
  1. If an input gate of C has no value assigned, stop here.
  2. Compute $\mathsf{res}' = C(\mathsf{ch}'_1, \ldots, \mathsf{ch}'_n)$.
  3. If $|\mathsf{A}_{BT}| = 0$, set $\mathsf{res} = \mathsf{res}'$. If $|\mathsf{A}_{BT}| > 0$, output res' to A and wait for res from A. If $|\mathsf{A}_{BT}| < n$, the functionality accepts only $\mathsf{res} \in \{\perp, \mathsf{res}'\}$. If $|\mathsf{A}_{BT}| = n$, any value res is accepted.
  4. Output $(\mathsf{Output}, \mathsf{res})$ to all parties.
- **Audit**: On input $(\mathsf{Audit}, \mathsf{y})$ from J, and if $\mathsf{Compute}$ was executed, the functionality does the following:
  - If $\mathsf{res}' = \mathsf{res} = \mathsf{y}$, then output "accept y".
  - If $\mathsf{res} = \perp$, then output "no audit possible".
  - If $\mathsf{res}' \neq \mathsf{res}$ or $\mathsf{y} \neq \mathsf{res}$, then output "reject y".

Figure 5.7: Ideal functionality $\mathcal{F}_{AuditMPC}$ by Baum et al. describing the online phase.

the *actual* output of the run. In the *audit phase*, the ideal functionality checks whether the output res coincides with the correct result res'. For a voting protocol, the property that res' and res coincide is simply the goal γ_0 as introduced in Definition 4.

5.9.3 Discussion

The auditor of an MPC protocol realizing the ideal functionality $\mathcal{F}_{AuditMPC}$ in the UC framework always has to accept the result res of a run (with overwhelming probability) if the result is correct, regardless of the rest of the run (see the first case of the audit part in the ideal functionality). However, this fairness requirement is unrealistically strong since then, as long as the result of a protocol run is correct, the auditor has to accept the result even if, for example, ZK proofs are flawed.

If an attacker does not control all computing parties, i.e., if that there is at least one honest computing party, then an MPC protocol that realizes the ideal functionality guarantees that either the result is correct, or else not output is produced. This assumption is, however, unrealistically strong.

5.9.4 Casting in the KTV Framework

Protocol P_{BDO}. The set of agents Σ consists of the voters, the bulletin board B, the judge J, and the remaining participants. Since static corruption is considered, the agents only accept to be corrupted at the beginning of an election run. The bulletin board B and the judge J do not accept to be corrupted.

When a voter V_i runs her honest program π_{V_i}, she receives a choice ch_i and then runs $\mathsf{Vote}(\mathsf{ch}_i)$ (see Section 2.1).

The honest program of the judge J depends on the concrete voting protocol.

Public auditability. Let the goal γ_0 be defined as in Definition 4 by Küsters et al.[5]. *Public auditability* of a protocol P_{BDO} (as implied by the ideal functionality $\mathcal{F}_{AuditMPC}$) can essentially be characterized by the fact that (i) the protocol P_{BDO} is $(\gamma_0, 0)$-verifiable by the judge J (Definition 1), and (ii) the output of the protocol is either the correct one or $\perp$ if at least one computing party is honest. To see this, note that in its audit phase, the ideal functionality $\mathcal{F}_{AuditMPC}$ accepts the run *if and only if* the published result res is equal to the correct result res', i.e., γ_0 is achieved.

Additionally, as already mentioned, public auditability entails the fairness condition that requires a run to be accepted whenever the produced result is correct (as mentioned, this condition is too strong in the context of e-voting).

[5]Since Definition 18 does not distinguish between valid and invalid choices ch (see Discussion 5.9.3), we assume that the set of valid choices is the finite field over which the circuit C is defined.

5.10 Universal Verifiability by Chevallier-Mames et al.

In this section, we analyze the definition of universal verifiability proposed by Chevallier-Mames et al. [CFP+10].

5.10.1 Model

For each voter V_i, let B_i denote the transcript of V_i, i.e., the interactions between V_i and the voting authority. The bulletin board B is regarded as the set of transcripts. Any interaction, including those with the authorities, can be assumed public. Chevallier-Mames et al. assume that each voting protocol guarantees the following requirements.

1. Detection of individual fraud: From a partial list of transcripts B produced by $\mathsf{V}_1, \ldots, \mathsf{V}_n$, the voting authority should be able to determine whether a new transcript B_{n+1} produced by V_{n+1} is valid (well-formed and does not correspond to a double vote). More formally, there exists a boolean function f such that

$$\begin{aligned}
&\forall n, \forall \mathsf{V}_1, \ldots, \mathsf{V}_n, \mathsf{V}_{n+1} \\
&\forall \mathsf{B} \leftarrow \mathsf{V}_1, \ldots, \mathsf{V}_n, \mathsf{B}_{n+1} \leftarrow \mathsf{V}_{n+1}, \\
&\mathsf{B}_{n+1} \text{ valid } \wedge f(\mathsf{B}, \mathsf{B}_{n+1}) = \begin{cases} 0, & \text{if } \mathsf{V}_{n+1} \in \{\mathsf{V}_1, \ldots, \mathsf{V}_n\} \\ 1, & \text{if } \mathsf{V}_{n+1} \notin \{\mathsf{V}_1, \ldots, \mathsf{V}_n\} \end{cases}
\end{aligned}$$

The language of the bulletin boards B which are iteratively valid is denoted by $\mathcal{L}$.

2. Computation of the result: From the transcripts, the voting authority should be able to compute the result, that is a vector of the number of selections for each candidate: there exists an efficient function f' that, from the bulletin board B, outputs res,

$$\forall \mathsf{B} \in \mathcal{L}, f'(\mathsf{B}) = \sum_i \mathsf{ch}_i = \mathsf{res}.$$

3. Computation of the list of the voters: From the transcripts, the voting authority should be able to determine the list $\mathsf{V}_{\mathsf{cast}}$ of the voters who actually casted their ballots: there exists an efficient function f'' that, from the bulletin board B, extracts the sublist $\mathsf{V}_{\mathsf{cast}}$ of the voters,

$$\forall \mathsf{B} \in \mathcal{L}, f''(\mathsf{B}) = \mathsf{V}_{\mathsf{cast}}.$$

5.10.2 Universal Verifiability

The idea of universal verifiability by Chevallier-Mames et al. is that everybody should be able to check the correctness/validity of the votes and of the computation of the result and the voters: the bulletin-board B, the result res and the list of the voters $\mathsf{V}_{\mathsf{cast}}$ should rely in an NP language $\mathcal{L}'$, defined by the relation R: there exists a witness w which allows an efficient verification. Furthermore, for any B, the valid res and $\mathsf{V}_{\mathsf{cast}}$ should be unique.

Definition 19 (Universal Verifiability)**.** *Let R be the NP-relation for the language $\mathcal{L}'$ of the valid ballots and valid computation of the result. A voting scheme achieves the* universal verification *property if only one value for the result and the list of the voters can be accepted by the relation R, and the witness w can be easily computed from the bulletin-board B using a function g:*

$$\begin{aligned} \forall \mathsf{B} \in \mathcal{L}, \exists!(\mathsf{res}, \mathsf{V}_{\mathsf{cast}}), \exists w\colon & \; R(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}, w) = 1 \\ \forall \mathsf{B} \notin \mathcal{L}, \forall(\mathsf{res}, \mathsf{V}_{\mathsf{cast}}, w)\colon & \; R(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}, w) = 0 \\ \forall \mathsf{B} \in \mathcal{L}\colon & \; R(\mathsf{B}, f'(\mathsf{B}), f''(\mathsf{B}), g(\mathsf{B})) = 1. \end{aligned}$$

Note that g is a function private to the authorities, to compute a short string (the witness) that allows everybody to check the overall validity, granted the public relation R. The functions f, f', f'' and g may be keyed according to the system parameters: g is clearly private to the voting authority, while f and f'' may be public (which is the case in schemes based on homomorphic encryption). The function f' is likely to be private.

5.10.3 Discussion

The second requirement for voting schemes according to Chevallier-Mames et al. (computation of the result) excludes dishonest voters since for them ch_i remains undefined as they might not even produce ch_i; and even if a dishonest voter produces some ch_i as an honest voter does, she might not submit it.

Because of the same requirement, the model abstracts away from the problem that ballots might be dropped or manipulated in the casting phase: it implicitly assumes that each valid ballot of a voter who has not voted yet gets to the bulletin board. In addition, the model does not make a difference between a voter and her client.

Whether or not a voting scheme achieves universal verifiability according to Chevallier-Mames et al. (Definition 19), depends on how "B_i valid" is defined for the transcripts B_i used in the voting scheme.

The second condition in Definition 19 is too strong from a practical point of view because a voting scheme in which invalid ballots are removed can not achieve universal verifiability according to Definition 19. To see this, consider the case that in a run of a voting protocol a voter provides an invalid zero-knowledge proof for her ballot. Then, in order to guarantee correctness, this ballot is (typically) not considered in the result. Since the transcript of the voter who submitted the invalid ballot is not valid, we have $\mathsf{B} \notin \mathcal{L}$. By the second point of Definition 19, it follows that, even if all voting authorities are honest, then each zero-knowledge proof for the correct result of the election w.r.t. the voters who submitted valid ballots will be rejected.

If a voting scheme is universally verifiable according to Definition 19, and in a run of its protocol we have that $\mathsf{B} \in \mathcal{L}$, then by the first and third condition in Definition 19 it follows that in this run

$$(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}) \in \mathcal{L}' \Leftrightarrow (\mathsf{res}, \mathsf{V}_{\mathsf{cast}}) = (f'(\mathsf{B}), f''(\mathsf{B})).$$

By the second requirement for voting schemes (computation of the result), the fact that $\mathsf{B} \in \mathcal{L}$ and $(\mathsf{res}, \mathsf{V}_{\mathsf{cast}}) = (f'(\mathsf{B}), f''(\mathsf{B}))$ hold true in a run implies that the same run satisfies γ_0 as introduced in Definition 4. Conversely, the fact that a protocol run satisfies γ_0 does not imply that $\mathsf{B} \in \mathcal{L}$ holds true in the same run, and in particular, γ_0 does not imply $(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}) \in \mathcal{L}'$. To see this, consider the case above where invalid ballots are removed.

The conditions in Definition 19 have to be fulfilled in every run of the protocol, and not only with overwhelming probability. This requirement is typically too strong.

5.10.4 Casting in the KTV Framework

Protocol P_{CFPST}. The set of agents Σ consists of the voters, the judge J and the remaining participants. Let B and $\mathcal{L}$ be defined as in the model of Chevallier-Mames et al. (Section 5.10.1). Since static corruption is considered, the agents only accept to be corrupted at the beginning of an election run. The voters, the bulletin board, and the judge do not accept to be corrupted.

When V_i runs her honest program π_{V_i}, she expects a candidate ch_i as input. If the input is empty, or if the input is not empty but the voter has already been triggered and received a candidate before, she stops. Otherwise, she runs Vote(ch_i).

The honest program π_{J} of the judge J depends on the concrete election scheme. Intuitively, when the judge runs her honest program, she receives

$(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}})$ along with a zero-knowledge proof as her input, and then evaluates whether $(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}) \in \mathcal{L}'$ holds true. She outputs "accept" if and only if the evaluation is positive.

Universal verifiability. Let the goal γ_0 be defined as in Definition 4 by Küsters et al., and the language $\mathcal{L}$ be defined as in Section 5.10.1. For the protocol P_{CFPST}, *universal verifiability* according to Chevallier-Mames et al. (Definition 19) can essentially be characterized by the fact that the protocol P_{CFPST} is $(\gamma_0 \cap \mathcal{L}, 0)$-verifiable in the sense of Definition 1.

To see this, first note that if γ_0 is not satisfied in a run, then by the first and third condition in Definition 19 we have $(\mathsf{B}, \mathsf{res}, \mathsf{V}_{\mathsf{cast}}) \notin \mathcal{L}'$. Consequently, an honest judge as sketched above, does not accept the run. However, as shown above, there are runs in which γ_0 is satisfied but an honest judge as sketched above does not accept the run because $\mathsf{B} \notin \mathcal{L}$ holds true. Therefore, the goal γ requires that in a run γ_0 *and* $\mathsf{B} \in \mathcal{L}$ must hold true in order to describe Definition 19.

Definition 19 does not require any variant of fairness. The reason is that if all voting authorities are honest but one single voter is dishonest and submits an invalid ballot, then the judge (as sketched above) does not accept the result although γ_0 is achieved (see Discussion 5.10.3).

5.11 Universal Verifiability by Szepieniec et al.

In this section, we present and discuss the definition of universal verifiability by Szepieniec et al. [SP15]. Due to its shortcomings (see discussion below), we omit the casting in the KTV framework.

5.11.1 Model

Definition 20 is supposed to be applicable to any protocol P that can be analyzed in the universal composability (UC) framework.

5.11.2 Universal Verifiability

In the universal verifiability definition by Szepieniec et al. , the judge J takes as input the transcript as produced by an adversary A attacking a protocol P. The judge eventually outputs a bit $\tilde{b}$. Let b be a variable indicating whether the protocol was executed correctly by all parties – i.e., the parties behaved honestly and were not corrupted by the adversary – and if the transcript is authentic, by assuming the value 1 if this is the case and 0 otherwise.

Definition 20 (Universal Verifiability)**.** *A protocol P is* universally verifiable *if there exists a judge* J *such that, for all adversaries* A *attacking the protocol,* Ver *has significant distinguishing power:*

$$\left|\mathsf{Pr}[b = \tilde{b}] - \mathsf{Pr}[b \neq \tilde{b}]\right| \geq \frac{1}{2},$$

where the probabilities are taken over all random coins used by J*,* A *and* P*.*

Szepieniec et al. stress that the judge J in Definition 20 must be able to differentiate between simulated parties created by the adversary A and genuine protocol participants.

5.11.3 Discussion

The universal verifiability definition by Szepieniec et al. has two fundamental shortcomings.

Firstly, elementary expressions used in Definition 20 remain undefined in [SP15]. For example, it is unclear what the terms "attacking the protocol", "behaved honestly" and "corrupted" formally mean.

Secondly, if one assumes common definitions of honest/corrupted, the universal verifiability definition is clearly too strong because, as Szepieniec et al. point out, the judge must be able to differentiate between honest and dishonest participants. This is, however, typically impossible for any (not necessarily ppt) judge in any protocol because a corrupted participant can still follow its honest program. On a high level, Definition 20 requires that there has to exist a judge who can "look inside" the (corrupted) participants, rather than a judge who opts for possible deviations in the publicly available data.

5.12 Summary and Conclusion

In the previous sections, we have studied the formal definitions of verifiability for e-voting system proposed in the literature. We have presented the original definitions and cast them in the KTV framework. This casting has demonstrated that the essence of these notions can be captured within a uniform framework and enabled us to identify their relative and recurrent merits and weaknesses as well as their specific (partly severe) limitations and problems.

In Section 5.12.1, we distill these discussions and insights into detailed requirements and guidelines that highlight several aspects any verifiability definition should cover. We also summarize from the previous sections how

the different existing definitions of verifiability from the literature handle these aspects. Finally, in Section 5.12.2, as a viable and concrete embodiment of our guidelines, we instantiate the KTV framework accordingly, obtaining a solid and ready to use definition of verifiability.

5.12.1 Guidelines

We now present our requirements and guidelines for the following central aspects, along with a summary of the previous sections concerning these aspects.

Generality. Many verifiability definitions are designed for protocols with specific protocol structures and are tailored to them (see Sections 5.4, 5.5, 5.6, 5.7, 5.8). As a result, for new classes of protocols often new definitions are necessary.

Clearly, it is desirable for a verifiability definition to be applicable to as many protocols as possible. It provides not only reusability, but also comparability: by applying the same definition to different protocols and protocol classes we can clearly see the differences in the level and nature of verifiability they provide. A very minimal set of assumptions on the protocol structure is sufficient to express a meaningful notion of verifiability, as illustrated by the definition in Section 5.2 and also by the instantiation of the KTV framework given below.

Note, however, that some additional assumptions on the protocol structure allow one to express some specific properties, such as universal verifiability, which, as discussed in the previous sections, on their own do not capture end-to-end verifiability, but may be seen as valuable additions.

Static versus dynamic corruption. We observe that most of the studied verifiability definitions focus on static corruption, except the definitions in Sections 5.4 and 5.5, which capture the dynamic corruption of voters. In general, modeling dynamic corruption can yield stronger security guarantees. In the context of verifiability, one could, for example, provide guarantees not only to honest voters but also to certain corrupted voters. If a voter is corrupted only late in the election, e.g., when the voting phase, one might still want to guarantee that her vote is counted. None of the existing definitions provide this kind of guarantee so far. We briefly discuss how this can be captured in the KTV framework in Section 5.12.2.

Binary versus quantitative verifiability. As discussed in Section 2.3, the probability δ (see Definition 1) that under realistic assumptions some cheating by an adversary remains undetected may be bigger than 0 even for reasonable protocols: often some kind of partial and/or probabilistic checking

is carried out, with Benaloh audits (see Section 5.5.7) being an example. These checks might fail to detect manipulations with some non-negligible probability. Still, as we have seen when casting the different verifiability notions in the KTV framework, most of the studied definitions assume the verifiability tolerance to be $\delta = 0$. This yields a binary notion of verifiability which, as explained, outright rejects reasonable protocols.

In contrast, the definitions studied in the KTV framework (including Section 5.2) as well as the ones in Sections 5.3 and 5.4 , allow for measuring the level of verifiability. This gives more expressiveness and allows one to establish meaningful verifiability results for (reasonable) protocols which do not provide perfect verifiability.

Goals. As pointed out in Section 5.2, the goal γ_0, which, among others, requires that all the ballots cast by honest voters are correctly tallied and make it to the final result is very strong and typically too strong. In order to satisfy this goal very strong trust assumptions are necessary, for instance, the assumptions taken in the definition of weak verifiability in Section 5.5.

From the previous sections, two main and reasonable approaches for defining a goal emerged, which one could characterize as quantitative and qualitative, respectively:

- *Quantitative.* In Section 5.2, a family of goals γ_k, $k \geq 0$, together with a non-zero tolerance level δ is considered; a similar approach is taken in Section 5.4, but see the discussion in this section. This approach, among others, captures that the probability that more than k votes of honest voters can be changed without anybody noticing should be small, i.e., bounded by δ. To be more precise and allow for stronger guarantees, this approach could be combined with an aspect of the goal defined for strong verifiability, namely the distinction between votes that are manipulated and those that are "just" dropped (see Section 5.5).

- *Qualitative.* In Section 5.5 ("strong verifiability"), the protocol goal (as cast in the KTV framework), among others, stipulates that votes of voters who verify their receipt are contained in the final result. To be general, this approach should also be combined with a non-zero tolerance level δ (which, however, was not captured in the original definition). The reason is that checks (such as Benaloh challenges) might not be perfect, i.e., manipulation might go undetected with some probability.

In both cases, votes of dishonest voters were restricted to be counted at most once (no ballot stuffing).

The quantitative approach, on the one hand, provides overall guarantees about the deviation of the published result from the correct one and measures the probability δ that the deviation is too big (bigger than k) but nobody notices this. On the other hand, it does not explicitly require that voters who check their receipts can be sure (up to some probability) that their votes were counted. But, of course, to prove verifiability of a system w.r.t. this goal, one has to take into account whether or not voters checked, and more precisely, the probabilities thereof. These probabilities also capture the uncertainty of the adversary about whether or not specific voters check, and by this, provides protection even for voters who do not check.

The qualitative approach explicitly provides guarantees for those honest voters who verify their receipts. On the one hand, this has the advantage that one does not need to consider probabilities of voters checking or not, which simplifies the analysis of systems. On the other hand, such probabilities of course play an important role for measuring the overall security of a system, an aspect this simpler approach abstracts away. Nevertheless, it provides a good qualitative assessment of a system.

Interestingly, one could in principle combine both approaches, i.e., consider the intersection of both goals. While this would give voters also in the quantitative approach direct guarantees (in addition to the aspect of making a distinction between manipulating and dropping votes, mentioned above already), it would typically not really change the analysis and its result: as mentioned, in the quantitative analysis one would anyway have to analyze and take into account the guarantees offered when checking receipts.

Below, we provide concrete instantiations for both approaches in the KTV framework.

Ballot stuffing. Not all definitions of verifiabiltiy rule out ballot stuffing, even though ballot stuffing, if unnoticed, can dramatically change the election result. Some definitions go even further and abstract away from this problem by assuming that there are only honest voters (see trust assumptions below).

Clearly, allowing undetected ballot stuffing makes a verifiability definition too weak. We recommend that a verifiability definition should exclude undetected ballot stuffing. It might also be useful to capture different levels of ballot stuffing in order to distinguish systems where it is very risky to add even a small number of ballots from those where adding such a small number is relatively safe. The goals discussed above, as mentioned, both require that no ballot stuffing is possible at all.

Trust assumptions. Some verifiability definitions assume some protocol participants to be always honest, for example the bulletin board (Sections 5.3, 5.4, 5.9, 5.6, 5.7, 5.8), 5.10 or all voters (Section 5.7) or all voter supporting

devices (Sections 5.6, 5.5), or some disjunctions of participants (Section 5.5); the definition discussed in Section 5.2 does not make such assumptions. We think that verifiability definitions which rely on the unrealistic assumption that all voters are honest are too weak. The other trust assumptions might be reasonable depending on the threat scenario. A general verifiability definition should be capable of expressing different trust assumptions and make them explicit; embedding trust assumptions into a definition not only makes the definition less general, but also makes the assumptions more implicit, and hence, easy to overlook.

Individual and universal verifiability. In Section 5.6 and 5.8, definitions of individual and universal verifiability were presented. We already pointed out that the split-up of end-to-end verifiability into sub-properties is problematic. In fact, Küsters et al. [KTV12b] have proven that, in general, individual and universal verifiability (even assuming that only eligible voters vote) do not imply end-to-end verifiability, e.g. for ThreeBallot [Smi07] (see Section 2.3.3). For the definitions of individual and universal verifiability presented in Section 5.5, it was shown in [CEK+15] that they imply end-to-end verifiability under the assumption that there are no clashes [KTV12b]. However, the notion of end-to-end verifiability considered there is too weak since it allows ballot stuffing. For the definitions of individual and universal verifiability in Section 5.6 no such proof was provided, and therefore, it remains unclear whether it implies end-to-end verifiability. (In fact, technically these definitions, without some fixes applied, do not provide end-to-end verifiability as pointed out in Section 5.6.)

The (combination of) notions of individual and universal verifiability (and other properties and subproperties, such as eligibility verifiability, cast-as-intended, recorded-as-cast, and counted-as-recorded) should not be used as a replacement for end-to-end verifiability per se since they capture only specific aspects rather than the full picture. Unless formally proven that their combination in fact implies end-to-end verifiability they might miss important aspects, as discussed above. Therefore, the security analysis of e-voting systems should be based on the notion of end-to-end verifiability (as, for example, concretely defined below). Subproperties could then possibly be used as useful proof techniques.

5.12.2 Exemplified Instantiation of the Guidelines

We now demonstrate how the guidelines given above can be put into practice, using, as an example, the KTV framework. By this, we obtain a solid, ready-to-use definition of verifiability. More specifically, we propose two variants,

one for qualitative and one for quantitative reasoning, as explained next.

The distillation of our observations given in Section 5.12.1 reviews different aspects of verifiability and, in most cases, it clearly identifies the best and favorable ways they should be handled by verifiability definitions. When it comes to the distinction between qualitative and quantitative approaches to define verifiability goals, we have, however, found out that both approaches have merits and both can yield viable definitions of verifiability. This is why we propose two instantiations of the KTV framework, one following the qualitative approach and one for the quantitative approach.

To instantiate the KTV framework, one only has to provide a definition of a goal (a family of goals) that a protocol is supposed to guarantee. Note that, as for the second parameter of Definition 1, δ, one should always try, for a given goal, to establish an as small δ as possible. In other words, the value of δ is the result of the analysis of a concrete system, rather than something fixed up front.

In the following, we define two goals corresponding to the two variants of verifiability discussed above: goal $\gamma_{ql}(\varphi)$ for the qualitative variant and goal $\gamma_{qn}(k, \varphi)$ for the quantitative one. We explain the meaning of the parameters below. Here we only remark that the common parameter φ describes the trust assumptions (i.e., it determines which parties are assumed to be honest and which can be corrupted and when) under which the protocol is supposed to provide specific guarantees. Recall that, in the KTV framework, the adversary sends a special message $\mathsf{corrupt}$ to a participant in order to corrupt it (a participant can then accept or reject such a message). This allows for modeling various forms of static and dynamic corruption. Note also that it is easily visible, given a run, if and when a party is corrupted.

In the following, for a given run r of an e-voting protocol with n_{voters} eligible voters, we denote by $n_{\mathsf{voters}}^{\mathsf{honest}}$ the number of honest and by $n_{\mathsf{voters}}^{\mathsf{dishonest}}$ the number of dishonest voters in r. Recall that we say a party is honest in a run r if it has not received a $\mathsf{corrupt}$ message or at least has not accepted such a message throughout the whole run. We denote by $\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}}$ the actual choices of the honest voters in this run (which might include abstention), as defined in Section 2.3.

Qualitative goal. The goal $\gamma_{ql}(\varphi)$ we define here corresponds to the strong verifiability goal γ_{SV} from Section 5.5. In contrast to γ_{SV}, $\gamma_{ql}(\varphi)$ has the parameter φ for the trust assumptions, which were fixed in γ_{SV}. Informally, this goal requires that, if the trust assumption φ holds true in a protocol run, then (i) the choices of all honest voters who successfully performed their checks are included in the final result, (ii) votes of those honest voters who did not performed their check may be dropped, but not altered, and

(*iii*) there is only at most one ballot cast for every dishonest voter (no ballot stuffing). If the trust assumptions φ are not met in a protocol run, we do not expect the protocol to provide any guarantees in this run. For example, if in a setting with two bulletin boards, φ says that at least one of the bulletin boards should be honest in a run, but in the run considered both have been corrupted by the adversary, then no guarantees need to be provided in this run.

Formally, the goal $\gamma_{ql}(\varphi)$ is satisfied in r (i.e., $r \in \gamma_{ql}(\varphi)$) if either (a) the trust assumption φ does not hold true in r, or if (b) φ holds true in r and there exist valid choices $\tilde{\mathsf{ch}}_1, \ldots, \tilde{\mathsf{ch}}_{n_{\mathsf{voters}}}$ for which the following conditions are satisfied:

1. An election result is published in r and it equals to $f_{\mathsf{res}}(\tilde{\mathsf{ch}}_1, \ldots, \tilde{\mathsf{ch}}_{n_{\mathsf{voters}}})$.
2. The multiset $\{\tilde{\mathsf{ch}}_1, \ldots, \tilde{\mathsf{ch}}_{n_{\mathsf{voters}}}\}$ consists of all the actual choices of honest voters who successfully performed their check, plus a subset of actual choices of honest voters who did not perform their check (successfully), and plus at most $n_{\mathsf{voters}}^{\mathsf{dishonest}}$ additional choices.

If the checks performed by voters do not fully guarantee that their votes are actually counted, because, for example, Benaloh checks were performed (and hence, some probabilistic checking), then along with this goal one will obtain a $\delta > 0$, as there is some probability for cheating going undetected. Also, the requirement that votes of honest voters who do not checked can at most be dropped, but not altered, might only be achievable under certain trust assumptions. If one wants to make weaker trust assumptions, one would have to weaken $\gamma_{ql}(\varphi)$ accordingly.

Quantitative goal. The goal $\gamma_{qn}(k, \varphi)$ of the quantitative verifiability definition is a refinement of the goal γ_k from Section 5.2 (note that now, φ can specify trust assumption with dynamic corruption). The goal $\gamma_{qn}(k, \varphi)$ is the same as the goal $\gamma(k, \varphi)$ that has been described in Section 2.3.2 and that has been applied in Section 3.4 and Section 4.4 to analyze the verifiability level of sElect and Ordinos, respectively.

Chapter 6

Conclusion and Future Work

In this thesis, we have contributed to the field of secure e-voting as follows.

Firstly, we have introduced two new e-voting system, sElect and Ordinos, each of which provides its own balance between usability, security, and practicality. sElect has been designed to be easy to use and to comprehend, yet to still provide a good level of security. Ordinos invokes complex cryptographic primitives and subprotocols in order to hide the tally and thus provide a superior level of privacy. Importantly, we have formally analyzed sElect and Ordinos w.r.t. verifiability, accountability, and privacy. Both systems have been implemented to demonstrate their practicality. One limitation of sElect is that, for privacy reasons, a voter needs to be able to anonymously publish blaming evidence on the bulletin board in case a mix server has manipulated or dropped her vote. Interesting future work includes providing voters with a simple mechanism to establish such an anonymous channel. Regarding Ordinos, it would be interesting to also implement the offline phase for malicious environments and to realize more complex result functions, such as the D'Hondt or Sainte-Laguë methods.

Secondly, we have reviewed all formal definitions of verifiability proposed in the literature and provided a detailed comparison of them. We have thoroughly discussed advantages and disadvantages, and pointed to limitations and problems. Finally, from these discussions, we have distilled a general definition of verifiability, which can be instantiated in various ways, and provide precise guidelines for its instantiation.

Appendix A

Cryptographic Primitives

A.1 Public-Key Encryption

Public-key encryption schemes. A public-key encryption scheme $\mathcal{E}$ consists of a triple of algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where

- KeyGen, the *key generation algorithm*, is a probabilistic algorithm that takes a security parameter ℓ as input and returns a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and secret keys.
- Enc, the *encryption algorithm*, is a probabilistic algorithm that takes a public key pk and a message $\mathsf{m} \in \{0,1\}^*$ as input and outputs a ciphertext c.
- Dec, the *decryption algorithm*, is a deterministic algorithm which takes a secret key sk and a ciphertext c as input and outputs either a message $\mathsf{m} \in \{0,1\}^*$ or a special symbol $\perp$ to indicate that the ciphertext was invalid.

We require that for all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell)$, for all $\mathsf{m} \in \{0,1\}^*$, and for all $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, we have that $\mathsf{Dec}(\mathsf{sk}, \mathsf{c}) = \mathsf{m}$. We also require that KeyGen, Enc and Dec can be computed in polynomial time.

Encryption of vectors. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. Let $\mathsf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_n)$ and $\mathsf{c} = (\mathsf{c}_1, \ldots, \mathsf{c}_n)$ be vectors of entries in $\{0,1\}^*$. We write

$$\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) = (\mathsf{Enc}(\mathsf{pk}, \mathsf{m}_1), \ldots, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_n))$$
$$\mathsf{Dec}(\mathsf{sk}, \mathsf{c}) = (\mathsf{Dec}(\mathsf{sk}, \mathsf{c}_1), \ldots, \mathsf{Dec}(\mathsf{sk}, \mathsf{c}_n))$$

for every public key pk and every secret key sk.

Challenger. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. The *(CCA-) challenger* Ch is a probabilistic polynomial-time algorithm that takes as input a bit b as well as a key pair $(\mathsf{pk}, \mathsf{sk})$, and that serves two types of queries:

- For a vector of messages c, the challenger returns the decryption of c, that is $\mathsf{Dec}(\mathsf{sk}, \mathsf{c})$.
- For a pair of vectors of messages $(\mathsf{m}_0, \mathsf{m}_1)$ where both vectors have the same size and all messages at the same position in the vectors have the same length, the challenger encrypts m_b under pk and returns the vector of ciphertexts, that is $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b)$.

IND-CCA2-security. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with security parameter ℓ and let Ch be the challenger. Then the encryption scheme $\mathcal{E}$ is *IND-CCA2-secure*, if for every polynomially bounded adversary A who never submits decryption queries for (parts of) a vector of messages c previously returned by a challenge query, we have that

$$\begin{aligned} &|\mathsf{Pr}\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell); b' \leftarrow \mathsf{A}^{\mathsf{Ch}(1,\mathsf{pk},\mathsf{sk})}(1^\ell, \mathsf{pk}); b' = 1\right] \\ - \quad &\mathsf{Pr}\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\ell); b' \leftarrow \mathsf{A}^{\mathsf{Ch}(0,\mathsf{pk},\mathsf{sk})}(1^\ell, \mathsf{pk}); b' = 0\right]| \end{aligned}$$

is a negligible function in ℓ.

Threshold Public-Key Encryption Scheme. Let n_{trustees} be the number of trustees T_k and t be a threshold. Let prm be the parameters including the security parameter 1^ℓ.[1] A $(n_{\mathsf{trustees}}, t)$-*threshold public-key encryption scheme* is a tuple of polynomial-time algorithms $(\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$ such that:

- $\mathsf{KeyShareGen}$ (which is run by a single trustee T_k) is probabilistic and outputs two keys $(\mathsf{pk}_k, \mathsf{sk}_k)$, called the *public-key share* pk_k and the *secret-key share* sk_k,
- $\mathsf{PublicKeyGen}$ is deterministic and takes as input public-key shares $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}}$, and outputs a public key pk; this algorithm may fail (output $\perp$) if the public-key shares are invalid,
- Enc is probabilistic and takes as input a public key pk and a message m, and outputs a ciphertext c,
- $\mathsf{DecShare}$ (which is run by a single trustee T_k) is probabilistic and takes as input a ciphertext c and a secret-key share sk_k, and outputs a decryption share dec_k,

[1] We implicitly assume that all algorithms have prm as input.

- Dec is deterministic and takes as input a tuple of decryption shares and returns a message m or $\perp$, in the case that decryption fails.

Furthermore, the following correctness condition has to be guaranteed. Let $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyShareGen}$ for all $k \in \{1, \ldots, n_{\mathsf{trustees}}\}$ and let $\mathsf{pk} \leftarrow \mathsf{PublicKeyGen}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}})$. Let $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, and for all $k \in I \subseteq \{1, \ldots, n_{\mathsf{trustees}}\}$, let $\mathsf{dec}_k \leftarrow \mathsf{DecShare}(\mathsf{sk}_k, \mathsf{c})$. Then, we have

$$\mathsf{Dec}(\{\mathsf{dec}_k\}_{k \in I}) = \begin{cases} \mathsf{m} & \text{if } |I| \geq t \\ \perp & \text{otherwise} \end{cases}.$$

IND-CPA Security. Let $(\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$ be a $(n_{\mathsf{trustees}}, t)$-threshold public-key encryption scheme.

Let $\mathsf{Ch}^{\mathsf{Enc}}$ be a ppt algorithm, called a *(CPA-) challenger*, which takes as input a bit b and a public key pk and (only) serves the second challenge query of the CCA-challenger defined above.

Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$ be an adversary, where $\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3$ share state and A_3 has oracle access to $\mathsf{Ch}^{\mathsf{Enc}}$.

Let $\mathsf{Exp}_{\mathsf{A}}(b)$ be defined as follows:

1. $I \leftarrow \mathsf{A}_1()$ where $I \subseteq \{1, \ldots, n_{\mathsf{trustees}}\}$ and $|I| \geq t$
2. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyShareGen}()$ for $i \in I$
3. $\mathsf{pk}_j \leftarrow \mathsf{A}_2(\{\mathsf{pk}_i\}_{i \in I})$ for $j \in \{1, \ldots, n_{\mathsf{trustees}}\} \setminus I$
4. $\mathsf{pk} \leftarrow \mathsf{PublicKeyGen}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}})$
5. $b' \leftarrow \mathsf{A}_3^{\mathsf{Ch}^{\mathsf{Enc}}(b, \mathsf{pk})}()$
6. output b'

We say that the $(n_{\mathsf{trustees}}, t)$-threshold public-key encryption scheme is *IND-CPA secure* if for all ppt adversaries $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$

$$|\Pr(\mathsf{Exp}_{\mathsf{A}}(0) \text{ outputs } 1) - \Pr(\mathsf{Exp}_{\mathsf{A}}(1) \text{ outputs } 1)|$$

is negligible as a function in the security parameter ℓ.

A.2 Digital Signatures

Signature schemes. A *digital signature scheme* $\mathcal{S}$ is a triple of polynomial-time algorithms $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where

1. KeyGen, the *key generation algorithm*, is a probabilistic algorithm that takes a security parameter ℓ and returns a pair $(\mathsf{Verify}, \mathsf{sign})$ of matching secret signing and public verification keys.

2. Sign, the *signing algorithm*, is a (possibly) probabilistic algorithm that takes a private signing key sign and a message $x \in \{0,1\}^*$ to produce a signature σ.

3. Verify, the *verification algorithm*, is a deterministic algorithm which takes a public verification key Verify, a message $x \in \{0,1\}^*$ and a signature σ to produce a boolean value.

We require that for all key pairs $(\mathsf{Verify}, \mathsf{sign})$ which can be output by the key generation algorithm $\mathsf{KeyGen}(1^\ell)$, for all messages $x \in \{0,1\}^*$, and for all signatures σ that can be output by $\mathsf{Sign}(\mathsf{sign}, x)$, we have that $\mathsf{Verify}(\mathsf{Verify}, x, \sigma)$ equals to true. We also require that KeyGen, Sign and Verify can be computed in polynomial time.

EUF-CMA-secure. Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with security parameter ℓ. Then the signature scheme is *existentially unforgeable under adaptive chosen-message attacks (EUF-CMA-secure)* if for every probabilistic (polynomial-time) algorithm A who has access to a signing oracle and who never outputs tuples (x, σ) for which x has previously been signed by the oracle, we have that

$$\Pr((\mathsf{Verify}, \mathsf{sign}) \leftarrow \mathsf{KeyGen}(1^\ell); \\ (x, \sigma) \leftarrow \mathsf{A}^{\mathsf{Sign}(\mathsf{sign}, \cdot)}(1^\ell, \mathsf{Verify}); \mathsf{Verify}(\mathsf{Verify}, x, \sigma) = \text{true})$$

is negligible as a function in ℓ.

A.3 Non-Interactive Zero-Knowledge Proofs

A.3.1 Definitions

Non-Interactive Proof Systems. Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$, x is called the *statement* and w is called the *witness*. Let $L_R = \{x \mid \exists w \colon (x, w) \in R\}$. A *non-interactive proof system* for the language L_R is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Prover}, \mathsf{Ver})$, where

- Setup (the common reference string generator) takes as input a security parameter 1^ℓ and the statement length n and produces a common reference string $\sigma \leftarrow \mathsf{Setup}(n)$,[2]
- Prover takes as input the security parameter 1^ℓ, a common reference string σ, a statement x, and a witness w and produces a proof $\pi \leftarrow \mathsf{Prover}(\sigma, x, w)$,
- Ver takes as input the security parameter 1^ℓ, a common reference string σ, a statement x, and a proof π and outputs $1/0 \leftarrow \mathsf{Ver}(\sigma, x, w)$ depending on whether it accepts π as a proof of x or not,

such that the following conditions are satisfied:

- *(Computational) Completeness:* Let $n = \ell^{O(1)}$ and A be an adversary that outputs $(x, w) \in R$ with $|x| = n$. Then, the probability

$$\Pr(\sigma \leftarrow \mathsf{Setup}(n); \\ (x, w) \leftarrow \mathsf{A}(\sigma); \pi \leftarrow \mathsf{Prover}(\sigma, x, w); b \leftarrow \mathsf{Ver}(\sigma, x, \pi) \colon b = 1)$$

  is overwhelming (as a function of the security parameter 1^ℓ). In other words, this condition guarantees that an honest prover should always be able to convince an honest verifier of a true statement (which can be chosen by the adversary A).

- *(Computational) Soundness:* Let $n = \ell^{O(1)}$ and A be a non-uniform polynomial time adversary. Then, the probability

$$\Pr(\sigma \leftarrow \mathsf{Setup}(n); (x, \pi) \leftarrow \mathsf{A}(\sigma); b \leftarrow \mathsf{Ver}(\sigma, x, \pi) \colon b = 1 \text{ and } x \notin L_R)$$

  is negligible (as a function of the security parameter 1^ℓ). In other words, this condition guarantees that it should be infeasible for an adversary to come up with a proof π of a false statement x that is nevertheless accepted by the verifier.

Zero-Knowledge. We say that a non-interactive proof system (Setup, $\mathsf{Prover}, \mathsf{Ver}$) is *zero-knowledge* (NIZKP) if the following condition is satisfied.

Let $n = \ell^{O(1)}$. There exists a polynomial-time simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for all stateful, interactive, non-uniform polynomial-time adversaries $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ that output $(x, w) \in R$ with $|x| = n$, we have

$$\Pr(\sigma \leftarrow \mathsf{Setup}(n); (x, w) \leftarrow \mathsf{A}_1(\sigma); \pi \leftarrow \mathsf{Prover}(\sigma, x, w); b \leftarrow \mathsf{A}_2(\pi) \colon b = 1) \\ \approx \Pr((\sigma, \tau) \leftarrow \mathsf{Sim}_1(n); (x, w) \leftarrow \mathsf{A}_1(\sigma); \pi \leftarrow \mathsf{Sim}_2(\sigma, x, \tau); b \leftarrow \mathsf{A}_2(\pi) \colon b = 1)$$

[2]For simplicity of notation, we omit the security parameter in the notation, also for the prover and the verifier.

(where $\approx$ means that the difference between the two probabilities is negligible as a function of the security parameter).

We use here the *single-theorem variant* of the zero-knowledge property, where the common reference string is used to produce (and verify) only one ZK proof, as opposed to the (general) multi-theorem variant of the zero-knowledge property, where the same common reference string can be used to produce many proofs. This suffices for our application, because, in the voting protocol we consider, the number of ZK-proofs is bounded, which corresponds to the case, where A can only submit a bounded number of queries. In such a case, the single-theorem variant of the zero-knowledge property implies the multi-theorem variant (the length of σ can be expanded by a factor of M, where M is the bound on the number of ZKPs).

Proof of Knowledge. We say that a non-interactive proof system $(\mathsf{Setup}, \mathsf{Prover}, \mathsf{Ver})$ produces a *proof of knowledge* if the following condition is satisfied.

There exists a *knowledge extractor* $\mathsf{Extr} = (\mathsf{Extr}_1, \mathsf{Extr}_2)$ such that for $n = \ell^{O(1)}$, the following conditions hold true:

- For all non-uniform polynomial-time adversaries A, we have that

$$\begin{aligned} & \Pr(\sigma \leftarrow \mathsf{Setup}(n); b \leftarrow \mathsf{A}(\sigma) \colon b = 1) \\ \approx \quad & \Pr((\sigma, \tau) \leftarrow \mathsf{Extr}_1(n); b \leftarrow \mathsf{A}(\sigma) \colon b = 1). \end{aligned}$$

- For all non-uniform polynomial-time adversaries A, we have that the probability

$$\begin{aligned} \Pr((\sigma, \tau) \leftarrow \mathsf{Extr}_1(n); (x, \pi) \leftarrow \mathsf{A}(\sigma); \\ w \leftarrow \mathsf{Extr}_2(\sigma, \tau, x, \pi); \\ b \leftarrow \mathsf{Ver}(\sigma, x, \pi) \colon b = 0 \text{ or } (x, w) \in R) \end{aligned}$$

  is overwhelming (as a function of the security parameter).

Note that (computational) knowledge extraction implies the existence of a witness and, therefore, it implies (computational) adaptive soundness.

A.3.2 (NIZK) Proofs used in Ordinos

Let $(\mathsf{KeyShareGen}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Dec})$ be a (threshold) public-key encryption scheme as defined in A.1. 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:

$$\exists \mathsf{sk}_i \colon (\mathsf{pk}_i, \mathsf{sk}_i) \text{ is a valid key share pair.}$$

- *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 B

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 first note that we can model each MPC protocol as a protocol P_{MPC} in the computational model presented in Section 2.2. More precisely, each protocol P_{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 Ordinos voting protocol, in particular, they are all assumed honest (recall Section 4.3 for details).

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

In what follows, we precisely define the security properties, privacy and individual accountability, that the MPC protocol P_{MPC} invoked in Ordinos is supposed to guarantee so that Ordinos provides privacy and accountability (Theorem 6 and 7).

B.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 P_{MPC} provides privacy if it realizes $\mathcal{I}_{\mathsf{MPC}}(\mathcal{E}, f_{\mathsf{MPC}})$, as defined in Fig. B.1.

B.2 Individual Accountability

We require that if the real outcome of P_{MPC} does not correspond to its input, then P_{MPC} provides evidence to individually blame (at least) one misbehaving trustee T_k. More precisely, we require that the protocol P_{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 2.3 for details). Formally, the accountability property Φ of P_{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.

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

Parameters:

- A $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E}$.
- Function $f_{\mathsf{MPC}} \colon \{0,1\}^* \to \{0,1\}^*$.
- Number of honest trustees $n^{\mathsf{honest}}_{\mathsf{trustees}}$.[a]
- $K \leftarrow \emptyset$ (initially).

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

1. If $k \notin \{1, \ldots, n^{\mathsf{honest}}_{\mathsf{trustees}}\}$, return $\perp$.
2. Set $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyShareGen}$ and $K \leftarrow K \cup \{k\}$.
3. 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. Set $\mathsf{sk}_k \leftarrow \mathsf{sk}$ and $K \leftarrow K \cup \{k\}$.
3. Return $\mathsf{success}$.

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.

[a] W.l.o.g., we assume that the first $n^{\mathsf{honest}}_{\mathsf{trustees}}$ are honest.

Figure B.1: Ideal MPC protocol.

Appendix C

Formal Proofs

C.1 Verifiability and Accountability Proof for sElect

In this section, we first formally prove the accountability result of sElect (Theorem 3).

Lemma 1 (Fairness)**.** *Under the assumptions stated at the beginning of Section 3.5 and the mentioned judging procedure run by the judge* J*, the judge* J *is computationally fair in the protocol* $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$*.*

Proving fairness follows immediately from the correctness of the encryption scheme and the signature scheme invoked.

Lemma 2 (Completeness)**.** *Under the assumptions stated at the beginning of Section 3.5 and the mentioned judging procedure run by the judge* J*, for* $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ *we have that*

$$\Pr[\pi(1^\ell) \mapsto \neg(\mathsf{J}\colon \Phi_k)] \leq \delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$$

with overwhelming probability as a function of ℓ*.*

Proof. In order to prove the lemma, we have to show that the probabilities

$\Pr[\pi(1^\ell) \mapsto (\chi_i \wedge \neg\mathsf{dis}(\mathsf{V}_i) \wedge \neg\mathsf{dis}(\mathsf{AS}))]$,
$\Pr[\pi(1^\ell) \mapsto (\chi'_i \wedge \neg\mathsf{dis}(\mathsf{V}_i) \wedge \neg\mathsf{dis}(\mathsf{AS}))]$, and
$\Pr[\pi(1^\ell) \mapsto (\neg\gamma(k,\varphi) \wedge \neg\chi \wedge \neg\mathsf{dis}(\mathsf{AS}) \wedge \neg\mathsf{dis}(\mathsf{M}_1) \wedge \ldots \wedge \neg\mathsf{dis}(\mathsf{M}_{n_{\mathsf{servers}}}))]$

are $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$-bounded for every $i \in \{1, \ldots, n_{\mathsf{voters}}\}$.

The first two probabilities are equal to 0. In fact, if a voter V_i complains in an authenticated way that she did not receive a valid acknowledgement although she submitted a valid ballot (i.e., when χ_i holds true), or if V_i complains in an authenticated way that she abstained from voting although her name appears in a ballot in $\vec{\mathsf{b}}$ (i.e., when χ'_i holds true), then, by the definition of the honest programs, the honest bulletin board B publishes the respective complaint and the judge J outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

To complete the proof, we need to show that the probability of the event

$$X = \neg\gamma(k, \varphi) \wedge \neg\chi \wedge \neg\mathsf{IB}$$

is $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$-bounded as a function of ℓ, where

$$\mathsf{IB} = \mathsf{dis}(\mathsf{AS}) \vee \mathsf{dis}(\mathsf{M}_1) \vee \ldots \vee \mathsf{dis}(\mathsf{M}_{n_{\mathsf{servers}}}).$$

In other words, $\neg\mathsf{IB}$ describes the event that none of the mix servers M or the authentication server AS are individually blamed by the judge J.

Let $\beta = \chi_1 \cup \cdots \cup \chi_n$ and $\beta' = \chi'_1 \cup \cdots \cup \chi'_n$ (note that $\neg\chi = \neg\beta \wedge \neg\beta'$). Finally, let $Y = (\neg\gamma_k \wedge \neg\beta)$. We can now write $X = Y \wedge \neg\beta' \wedge \neg IB$.

To show that X is $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$-bounded, we will show a stronger fact, namely, we will show that $\mathsf{Pr}\left[\neg\mathsf{IB} \wedge \neg\beta' \mid Y\right] \leq \delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$ (assuming that the probability of Y is > 0, as otherwise the proof is trivial).

First, let us observe that in runs in Y the following is true. Since $\neg\beta$ holds true, we know that $\neg\chi_i$ holds true for every voter V_i. In particular, this means that no honest voter who cast a ballot claims that she has not received a valid acknowledgement. Therefore, for all runs in Y, each honest voter must have received a valid acknowledgement if she cast a ballot. It follows that every honest voter who cast a ballot has all the data necessary to individually blame a server (acknowledgement and random coins used to encrypt her vote), if this server manipulates her vote (i.e., if the pair $\alpha^i_{n_{\mathsf{servers}}}$ does not appear in the result list).

Recall that a run r of an instance π of P_{sElect} is determined by the random coins the dishonest parties in π (the adversary) and the honest parties use. Let ω denote the random coins used in r. We can represent ω as $\langle\omega', \omega_v\rangle$ where ω_v are the random coins used by the honest voters to determine whether they check their verification codes (see Section 3.2, the verification phase) and ω' contains the remaining part of ω. Note that ω' completely determines the run of the protocol up to the verification phase. In particular, ω' determines the output of the last mix server and it determines whether the goal $\gamma(k, \varphi)$ is satisfied or not ($\gamma(k, \varphi)$ does not depend on ω_v). Let us interpret ω' as an event, i.e., a set of runs of P_{sElect} where the random coins are partially fixed to be ω' and ω_v is arbitrary. Then there are two possible cases. Either

the adversary is k-risk-avoiding in all runs of ω', and hence, $\omega' \subseteq \gamma(k,\varphi)$, or the adversary is not k-risk-avoiding in all runs of ω', i.e., $\omega' \cap \gamma'_k = \emptyset$. In particular ω' determines whether β is satisfied or not. This means that either $\omega' \subseteq Y$ or $\omega' \cap Y = \emptyset$.

Let Ω_Y be the set of those ω' that are inside Y. To complete the proof, it is enough to show that, for each $\omega' \in \Omega_Y$ we have

$$\mathsf{Pr}\left[\neg\mathsf{IB} \wedge \neg\beta' \mid \omega'\right] \leq \delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}). \tag{C.1}$$

Let us recall that ω' completely determines the run up to the audit coins (which are drawn, when the result is already determined). In particular, ω' determines whether or not a result is published at all. If no result is published, then, by (J1) of the judging procedure, some server will be blamed individually, and hence, IB would be true. So, in this case $\mathsf{Pr}\left[\neg\mathsf{IB} \wedge \neg\beta' \mid \omega'\right] = 0$. Otherwise, if a result is output, ω' also determines

- the set V_1 of those honest voters who did not vote, but are listed in $\vec{\mathsf{id}}$,
- the set V_2 of those honest voters who cast their ballots, but their vote/verification code pairs are not listed in the final result and their ids are not listed in $\vec{\mathsf{id}}$, and
- the set V_3 of those honest voters who cast their ballots, but their vote/verification code pairs are not listed in the final result and their ids are listed in $\vec{\mathsf{id}}$.

One can see, by the definition of the goal $\gamma(k,\varphi)$ (which is violated in ω'), that $|V_1| + |V_2| + 2 \cdot |V_3| > k$ (otherwise this goal would not be violated).

Now given V_1, V_2, V_3, it is easy to compute the probability $\neg\mathsf{IB} \wedge \neg\beta'$ given ω': this events happens only when none of the voters in V_1, V_2, V_3 verifies the result. Note that, indeed, if a voter in V_1 verifies the result, she complains and β' is automatically satisfied. Similarly, if a voter in V_2 or V_3 verifies the result, she complains by providing a valid evidence of misbehaviour (as discussed above) and the judge states individual blame (IB is satisfied). This probability is $(1 - p^{\mathsf{abst}}_{\mathsf{verif}})^{|V_1|}(1 - p^{\mathsf{vote}}_{\mathsf{verif}})^{|V_2|+|V_3|}$, because the voters in V_1, V_2, V_3 carry out the verification process with probability $p^{\mathsf{abst}}_{\mathsf{verif}}$ and $p^{\mathsf{vote}}_{\mathsf{verif}}$, respectively, independently of anything else (the random coins used in this choices are independent of ω' and of each other). Recall that $|V_1| + |V_2| + 2 \cdot |V_3| > k$.

Therefore we have:

$$\begin{aligned}
\mathsf{Pr}\left[\neg\mathsf{IB} \wedge \neg\beta' \mid \omega'\right] &= (1 - p^{\mathsf{abst}}_{\mathsf{verif}})^{|V_1|}(1 - p^{\mathsf{vote}}_{\mathsf{verif}})^{|V_2|+|V_3|} \\
&\leq \max_{k_1+k_2+2\cdot k_3 \geq k+1} (1 - p^{\mathsf{abst}}_{\mathsf{verif}})^{k_1}(1 - p^{\mathsf{vote}}_{\mathsf{verif}})^{k_2+k_3} \\
&= \delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}).
\end{aligned}$$

This completes the proof. □

The proof of the verifiability result (Theorem 2) follows analogously to the accountability proof presented above. To see this, observe that the accountability proof holds true for the set of protocol runs of sElect in which no clashes occur. This set of runs has overwhelming probability under the assumption that the VSDs of honest voters are honest as well which is assumed for accountability. For the verifiability theorem, however, all VSDs can be dishonest and, hence, there is a non-negligible chance that at least one clash occurs (denoted by p_{clash} in Theorem 2). We cannot guarantee that manipulating clashed votes can be detected. For protocol runs without clashes of honest votes, the accountability proof yields an upper bound (i.e., $\delta_k(p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}})$) for the probability that breaking $\gamma(k, \varphi)$ remains undetected. From this, Theorem 2 follows.

C.2 Privacy Proof for sElect

Recall that, by assumption, for all honest voters in $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$, the length of the candidate plaintext as well as the length of the nonce, respectively, have the same size in each run of the protocol, given a security parameter. Also, recall from Section 3.2 that for the public-key encryption scheme we require that for every public key pk and any two plaintexts of the same length their encryption always yields ciphertexts of the same length. It follows that for each mix server M_j, the ciphertext

$$\alpha^i_j = \mathsf{Enc}(\mathsf{pk}_j, \ldots \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{servers}}}, (\mathsf{cand}_i, \mathsf{code}_i); r^i_{n_{\mathsf{servers}}}) \ldots ; r^i_j).$$

computed by an honest voter V_i for M_j must have the same size for all honest voters. Hence, there exists a function η_j in the security parameter such that for every instance $\pi^{(\ell)}$ of $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ and for every honest voter V_i in $\pi^{(\ell)}$ and every run of $\pi^{(\ell)}$, the size $|\alpha^i_j|$ of α^i_j is $\eta_j(\ell)$. In what follows, we simply write $\eta^i_j = \eta^i_j(\ell)$. In order to determine η_j one can take an arbitrary candidate and an arbitrary nonce of correct size and encrypt the pair under the public keys $\mathsf{pk}_{n_{\mathsf{servers}}}, \ldots, \mathsf{pk}_j$.

In order to prove the privacy theorem for $P_{\mathsf{sElect}}(n_{\mathsf{voters}}, n_{\mathsf{servers}}, \mu, p^{\mathsf{vote}}_{\mathsf{verif}}, p^{\mathsf{abst}}_{\mathsf{verif}}, f_{\mathsf{sElect}})$ with the voter $\mathsf{V}_{\mathsf{obs}}$ under observation, we have to show that

$$\left|\mathsf{Pr}[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_0)\|\pi^*)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_1)\|\pi^*)^{(\ell)} \mapsto 1]\right| \qquad \text{(C.2)}$$

is $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)}(f_{\mathsf{sElect}})$-bounded as a function of the security parameter ℓ, for all choices $\mathsf{ch}_0, \mathsf{ch}_1$ ($\mathsf{ch}_0, \mathsf{ch}_1 \neq \mathsf{abstain}$) and all programs π^* of the

remaining parties such that at least $n^{\mathsf{honest}}_{\mathsf{voters}}$ voters are honest in π^* (excluding the voter under observation $\mathsf{V}_{\mathsf{obs}}$) and such that the adversary (the dishonest parties in π^*) is k-risk-avoiding.

We can split up the composition π^* in its honest and its (potentially) dishonest part. Let HV be the set of all honest voters (without the voter under observation) and $\hat{\pi}_{\mathsf{HV}}$ be the composition of their honest programs. Recall that the judge J, the scheduler S, the bulletin board B, the voting authority Auth, and at least one out of mix server are honest (w.l.o.g., we assume that the j-th mix server is honest). Therefore, the honest part, which we denote by

$$\hat{\pi}_{\mathsf{H}} = \hat{\pi}_{\mathsf{J}} \| \hat{\pi}_{\mathsf{Auth}} \| \hat{\pi}_{\mathsf{B}} \| \hat{\pi}_{\mathsf{S}} \| \hat{\pi}_{\mathsf{M}_j} \| \hat{\pi}_{\mathsf{HV}},$$

consists of the honest programs $\hat{\pi}_{\mathsf{J}}, \hat{\pi}_{\mathsf{Auth}}, \hat{\pi}_{\mathsf{B}}, \hat{\pi}_{\mathsf{S}}, \hat{\pi}_{\mathsf{T}_k}, \hat{\pi}_{\mathsf{HV}}$ of the judge J, the voting authority Auth, the bulletin board B, the scheduler S, the mix server M_j, and the honest voters HV, respectively. By $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})$ we will denote the composition of all honest programs including the program of the voter under observation $\mathsf{V}_{\mathsf{obs}}$ voting for ch, i.e., $\hat{\pi}_{\mathsf{H}}(\mathsf{ch}) = \hat{\pi}_{\mathsf{H}} \| \hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})$. All remaining parties are subsumed by the adversarial process π_{A}. This means that we can write $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}) \| \pi^*$ as $\hat{\pi}_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$. Recall that, by assumption, the adversary π_{A} is k-risk-avoiding.

In order to prove the result, we use a sequence of games. We fix $\mathsf{ch} \in \mathsf{C}$ and start with Game 0 which is simply the process $\hat{\pi}_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$. Step by step, we transform Game 0 into Game 4 which is the composition $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$ for some process $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ and the same adversarial process π_{A}. Game 4 will be proven indistinguishable from Game 0 from the adversary's point of view, which means that

$$\left| \Pr[(\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}) \mapsto 1] \right|$$

is negligible for a fixed $\mathsf{ch} \in \mathsf{C}$ (as a function of the security parameter). On the other hand, it will be straightforward to show that in Game 4 for arbitrary $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C} \setminus \{\mathsf{abstain}\}$, the distance

$$\left| \Pr[(\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}_0) \| \pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}_1) \| \pi_{\mathsf{A}}) \mapsto 1] \right|$$

is bounded by $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)}(f_{\mathsf{sElect}})$ because $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}_0)$ and $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch}_0)$ use the ideal voting protocol for $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters. Using the triangle inequality, we can therefore deduce that

$$|\Pr[(\hat{\pi}_{\mathsf{H}}(\mathsf{ch}_0) \| \pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}(\mathsf{ch}_1) \| \pi_{\mathsf{A}}) \mapsto 1]|$$

is $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)}(f_{\mathsf{sElect}})$-bounded for all $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C}$ (as a function of the security parameter).

Game 0. In what follows we write $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ for $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})$ and consider $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ as one atomic process (one program) and not as a composition of processes.[1] Now, Game 0 is simply the process $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$. △

In the first step, we construct Game 1 which will be proven indistinguishable from Game 0 in Claim 1 based on the IND-CCA2-security of the public-key encryption scheme. More precisely, the adversary will only receive fake ballots encrypting a random string at the beginning. These fake ballots will then be replaced in the honest mixing phase by ciphertexts encrypting the real choices.

Game 1. For Game 1, we modify $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ in the following way in order to obtain $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ are identical.

Ballot creation (simulated): Recall that, in order to create her ballot α^i_0, an honest voter V_i first chooses a candidate (either cand, if under observation, or according to μ, otherwise) and a nonce code_i, and then encrypts the tuple under the public keys of the mix servers, starting with the public key $\mathsf{pk}_{n_{\mathsf{servers}}}$ of the last mix server and then going to the public key pk_1 of the first mix server.

To simulate the process $\hat{\pi}_{\mathsf{V}_i}$ of an arbitrary honest voter V_i, the process $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ follows $\hat{\pi}_{\mathsf{V}_i}$ until the encryption of V_i's choice under the public key pk_{j+1} of the mix server M_{j+1}: $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ first chooses a candidate and a nonce as before and encrypts it with the public keys $\mathsf{pk}_{n_{\mathsf{servers}}}, \ldots, \mathsf{pk}_{j+1}$ of the mix servers after the honest mix server M_j to obtain α^i_j (which is supposed to be the output of M_j). Now, however, $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ does not encrypt α^i_j (containing the choice) further. Instead, $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ encrypts 0^{η_j} under the remaining public keys $\mathsf{pk}_j, \mathsf{pk}_{j-1}, \ldots, \mathsf{pk}_1$ to obtain the ciphertexts $\alpha^i_{j-1}, ..., \alpha^i_0$, where η_j is defined as above. The pair $\alpha^i_j, \alpha^i_{j-1}$ is logged by $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ for replacement later on. After that and before simulating the process $\hat{\pi}_{\mathsf{M}_j}$ of the honest mix server M_j, $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ are identical. This means that the ciphertext α^i_0 encrypting 0^{η_j} is supposed to fake the ballot of V_i.

Honest mixing (simulated): $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ simulates $\hat{\pi}_{\mathsf{M}_j}$ in the following way. Let $\vec{\mathsf{c}}_{j-1}$ be the input to the (simulated) honest mix server M_j (from the adversary's point of view). For all voters V_i whose associated ciphertext α^i_{j-1} is in $\vec{\mathsf{c}}_{j-1}$ (recall that ciphertexts can be dropped or manipulated by the adversary), $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ adds α^i_j to its output $\vec{\mathsf{c}}_j$ (which is supposed to fake the output of the honest mix server M_j). Apart from this, $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ follows $\hat{\pi}_{\mathsf{M}_j}$. In particular, if the input to M_j contains a ciphertext c which has not been logged before as α^i_{j-1} (for some i), then this ciphertext is decrypted (using the decryption key of M_j) and, if successful, added to the output of M_j. △

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

Game 1 and Game 2 are completely identical with the difference being that the simulator (i.e., $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ in Game 1 and $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ in Game 2) halts if the adversary dropped or manipulated more than k honest voters' ciphertexts prior to the (simulated) honest mix server M_j. In Claim 3, we will show that this can only happen with negligible probability if the adversary is k-risk-avoiding. Therefore, Game 1 and Game 2 can be proven computationally indistinguishable as stated in Claim 4.

Game 2. The process $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ is completely identical to $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ with only one modification: $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ halts if there are less than $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ ciphertexts associated to the honest voters in the input $\vec{\mathsf{c}}_{j-1}$ of the (simulated) honest mix server M_j. In this case, $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ halts when it is triggered the first time after $\vec{\mathsf{c}}_{j-1}$ has been published. △

We modify $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ in such a way that the point when the honest voters are supposed to pick their candidates is postponed to the point when the honest mix server is triggered to mix its input. Game 2 and Game 3 are perfectly indistinguishable as stated in Claim 5.

Game 3. For Game 3, we modify $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ in the following way in order to obtain $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ are identical.

Ballot creation (simulated): Let V_i be an arbitrary honest voter. In contrast to $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$, $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ does not pick a choice when creating the ballot α^i_0 and therefore does not encrypt the choice under the public keys $\mathsf{pk}_{n_{\mathsf{servers}}}, \ldots, \mathsf{pk}_{j+1}$ to obtain α^i_j. Instead, $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ encrypts 0^{η_j} under the first j public keys $\mathsf{pk}_j, \mathsf{pk}_{j-1}, \ldots, \mathsf{pk}_1$ in reverse order to build and publish the fake ballot α^i_0 for the voter V_i as in $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$. The pair $(\alpha^i_{j-1}, \mathsf{V}_i)$ is logged by $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$.

Honest mixing (simulated): Let $\vec{\mathsf{c}}_{j-1}$ be the input to the (simulated) honest mix server M_j (from the adversary's point of view). For all voters V_i whose associated ciphertext α^i_{j-1} is in $\vec{\mathsf{c}}_{j-1}$, $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ picks a choice (either ch or according to μ, respectively) and encrypts it under the public keys $\mathsf{pk}_{n_{\mathsf{servers}}}, \ldots, \mathsf{pk}_{j+1}$ to obtain α^i_j. Afterwards $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ adds α^i_j to the output $\vec{\mathsf{c}}_j$ (which is supposed to fake the output of M_j). Apart from this, $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ follows $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$. △

The only difference between Game 3 and Game 4 is that the simulator invokes the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{sElect}}, n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)$ with $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters (see Section 2.5.2) in order to receive $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest choices including the choice of the voter under observation. The simulator receives these choices in a completely random order and as plaintexts. If necessary, the simulator generates remaining choices itself. Then it continues as before. As stated in Claim 6 both games are perfectly indistinguishable. Additionally, and this is the main idea of the proof, the advantage of the ad-

versary to tell what the voter under observation voted for is bounded by the privacy level $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{sElect}})$ of the ideal voting protocol. To see this, assume that the adversary also controlled the simulator without the ideal voting protocol for $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters. In this case, the adversary's advantage is obviously bounded by $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{sElect}})$. This is Claim 7.

Game 4. $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ is identical to $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ except for the simulation of the honest mix server M_j. In the honest mixing phase the process $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ (which is now independent of ch) uses the ideal voting protocol (which now depends on ch) to generate the choices of the first $n^{\mathsf{honest}}_{\mathsf{voters}} - k - 1$ honest voters plus the voter under observation (as described below).

Honest mixing (simulated): Let $\vec{\mathsf{c}}_{j-1}$ be the input to the (simulated) honest mix server M_j. Note that, according to Game 2, $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$ halts if $\vec{\mathsf{c}}_{j-1}$ contains less that $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ ciphertexts associated to the honest voters. This is done in $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ as well. Otherwise, at the beginning of the honest mixing phase, $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ triggers the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{sElect}}, n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}} - k, \mu)$ (see Section 2.5.2); we implicitly compose the program of the observer in the ideal voting protocol with the adversary π_{A}. The ideal voting protocol then outputs the list of permutated choices (as plaintexts) that have been chosen by $n^{\mathsf{honest}}_{\mathsf{voters}} - k - 1$ honest voters plus the voter under observation (inside the ideal voting protocol and independently of $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$). Now, $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ counts the number κ of ciphertexts α^i_{j-1} associated to the honest voters in the input $\vec{\mathsf{c}}_{j-1}$ of the (simulated) honest mix server M_j. If $\kappa - (n^{\mathsf{honest}}_{\mathsf{voters}} - k) > 0$, then $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ picks $\kappa - (n^{\mathsf{honest}}_{\mathsf{voters}} - k)$ candidates according to μ. Afterwards, $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ has a list of κ plaintexts: $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ from the ideal functionality and $\kappa - (n^{\mathsf{honest}}_{\mathsf{voters}} - k)$ generated by itself. Then $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ encrypts each of these vote-nonce pairs under the public keys $\mathsf{pk}_{n_{\mathsf{servers}}}, \ldots, \mathsf{pk}_{j+1}$ as in the previous games. Afterwards, it adds them to the output $\vec{\mathsf{c}}_j$. The rest of $\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})$ is identical to $\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch})$. △

We prove that each game is computationally (or even perfectly) indistinguishable of the previous one (if any). We will also prove that in the final game (Game 4), for every adversary π_{A} the advantage to correctly tell the choice the voter under observation voted for is bounded by the privacy level of the ideal voting protocol for $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters. This result in combination with the indistinguishability of all games yields Theorem 4.

Claim 1. *Game 0 and Game 1 are computationally indistinguishable, i.e., we have that*

$$\left|\mathsf{Pr}[(\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1]\right| \tag{C.3}$$

is negligible as a function of ℓ.

Proof. We prove that, if π_{A} can distinguish between Game 0 and Game 1 for some choice ch, then there exists an attacker $\pi_{\mathsf{A}'}$ who can break the IND-CCA2-security of the public-key encryption scheme by using π_{A} (see Section A.1). So, let us assume that the difference (C.3) is *non-negligible*.

Let $\hat{\pi}_C(b) = \hat{\pi}_C(\mathsf{pk}_j, \mathsf{sk}_j, b)$ be a (CCA2-) challenger as in Section A.1, meaning that $\hat{\pi}_C(b)$ outputs a ciphertext of x_b under pk_j when given two vectors (x_0, x_1). In what follows, we will construct an attacker $\pi_{\mathsf{A}'}$ on the public-key encryption scheme with key pair $(\mathsf{pk}_j, \mathsf{sk}_j)$ such that for the adversary π_{A} the process $\pi_{\mathsf{A}} \| \pi_{\mathsf{A}'} \| \hat{\pi}_C(b)$ is identical to $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$, if $b = 0$, and to $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch}) \| \pi_A$, if $b = 1$.

The process $\pi_{\mathsf{A}'}$ is defined to be identical to $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ until the encryption of the ciphertexts α^i_j under the public key pk_j. This step is modified in the following way. The attacker $\pi_{\mathsf{A}'}$ sends two vectors to the challenger $\hat{\pi}_C(b)$: the first vector contains the ciphertexts α^i_j of all honest voters and the second vector contains 0^{η_j} at each position. Then, using the public key pk_j, the challenger $\hat{\pi}_C(b)$ encrypts and returns the first vector, if $b = 0$, and the second vector, if $b = 1$. Afterwards and until the end, $\pi_{\mathsf{A}'}$ follows the process $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ with one exception explained below. In particular, this means that $\pi_{\mathsf{A}'}$ encrypts the vector of ciphertexts it has received from the challenger under the remaining public keys $\mathsf{pk}_{j-1}, ..., \mathsf{pk}_1$. Also, for each honest voter V_i whose associated ciphertext α^i_{j-1} is in $\vec{\mathsf{c}}_{j-1}$, $\pi_{\mathsf{A}'}$ adds α^i_j to the output $\vec{\mathsf{c}}_j$. The only difference between $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ and $\pi_{\mathsf{A}'}$ at this point is that whenever $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ would decrypt a ciphertext c (where c is in the input to M_j but it is none of the logged α^i_{j-1}), $\pi_{\mathsf{A}'}$ obtains the decryption of c by querying the decryption oracle of the challenger.

Observe that for $b = 0$, the ciphertext α^i_{j-1} is an encryption of α^i_j under pk_j as in Game 0, and for $b = 1$, the ciphertext α^i_{j-1} is an encryption of 0^{η_j} under pk_j as in Game 1. Also note that in neither game, and hence, also not in $\pi_{\mathsf{A}} \| \pi_{\mathsf{A}'} \| \hat{\pi}_C(b)$ honest voters blame the honest mix server M_j since if α^i_{j-1} is in the input of M_j, M_j adds α^i_j to its output. In particular, honest voters do not need to be able to provide evidence for the misbehavior of M_j, which in $\pi_{\mathsf{A}} \| \pi_{\mathsf{A}'} \| \hat{\pi}_C(b)$ they would not be able to do since encryption is done by the challenger. Now, it is straightforward to see that from the point of view of the adversary π_{A} the process $\pi_{\mathsf{A}} \| \pi_{\mathsf{A}'} \| \hat{\pi}_C(b)$ is identical to Game 0, if $b = 0$, and to Game 1, if $b = 1$.

By assumption, the adversary π_{A} can distinguish between Game 0 and Game 1. Defining that $\pi_{\mathsf{A}'}$ outputs 0, if π_{A} outputs 1, and 1, otherwise, the attacker $\pi_{\mathsf{A}'}$ has a non-negligible advantage in the IND-CCA2-security game with the challenger $\hat{\pi}_C(b)$ (see Section A.1). Therefore, the IND-CCA2-security of the public-key encryption scheme is broken, in contradiction to the assumption that the public-key encryption scheme is IND-CCA2-secure.

Therefore, the claim follows. □

Claim 2. *The adversary π_{A} is k-risk-avoiding in Game 1 (meaning that with overwhelming probability a run of the system does not stop before $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ is published and there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ vote-nonce pairs in $\vec{\mathsf{c}}_{n_{\mathsf{voters}}^{\mathsf{honest}}}$ chosen by honest voters).*

Proof. By assumption, π_{A} is k-risk-avoiding in Game 0. Now, the claim follows immediately from the proof of Claim 1: one could modify $\pi_{\mathsf{A}'}$ in such a way that it checks whether $\gamma(k, \varphi)$ is satisfied or not (which $\pi_{\mathsf{A}'}$ can do efficiently). □

Claim 3. *The probability that in a run of the process $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch}) \| \pi_A$, there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters in the input $\vec{\mathsf{c}}_{j-1}$ of the (simulated) honest mix server M_j is overwhelming.*

Proof. From Claim 2 we know that with overwhelming probability in a run of $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch}) \| \pi_{\mathsf{A}}$ there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ vote-nonce pairs in $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ that have been chosen by different honest voters. We will now show that the probability (over all possible runs of the process $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch}) \| \pi_{\mathsf{A}}$) that there are *less* than $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters in the input $\vec{\mathsf{c}}_{j-1}$ of the (simulated) honest mix server M_j is negligible as a function of the security parameter ℓ.

Let r be an arbitrary run of the system $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch}) \| \pi_{\mathsf{A}}$ in which there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ vote-nonce pairs of honest voters in $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ while there are less than $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters in the input of the honest mix server $\vec{\mathsf{c}}_{j-1}$. Then there exists an honest voter $\hat{\pi}_{\mathsf{V}_i}$ (or $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})$) whose associated ciphertext α_{j-1}^i is not in $\vec{\mathsf{c}}_{j-1}$ while her vote-nonce pair $(\mathsf{cand}_i, \mathsf{code}_i^{\mathsf{voter}})$ is in the final output $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$. Since α_{j-1}^i is not in $\vec{\mathsf{c}}_{j-1}$, the process $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch})$ does not add α_j^i to its output $\vec{\mathsf{c}}_j$. By the definition of Game 1, it is straightforward to see that therefore, the adversary π_{A} does not receive any information about the vote-nonce pair $(\mathsf{cand}_i, \mathsf{code}_i^{\mathsf{voter}})$ throughout the whole run. That is, the run is independent of $\mathsf{code}_i^{\mathsf{voter}}$.

Let $\hat{\pi}_{\mathsf{V}_i}$ be an arbitrary honest voter. We split up the set Ω of random bit strings used to determine the runs of $\hat{\pi}_{\mathsf{H}}^1(\mathsf{ch}) \| \pi_{\mathsf{A}}$ into Ω_i which consists of all random coins used to determine the nonce of $\hat{\pi}_{\mathsf{V}_i}$ and Ω_i' which consists of the remaining random coins. This means that Ω can be represented as $\Omega_i \times \Omega_i'$. Now, let E_i be the event that α_{j-1}^i is not in $\vec{\mathsf{c}}_{j-1}$ while the vote-nonce pair of $\hat{\pi}_{\mathsf{V}_i}$ is in the final output $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ (recall that π_{A} is k-risk-avoiding and thus $\vec{\mathsf{c}}_{n_{\mathsf{servers}}}$ is published).

Let r be a run in E_i and let $\omega \in \Omega$ be the random coins of this run. Let $\omega_{\mathsf{code}^{\mathsf{voter}}} \in \Omega_i$ be the random coins used to determine the nonce of $\hat{\pi}_{\mathsf{V}_i}$ in r

and $\omega' \in \Omega'_i$ be the remaining random coins. This means that $\langle \omega_{\mathsf{code}^{\mathsf{voter}}}, \omega' \rangle$ represents ω as described above. Note that by ω' up to and including the publication of the result, the view of the adversary is completely determined and independent of $\omega_{\mathsf{code}^{\mathsf{voter}}}$. Thus we have that

$$\mathsf{Pr}\left[E_i \mid \omega'\right] \leq \frac{n_{\mathsf{voters}}}{2^\ell}. \tag{C.4}$$

(Recall that n_{voters} is the number of voters and ℓ is the size of the verification codes of honest voters.) Therefore, we know that

$$\mathsf{Pr}\left[E_i\right] = \sum_{\omega' \in \Omega'_i} \mathsf{Pr}\left[E_i \mid \omega'\right] \cdot \mathsf{Pr}\left[\omega'\right] = \sum_{\omega' \in \Omega'_i} \frac{n_{\mathsf{voters}}}{2^\ell} \cdot \mathsf{Pr}\left[\omega'\right] = \frac{n_{\mathsf{voters}}}{2^\ell} \tag{C.5}$$

holds true.

Now, let E be the event that there are less than $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters in the input $\vec{\mathsf{c}}_{j-1}$ of the (simulated) honest mix server M_j in the process $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$. Let E' be the event that an adversary in a run of $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}}$ is k-risk-avoiding. Since the probability for E' is overwhelming, it suffices to show that the probability of $E \cap E'$ is negligible. From what we have shown above, we can conclude that

$$\mathsf{Pr}\left[E \cap E'\right] = \mathsf{Pr}\left[\bigcup_{i=1}^{n_{\mathsf{voters}}^{\mathsf{honest}}} E_i\right] \leq \sum_{i=1}^{n_{\mathsf{voters}}^{\mathsf{honest}}} \mathsf{Pr}\left[E_i\right] = \frac{n_{\mathsf{voters}} \cdot n_{\mathsf{voters}}^{\mathsf{honest}}}{2^\ell}. \tag{C.6}$$

Hence, $\mathsf{Pr}\left[E \cap E'\right]$ is negligible. □

Claim 4. *Game 1 and Game 2 are computationally indistinguishable, i.e., for each choice* ch *we have that*

$$\left|\mathsf{Pr}[(\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1]\right| \tag{C.7}$$

is negligible as a function of ℓ.

Proof. This follows immediately from Claim 3. □

Claim 5. *Game 2 and Game 3 are perfectly indistinguishable, i.e., for each choice* ch *we have that*

$$\left|\mathsf{Pr}[(\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}^3_{\mathsf{H}}(\mathsf{ch}) \| \pi_{\mathsf{A}})^{(\ell)} \mapsto 1]\right| = 0 \tag{C.8}$$

holds true.

Proof. The postponed construction of all α_j^i in Game 3 has no impact on the information the adversary can derive throughout the game because the ciphertexts α_j^i in Game 2 are not output before the honest mixing phase. Therefore, the claim holds true. □

Claim 6. *Game 3 and Game 4 are perfectly indistinguishable, i.e., for each choice* ch *we have that*

$$\left|\mathsf{Pr}[(\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1]\right| = 0 \tag{C.9}$$

holds true.

Proof. The only difference between Game 3 and Game 4 is the fact that in Game 4 $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ honest choices are not generated by $\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})$ but by the ideal voting protocol. But this is done in the same way. So the two games are essentially identical. □

Claim 7. *For Game 4, we have that*

$$\left|\mathsf{Pr}[(\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch}_0)\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch}_1)\|\pi_{\mathsf{A}})^{(\ell)} \mapsto 1]\right|$$

is bounded by $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}-k,\mu)}(f_{\mathsf{sElect}})$ *for all choices* ch_0 *and* ch_1 *(*$\mathsf{ch}_0, \mathsf{ch}_1 \neq \mathsf{abstain}$*).*

Proof. This follows immediately from Theorem 1 (privacy level of the ideal voting protocol). □

From these claims, Theorem 4 follows immediately.

C.3 Verifiability and Accountability Proof for Ordinos

Lemma 3 (Fairness)**.** *Under the assumptions stated at the beginning of Section 4.4 and the mentioned judging procedure run by the judge* J*, the judge* J *is computationally fair in the protocol* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$*.*

Proving fairness follows immediately from the correctness of the encryption scheme, the signature scheme, the MPC protocol, and all the NIZKPs invoked.

Lemma 4 (Completeness)**.** *Under the assumptions stated at the beginning of Section 4.4 and the mentioned judging procedure run by the judge* J*, for* $P_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, f_{\mathsf{tally}})$ *we have that*

$$\Pr[\pi(1^\ell) \mapsto \neg(\mathsf{J}\colon \Phi_k)] \leq \delta_k(p_{\mathsf{verify}})$$

with overwhelming probability as a function of ℓ*.*

Proof. In order to prove the lemma, we have to show that the probabilities

$$\Pr[\pi(1^\ell) \mapsto (\chi_i \wedge \neg\mathsf{dis}(\mathsf{V}_i) \wedge \neg\mathsf{dis}(\mathsf{AS}))],$$
$$\Pr[\pi(1^\ell) \mapsto (\chi_i' \wedge \neg\mathsf{dis}(\mathsf{V}_i) \wedge \neg\mathsf{dis}(\mathsf{AS}))], \text{ and}$$
$$\Pr[\pi(1^\ell) \mapsto (\neg\gamma(k,\varphi) \wedge \neg\chi \wedge \neg\mathsf{dis}(\mathsf{AS}) \wedge \neg\mathsf{dis}(\mathsf{T}_1) \wedge \ldots \wedge \neg\mathsf{dis}(\mathsf{T}_{n_{\mathsf{trustees}}}))]$$

are $\delta_k(p_{\mathsf{verify}})$-bounded for every $i \in \{1, \ldots, n_{\mathsf{voters}}\}$.

The first two probabilities are equal to 0. In fact, if a voter V_i complains in an authenticated way that she did not receive a valid acknowledgement although she submitted a valid ballot (i.e., when χ_i holds true), or if V_i complains in an authenticated way that she abstained from voting although her name appears in a ballot in $\vec{\mathsf{b}}$ (i.e., when χ_i' holds true), then, by the definition of the honest programs, the honest bulletin board B publishes the respective complaint and the judge J outputs the verdict $\mathsf{dis}(\mathsf{V}_i) \vee \mathsf{dis}(\mathsf{AS})$.

To complete the proof, we need to show that the probability of the event

$$X = \neg\gamma(k,\varphi) \wedge \neg\chi \wedge \neg\mathsf{IB}$$

is $\delta_k(p_{\mathsf{verify}})$-bounded as a function of ℓ, where

$$\mathsf{IB} = \mathsf{dis}(\mathsf{AS}) \vee \mathsf{dis}(\mathsf{T}_1) \vee \ldots \vee \mathsf{dis}(\mathsf{T}_{n_{\mathsf{trustees}}}).$$

In other words, $\neg\mathsf{IB}$ describes the event that none of the trustees T or the authentication server AS are individually blamed by the judge J.

Let us first consider the case that an election outcome res is announced. This implies that all NIZKPs that are supposed to be published have in fact been published.

Now, if $\neg\mathsf{IB}$ holds true, then all NIZKPs $\pi_k^{\mathsf{KeyShareGen}}$ published by the trustees T_k are valid. Thus, by the computational completeness of the NIZKPs, it follows that for all $k \in \{1, \ldots, n_{\mathsf{trustees}}\}$ the published public key share pk_k is valid, i.e., there exists a secret key share sk_k such that $(\mathsf{pk}_k, \mathsf{sk}_k)$ is a valid public/secret key pair.

Furthermore, if $\neg\mathsf{IB}$ holds true, then for all ballots $\mathsf{b}_i \in \vec{\mathsf{b}}$ published by AS the NIZKPs π_i^{Enc} are valid (which are supposed to prove that each voter V_i votes for exactly one possible choice). Thus, by the computational completeness of the NIZKPs, it follows that for all $\mathsf{b}_i \in \vec{\mathsf{b}}$ containing a ciphertext vector $(\mathsf{c}_{i,1}, \ldots, \mathsf{c}_{i,n_{\mathsf{cand}}})$, there exist plaintexts $\mathsf{m}_{i,1}^{\mathsf{real}}, \ldots, \mathsf{m}_{i,n_{\mathsf{cand}}}^{\mathsf{real}}$ such that $\mathsf{c}_{i,j}$ encrypts $\mathsf{m}_{i,j}^{\mathsf{real}}$ under pk for all $j \in \{1, \ldots, n_{\mathsf{cand}}\}$ and that $(\mathsf{m}_{i,1}^{\mathsf{real}}, \ldots, \mathsf{m}_{i,n_{\mathsf{cand}}}^{\mathsf{real}}) \in \mathsf{C}$ holds true (recall that $\mathsf{C} \subseteq \{0, \ldots, n_{\mathsf{vpc}}\}^{n_{\mathsf{cand}}}$ is the set of possible choices).

Since we have assumed that the MPC protocol P_{MPC} provides individual accountability for the goal $\gamma_{\mathsf{MPC}}(\varphi)$, it follows that if $\neg\mathsf{IB}$ holds true, then

the overall NIZKP π^{MPC} of the MPC protocol, which has been run among the trustees, is valid. Recall that the goal $\gamma_{\mathsf{MPC}}(\varphi)$ contains all runs in which for the input to P_{MPC}, which equals

$$\mathsf{Enc}\left(\sum_{i=1}^{\mathsf{n_{ballots}}} \mathsf{m}_{i,1}^{\mathsf{real}}\right), \ldots, \mathsf{Enc}\left(\sum_{i=1}^{\mathsf{n_{ballots}}} \mathsf{m}_{i,n_{\mathsf{cand}}}^{\mathsf{real}}\right)$$

in this case, it is guaranteed that the output

$$f_{\mathsf{tally}}\left(\sum_{i=1}^{\mathsf{n_{ballots}}} \mathsf{m}_{i,1}^{\mathsf{real}}, \ldots, \sum_{i=1}^{\mathsf{n_{ballots}}} \mathsf{m}_{i,n_{\mathsf{cand}}}^{\mathsf{real}}\right)$$

of P_{MPC}, and hence of P_{Ordinos}, is correct (with overwhelming probability as a function of ℓ).

Let $\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}}$ be the actual choices made by the honest voters. Now, if the goal $\gamma(k, \varphi)$ is not met, then φ holds true so that, in particular, the bulletin board B is honest. Thus, for all possible valid $\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}^{\mathsf{dishonest}}}$ made by the dishonest voters, we have that the distance d, as measured in Section 2.3.2, between the vector $(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}}, \mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}^{\mathsf{dishonest}}})$ and the vector $(\mathsf{ch}_1^{\mathsf{real}}, \ldots, \mathsf{ch}_{\mathsf{n_{ballots}}}^{\mathsf{real}})$ is at least $k+1$. In other words, the authentication server AS receives the (encrypted) honest input $(\mathsf{ch}_1, \ldots, \mathsf{ch}_{n_{\mathsf{voters}}^{\mathsf{honest}}})$ together with a possibly dishonest input $(\mathsf{ch}'_1, \ldots, \mathsf{ch}'_{n_{\mathsf{voters}}^{\mathsf{dishonest}}})$ and outputs a list of ballots that encrypts $(\mathsf{ch}_1^{\mathsf{real}}, \ldots, \mathsf{ch}_{\mathsf{n_{ballots}}}^{\mathsf{real}})$ which differs from the input on at least $\lceil \frac{k+1}{2} \rceil$ positions. Since we quantify existentially over all possible dishonest inputs and $\mathsf{n_{ballots}} \leq n_{\mathsf{voters}}^{\mathsf{honest}} + n_{\mathsf{voters}}^{\mathsf{dishonest}}$, we can conclude that at least $\lceil \frac{k+1}{2} \rceil$ honest choices have been dropped or manipulated.

Recall that we want to prove that the probability of the event

$$X = \neg\gamma(k, \varphi) \wedge \neg\chi \wedge \neg\mathsf{IB}$$

is $\delta_k(p_{\mathsf{verify}})$-bounded as a function of ℓ. So far, we have analyzed the event $\neg\gamma(k, \varphi) \wedge \neg\mathsf{IB}$ and concluded that, with overwhelming probability, the authentication server AS has dropped or manipulated at least $\lceil \frac{k+1}{2} \rceil$ honest inputs. Under the assumption that all honest voters perform their verification procedure independently from each other, the probability that none of the betrayed honest voters complains is bounded by $\delta_k(p_{\mathsf{verify}}) = (1 - p_{\mathsf{verify}})^{\lceil \frac{k+1}{2} \rceil}$. Thus, we can conclude that the probability of the event X is $\delta_k(p_{\mathsf{verify}})$-bounded as a function of ℓ.

In the case that no election outcome res is announced, the judging procedure (J1) ensures that the authentication server AS or one of the trustees T_k are individually blamed. □

C.4 Privacy Proof for Ordinos

Overview of the proof. Recall that, in order to prove the theorem for the protocol Ordinos with n_{voters} voters, n_{trustees} trustees, voting distribution μ, verification rate $p_{\mathsf{verify}} \in [0,1]$, and voter under observation $\mathsf{V}_{\mathsf{obs}}$, we have to show that

$$|\Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_0)\|\pi^*) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch}_1)\|\pi^*) \mapsto 1]|$$

is $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}-k, \mu)}(f_{\mathsf{res}})$-bounded as a function of the security parameter ℓ, for all $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C}$ ($\mathsf{ch}_0, \mathsf{ch}_1 \neq \mathsf{abstain}$), all programs π^* of the remaining parties such that at least $n^{\mathsf{honest}}_{\mathsf{voters}}$ voters are honest in π^* (excluding the voter under observation $\mathsf{V}_{\mathsf{obs}}$), such that at most $t-1$ trustees are dishonest in π^*, and such that the adversary (the dishonest parties in π^*) is k-risk-avoiding.

We can split up the composition π^* in its honest and its (potentially) dishonest part. Let HV be the set of all honest voters (without the voter under observation) and $\hat{\pi}_{\mathsf{HV}}$ be the composition of their honest programs. Recall that the judge J, the scheduler S, the bulletin board B, the voting authority Auth, and $n^{\mathsf{honest}}_{\mathsf{trustees}} = n_{\mathsf{trustees}} - t + 1$ out of n_{trustees} trustees are honest (w.l.o.g., we assume that the first $n^{\mathsf{honest}}_{\mathsf{trustees}}$ trustees are honest). Therefore, the honest part, which we denote by

$$\hat{\pi}_{\mathsf{H}} = \hat{\pi}_{\mathsf{J}}\|\hat{\pi}_{\mathsf{Auth}}\|\hat{\pi}_{\mathsf{B}}\|\hat{\pi}_{\mathsf{S}}\|\hat{\pi}_{\mathsf{T}_1}\|\ldots\|\hat{\pi}_{\mathsf{T}_{n^{\mathsf{honest}}_{\mathsf{trustees}}}}\|\hat{\pi}_{\mathsf{HV}},$$

consists of the honest programs $\hat{\pi}_{\mathsf{J}}, \hat{\pi}_{\mathsf{Auth}}, \hat{\pi}_{\mathsf{B}}, \hat{\pi}_{\mathsf{S}}, \hat{\pi}_{\mathsf{T}_k}, \hat{\pi}_{\mathsf{HV}}$ of the judge J, the voting authority Auth, the bulletin board B, the scheduler S, the trustees T_k, and the honest voters HV, respectively. By $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})$ we will denote the composition of all honest programs including the program of the voter under observation $\mathsf{V}_{\mathsf{obs}}$, i.e., $\hat{\pi}_{\mathsf{H}}(\mathsf{ch}) = \hat{\pi}_{\mathsf{H}}\|\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})$. All remaining parties are subsumed by the adversarial process π_{A}. This means that we can write $\hat{\pi}_{\mathsf{V}_{\mathsf{obs}}}(\mathsf{ch})\|\pi^*$ as $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}$. Recall that, by assumption, the adversary π_{A} is k-risk-avoiding.

In order to prove the result, we use a sequence of games. We fix $\mathsf{ch} \in \mathsf{C}$ and start with Game 0 which is simply the process $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})\|\pi_A$. Step by step, we transform Game 0 into Game 7 which is the composition $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})\|\pi_A$ for some process $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})$ and the same adversarial process π_A. Game 7 will be proven indistinguishable from Game 0 from the adversary's point of view, which means that

$$\left|\Pr[(\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})\|\pi_A) \mapsto 1] - \Pr[(\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})\|\pi_A) \mapsto 1]\right|$$

is negligible for a fixed $\mathsf{ch} \in \mathsf{C}$ (as a function of the security parameter). On the other hand, it will be straightforward to show that in Game 7 for

arbitrary $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C} \setminus \{\mathsf{abstain}\}$, the distance

$$\left|\Pr[(\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch}_0)\|\pi_A) \mapsto 1] - \Pr[(\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch}_1)\|\pi_A) \mapsto 1]\right|$$

is bounded by $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{res}})$ because $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch}_0)$ and $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch}_0)$ use the ideal voting protocol for $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ honest voters. Using the triangle inequality, we can therefore deduce that

$$|\Pr[(\hat{\pi}_{\mathsf{H}}(\mathsf{ch}_0)\|\pi_A) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}(\mathsf{ch}_1)\|\pi_A) \mapsto 1]|$$

is $\delta^{\mathsf{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}-k,\mu)}(f_{\mathsf{res}})$-bounded for all $\mathsf{ch}_0, \mathsf{ch}_1 \in \mathsf{C}$ (as a function of the security parameter).

Game 0. In what follows, we write $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ for $\hat{\pi}_{\mathsf{H}}(\mathsf{ch})$ and consider $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ as one atomic process (one program) and not as a composition of processes.[2] Now, Game 0 is the process $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})\|\pi_A$. △

In the next step, the scheduler S modifies the CRSs for the NIZKPs used by the *dishonest* trustees for proving knowledge and correctness of the key shares in such a way that he can later extract all of these secret key shares.

Game 1. For Game 1, we modify $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ in the following way to obtain $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^0_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ are identical.
Modified CRSs for $\pi_k^{\mathsf{KeyShareGen}}$. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_k^{\mathsf{KeyShareGen}}$ for NIZKPs of knowledge and correctness of the secret key shares sk_k corresponding to the published public key shares pk_k of the *dishonest* trustees, the modified scheduler (as a subprocess of $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$) uses (the first component of) an extractor algorithm (that exists by the computational knowledge extraction property) to generate $\sigma_k^{\mathsf{KeyShareGen}}$ (which is given to the adversary) along with a trapdoor $\tau_k^{\mathsf{KeyShareGen}}$. △

In the next step, the scheduler S modifies the CRSs for the NIZKPs used by the *honest* trustees for proving knowledge and correctness of the key shares in such a way that he can later simulate these NIZKPs without actually knowing the secret key shares.

Game 2. For Game 2, we modify $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ in the following way to obtain $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^2_{\mathsf{H}}(\mathsf{ch})$ are identical.
Modified CRSs for $\pi_k^{\mathsf{KeyShareGen}}$. Instead of using the (honest) setup algorithm to generate common reference strings $\sigma_k^{\mathsf{KeyShareGen}}$ for NIZKPs of knowledge and correctness of the secret key shares sk_k corresponding to the published public key shares pk_k of the *honest* trustees, the modified scheduler (as a subprocess of $\hat{\pi}^1_{\mathsf{H}}(\mathsf{ch})$) uses (the first component of) an extractor algorithm

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

(that exists by the computational knowledge extraction property) to generate $\sigma_k^{\mathsf{KeyShareGen}}$ (which is given to the adversary) along with a trapdoor $\tau_k^{\mathsf{KeyShareGen}}$. △

In the next step, the scheduler S modifies the CRSs for the NIZKPs used by the *dishonest* voters for proving knowledge and correctness of their ballots in such a way that he can later extract these choices.

Game 3. For Game 3, we modify $\hat{\pi}_{\mathsf{H}}^2(\mathsf{ch})$ in the following way to obtain $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}_{\mathsf{H}}^2(\mathsf{ch})$ and $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$ are identical.

Modified CRSs for π_i^{Enc}. Instead of using the (honest) setup algorithm to generate common reference strings σ_i^{Enc} for NIZKPs of knowledge and correctness of ch_i to be used by the *dishonest* voters V_i, the modified scheduler (as a subprocess of $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$) uses (the first component of) an extractor algorithm (that exists by the computational knowledge extraction property) to generate σ_i^{Enc} (which is given to the adversary) along with a trapdoor τ_i^{Enc}. △

In the next step, the scheduler S modifies the CRSs for the NIZKPs used by the *honest* voters for proving knowledge and correctness of their ballots in such a way that he can later simulate these NIZKPs without actually knowing the honest choices.

Game 4. For Game 4, we modify $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$ in the following way to obtain $\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$ and $\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})$ are identical.

Modified CRSs for π_i^{Enc}. Instead of using the (honest) setup algorithm to generate common reference strings σ_i^{Enc} for NIZKPs of knowledge and correctness of ch_i to be used by the *honest* voters V_i, the modified scheduler (as a subprocess of $\hat{\pi}_{\mathsf{H}}^3(\mathsf{ch})$) uses a simulator algorithm (that exists by the computational zero-knowledge property) to generate σ_i^{Enc} along with a trapdoor τ_i^{Enc}. △

In the next step, we exploit the fact that the adversary is k-risk-avoiding which means that the adversary does not manipulate or drop more than k honest votes unless the voting protocol aborts before the final result is published. For Ordinos, this leads to the situation that the adversary can only manipulate or drop honest votes before the tallying has started because the tallying procedure itself provides perfect verifiability.

Game 5. For Game 5, we modify $\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})$ in the following way to obtain $\hat{\pi}_{\mathsf{H}}^5(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}_{\mathsf{H}}^4(\mathsf{ch})$ and $\hat{\pi}_{\mathsf{H}}^5(\mathsf{ch})$ are identical.

The process $\hat{\pi}_{\mathsf{H}}^5(\mathsf{ch})$ halts if there are less than $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ballots submitted by the honest voters in the list of ballots being output by the authentication server AS. In this case, $\hat{\pi}_{\mathsf{H}}^5(\mathsf{ch})$ halts if it is triggered the first time after the ballots have been published. △

In the next step, we will exploit the fact that the MPC protocol in Ordinos provides privacy so that the honest part of the voting protocol can "internally" replace the real MPC protocol with the ideal one and simulate it towards the adversary. In order to do this, the ideal MPC protocol requires the secret key shares of the dishonest trustees which can be extracted from the dishonest trustees' NIZKPs with the trapdoors that have been introduced in Game 1. Furthermore, the ideal MPC protocol does not reveal the secret key shares of the honest trustees so that the simulator has to simulate their NIZKPs without knowing the secret key shares. This can be done with the trapdoors introduced in Game 2.

Game 6. For Game 6, we modify $\hat{\pi}^5_{\mathsf{H}}(\mathsf{ch})$ in the following way to obtain $\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^5_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})$ are identical.

Simulating key generation. Each time, an honest trustee T_k is triggered to generate its key shares $(\mathsf{pk}, \mathsf{sk})$, the simulator does the following. Instead of letting T_k run $\mathsf{KeyShareGen}$, the simulator invokes the ideal MPC protocol $\mathcal{I}_{\mathsf{MPC}}$ for generating the public/secret key shares $(\mathsf{pk}_k, \mathsf{sk}_k)$ and outputting the public key share pk (recall that the secret key share sk is not revealed by $\mathcal{I}_{\mathsf{MPC}}$). Then, the simulator uses the trapdoor $\tau_k^{\mathsf{KeyShareGen}}$ from Game 2 to generate a simulated NIZKP $\pi_k^{\mathsf{KeyShareGen}}$ (without actually knowing the secret key share sk_k).

Extracting dishonest key shares. After the authentication server has published the list of ballots, the simulator uses the trapdoors $\tau_k^{\mathsf{KeyShareGen}}$ from Game 1 to extract the secret key shares sk_k of the dishonest trustees T_k. The simulator forwards these secret key shares to the ideal MPC protocol $\mathcal{I}_{\mathsf{MPC}}$.

Secure tallying. The simulator simulates the computing phase of the real MPC protocol P_{MPC} with the ideal MPC protocol $\mathcal{I}_{\mathsf{MPC}}$. △

In the next and final step, the complete Ordinos protocol will be replaced by the ideal voting protocol. In order to do this, the ideal voting protocol requires the choices of the dishonest voters which can be extracted from the dishonest voters' NIZKPs with the trapdoors that have been introduced in Game 3. Furthermore, the ideal voting protocol does not reveal the choices of the honest voters so that the simulator has to simulate their NIZKPs without knowing the choices. This can be done with the trapdoors introduced in Game 4.

Game 7. For Game 7, we modify $\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})$ in the following way to obtain $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})$. Apart from the modifications below, $\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})$ and $\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})$ are identical.

Simulating ballot generation. Each time, an honest voter V_i (including the voter under observation) is triggered to pick ch_i according to μ and create her ballot b_i, the simulator does the following. The simulator sets $\mathsf{ch}_i = 0^{n_{\mathsf{cand}}}$ and encrypts it to obtain $\vec{\mathsf{c}}_i$. Then, the simulator uses the trapdoor τ_i^{Enc} from

Game 4 to generate a simulated NIZKP π_i^{Enc}.

Extracting dishonest choices. After the authentication server has published the list of ballots, the simulator uses the trapdoors τ_i^{Enc} from Game 1 to extract ch_i of each published ballot b_i that belongs to a dishonest voter V_i.

Secure tallying. The simulator replaces the ideal MPC protocol $\mathcal{I}_{\mathsf{MPC}}$ with the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')$ where the $n_{\mathsf{voters}}^{\mathsf{honest}} - k'$ is the number of ballots submitted in the list of ballots being output by the authentication server AS. Let $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')(\mathsf{ch})$ denote the protocol $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')$ in which the choice $n_{\mathsf{voters}}^{\mathsf{honest}} - k' + 1$ is set as ch. Now, the simulator first triggers $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')(\mathsf{ch})$ in order to (internally) determine the choices of the $n_{\mathsf{voters}}^{\mathsf{honest}} - k'$ honest votes. Then, the simulator triggers $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')(\mathsf{ch})$ to set the choices of the dishonest voters as extracted above. The output of $\mathcal{I}_{\mathsf{voting}}(f_{\mathsf{res}}, \mu, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}} - k')(\mathsf{ch})$ will be the output of the tallying phase. $\triangle$

Lemma 5. *For all $i \in \{0, 1, 2, 3\}$, Game i and Game $i+1$ are computationally indistinguishable, i.e., we have that*

$$\left|\Pr[(\hat{\pi}_{\mathsf{H}}^{i}(\mathsf{ch}) \| \pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}_{\mathsf{H}}^{i+1}(\mathsf{ch}) \| \pi_{\mathsf{A}}) \mapsto 1]\right|$$

is negligible (as a function of the security parameter).

Proof. This follows from the fact that $\pi^{\mathsf{KeyShareGen}}$ and π^{Enc} are proofs of knowledge (recall Section A.3 for details). □

Lemma 6. *For all $i \in \{1, 2, 3, 4\}$, the adversary is k-risk-avoiding in Game i (meaning that with overwhelming probability a run of the protocol does not stop before the final result is published and there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ choices by honest voters in the final result).*

Proof. If the adversary was not k-risk-avoiding in Game $i+1$ for some $i \in \{0, 1, 2, 3\}$, it would be possible to construct a ppt algorithm that distinguishes between Game i and Game $i+1$. This would contradict the previous Lemma. □

Lemma 7. *The probability that in a run of the process $\hat{\pi}_{\mathsf{H}}^{4}(\mathsf{ch}) \| \pi_{\mathsf{A}}$, there are at least $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters in the input of the tallying phase is overwhelming.*

Proof. Assume that less than $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ ciphertexts associated to the honest voters were in the input of the MPC protocol in Game 4. Due to the

correctness of the MPC protocol, the output of the MPC protocol would be different from any election result in which at most k honest choices have been manipulated. This contradicts the fact that the adversary is k-risk-avoiding in Game 4, as we have seen in the previous Lemma. □

Lemma 8. *Game* 4 *and Game* 5 *are computationally indistinguishable, i.e., we have that*

$$\left|\Pr[(\hat{\pi}^4_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}^5_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1]\right|$$

is negligible (as a function of the security parameter).

Proof. Recall that Game 5 halts if there are less than $n^{\mathsf{honest}}_{\mathsf{voters}} - k$ ballots submitted by the honest voters in the list of ballots being output by the authentication server AS. Since this probability is negligible, as we have seen in the previous Lemma, Game 4 and Game 5 are indistinguishable. □

Lemma 9. *Game* 5 *and Game* 6 *are computationally indistinguishable, i.e., we have that*

$$\left|\Pr[(\hat{\pi}^5_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1]\right|$$

is negligible (as a function of the security parameter).

Proof. This follows from the fact that $\pi^{\mathsf{KeyShareGen}}$ is a proof of knowledge and that the MPC protocol realizes the ideal MPC protocol. □

Lemma 10. *Game* 6 *and Game* 7 *are perfectly indistinguishable, i.e., we have that*

$$\left|\Pr[(\hat{\pi}^6_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1] - \Pr[(\hat{\pi}^7_{\mathsf{H}}(\mathsf{ch})\|\pi_{\mathsf{A}}) \mapsto 1]\right|$$

is negligible (as a function of the security parameter).

Proof. This follows from the fact that π^{Enc} is a proof of knowledge and that the remaining difference between Game 6 and Game 7 is purely syntactical. □

Bibliography

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 417–432, 2002.

[ACW13] Mathilde Arnaud, Véronique Cortier, and Cyrille Wiedling. Analysis of an electronic boardroom voting system. In *E-Voting and Identify - 4th International Conference, Vote-ID 2013, Guildford, UK, July 17-19, 2013. Proceedings*, pages 109–126, 2013.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348, 2008.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 104–115, 2001.

[AKBW14] Claudia Z. Acemyan, Philip T. Kortum, Michael D. Byrne, and Dan S. Wallach. Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for Helios, Prêt à Voter, and Scantegrity II. In *2014 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, EVT/WOTE '14*. USENIX Association, 2014.

[AL07] Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In

Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings, pages 137–156, 2007.

[And11] Andrew Appel. https://freedom-to-tinker.com/blog/appel/nj-election-cover/, September 13th 2011.

[Ano18] Anonymous. Ordinos: A Verifiable Tally-Hiding E-Voting System. Technical report, 2018. this report is available at http://51.68.178.50/ordinos-tr.pdf, the implementation and detailed benchmarks are available at http://51.68.178.50/ordinos-implementation.tar.gz.

[ASW00] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[BCG+15] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 499–516, 2015.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Report 2014/075, 2014. http://eprint.iacr.org/.

[Ben86] Josh D. Benaloh. Improving Privacy in Cryptographic Elections (technical report). Technical report, 1986.

[Ben87] Josh D. Benaloh. *Verifiable Secret-Ballot Elections*. PhD thesis, 1987.

[Ben06] Josh Benaloh. Simple verifiable elections. In Dan S. Wallach and Ronald L. Rivest, editors, *2006 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'06, Vancouver, BC, Canada, August 1, 2006*. USENIX Association, 2006.

[Bon12] Joseph Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 538–552, 2012.

[BPA12] Joseph Bonneau, Sören Preibusch, and Ross Anderson. A birthday present every eleven wallets? The security of customer-chosen banking PINs. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 25–40. 2012.

[CCFG16] Pyrros Chaidos, Véronique Cortier, Georg Fuchsbauer, and David Galindo. Beleniosrf: A non-interactive receipt-free electronic voting scheme. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1614–1625, 2016.

[CCM08] Michael R. Clarkson, Stephen Chong, and Andrew C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 354–368, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, pages 174–187, 1994.

[ÇDSS15] Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. Depth optimized efficient homomorphic sorting. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 61–80, 2015.

[CEK+15] Véronique Cortier, Fabienne Eigner, Steve Kremer, Matteo Maffei, and Cyrille Wiedling. Type-Based Verification of Electronic Voting Protocols. In *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, pages 303–323, 2015.

[CFH+07] Joseph A. Calandrino, Ariel J. Feldman, J. Alex Halderman, David Wagner, Harlan Yu, and William P. Zeller. Source Code

Review of the Diebold Voting System, 2007. Report commissioned as part of the California Secretary of State's Top-To-Bottom Review of California voting systems. http://www.eecs.berkeley.edu/~daw/papers/dieboldsrc-ttbr.pdf.

[CFP+10] Benoît Chevallier-Mames, Pierre-Alain Fouque, David Pointcheval, Julien Stern, and Jacques Traoré. On Some Incompatible Properties of Voting Schemes. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 191–199, 2010.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, pages 327–344, 2014.

[CGK+16a] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Sok: Verifiability notions for e-voting protocols. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 779–798, 2016.

[CGK+16b] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. Verifiability Notions for E-Voting Protocols. *IACR Cryptology ePrint Archive*, 2016:287, 2016.

[Cha81] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[CPP13] Edouard Cuvelier, Olivier Pereira, and Thomas Peters. Election Verifiability or Ballot Privacy: Do We Need to Choose? In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 481–498. Springer, 2013.

[CPRT18] Chris Culnane, Olivier Pereira, Kim Ramchen, and Vanessa Teague. Universally Verifiable MPC with Applications to IRV

Ballot Counting. *IACR Cryptology ePrint Archive*, 2018:246, 2018.

[CPST18] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. Practical strategy-resistant privacy-preserving elections. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pages 331–349, 2018.

[CRST15] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[CS11] Véronique Cortier and Ben Smyth. Attacking and fixing helios: An analysis of ballot secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF 2011, Cernay-la-Ville, France, 27-29 June, 2011*, pages 297–311, 2011.

[CS14] Chris Culnane and Steve A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 169–183, 2014.

[CW17] Véronique Cortier and Cyrille Wiedling. A formal analysis of the Norwegian E-voting protocol. *Journal of Computer Security*, 25(1):21–57, 2017.

[Dan09] Daniel Kane. http://ucsdnews.ucsd.edu/newsrel/science/08-09ElectronicVoting.asp, August 10th 2009.

[DGK+15] Anupam Datta, Deepak Garg, Dilsun Kirli Kaynar, Divya Sharma, and Arunesh Sinha. Program actions as actual causes: A building block for accountability. In *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 261–275, 2015.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, pages 28–42, 2006.

[DLM82] Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic Protocols. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC 1982)*, pages 383–400. ACM, 1982.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[Eps15] Jeremy Epstein. Weakness in Depth: A Voting Machine's Demise. *IEEE Security & Privacy*, 13(3):55–58, 2015.

[FJW11] Joan Feigenbaum, Aaron D. Jaggard, and Rebecca N. Wright. Towards a formal model of accountability. In *NSPW 2011*, pages 45–56, 2011.

[Gam84] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.

[GM15] Gregor Gößler and Daniel Le Métayer. A general framework for blaming in component-based systems. *Sci. Comput. Program.*, 113:223–235, 2015.

[GRCC15] Gurchetan S. Grewal, Mark Dermot Ryan, Liqun Chen, and Michael R. Clarkson. Du-Vote: Remote Electronic Voting with Untrusted Computers. In *CSF 2015*, pages 155–169, 2015.

[HRT10] James Heather, Peter Y. A. Ryan, and Vanessa Teague. Pretty Good Democracy for More Expressive Voting Schemes. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security*, volume 6345 of *Lecture Notes in Computer Science*, pages 405–423. Springer, 2010.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure Multi-Party Computation with Identifiable Abort. In *CRYPTO 2014*, pages 369–386, 2014.

[IT17] The Register Iain Thomson. It took DEF CON hackers minutes to pwn these US voting machines, 2017. https://www.theregister.co.uk/2017/07/29/us_voting_machines_hacking/.

[JCJ10] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections, New Directions in Electronic Voting*, pages 37–63, 2010.

[JRSW04] David Jefferson, Aviel D. Rubin, Barbara Simons, and David Wagner. Analyzing internet voting security. *Communications of the ACM, Special issue: The problems and potentials of voting systems*, 47(10):59–64, 2004.

[KKL+18] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the Security Properties of e-Voting Bulletin Boards. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, pages 505–523, 2018.

[KKO+11] Fatih Karayumak, Michaela Kauer, Maina M. Olembo, Tobias Volk, and Melanie Volkamer. User study of the improved Helios voting system interfaces. In *1st Workshop on Socio-Technical Aspects in Security and Trust, STAST 2011, Milan, Italy, September 8, 2011*, pages 37–44, 2011.

[KM17] Ralf Küsters and Johannes Müller. Cryptographic Security Analysis of E-voting Systems: Achievements, Misconceptions, and Limitations. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, pages 21–41, 2017.

[KMST16a] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 341–354, 2016.

[KMST16b] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. *IACR Cryptology ePrint Archive*, 2016:438, 2016.

[KMW12] Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In Xiaoyun Wang

and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[KOKV11] Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability Analysis of Helios - An Open Source Verifiable Remote Electronic Voting System. In *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11, San Francisco, CA, USA, August 8-9, 2011*, 2011.

[KRS10] Steve Kremer, Mark Ryan, and Ben Smyth. Election Verifiability in Electronic Voting Protocols. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *15th European Symposium on Research in Computer Security (ESORICS2010)*, volume 6345 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 2010.

[KT16] Ralf Küsters and Tomasz Truderung. Security Analysis of Re-Encryption RPC Mix Nets. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 227–242, 2016.

[KTV10a] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and Its Applications. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 122–136, 2010.

[KTV10b] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010. Full version available at http://eprint.iacr.org/2010/236/.

[KTV10c] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Proving Coercion-Resistance of Scantegrity II. In *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*, pages 281–295, 2010.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pages 538–553, 2011.

[KTV12a] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-Based Definition of Coercion-Resistance and its Applications. *Journal of Computer Security*, 20(6):709–764, 2012.

[KTV12b] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 395–409, 2012.

[KTV14] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 343–358, 2014.

[KZZ15a] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles. In *CCS 2015*, pages 352–363, 2015.

[KZZ15b] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-End Verifiable Elections in the Standard Model. In *EUROCRYPT 2015*, pages 468–498, 2015.

[KZZ17] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. An Efficient E2E Verifiable E-voting System without Setup Assumptions. *S&P 2017*, 15(3):14–23, 2017.

[Loe14] Loek Essers. http://www.pcworld.com/article/2159260/software-bug-disrupts-evote-count-in-belgian-election.html, May 26th 2014.

[LT13] Helger Lipmaa and Tomas Toft. Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part II*, pages 645–656, 2013.

[MN06] Tal Moran and Moni Naor. Receipt-Free Universally-Verifiable Voting with Everlasting Privacy. In *Advances in Cryptology -*

CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings, pages 373–392, 2006.

[NFSF14] Stephan Neumann, Christian Feier, Perihan Sahin, and Sebastian Fach. Pretty Understandable Democracy 2.0. Technical Report 2014/625, Cryptology ePrint Archive, 2014. http://eprint.iacr.org/2014/625.

[NORV14] Stephan Neumann, Maina M. Olembo, Karen Renaud, and Melanie Volkamer. Helios Verification: To Alleviate, or to Nominate: Is That the Question, or Shall we Have Both? In *Electronic Government and the Information Systems Perspective - Third International Conference, EGOVIS 2014, Munich, Germany, September 1-3, 2014. Proceedings*, pages 246–260, 2014.

[OBV13] Maina M. Olembo, Steffen Bartsch, and Melanie Volkamer. Mental Models of Verifiability in Voting. In *Vote-ID 2013*, pages 142–155, 2013.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238, 1999.

[PRST06] David C. Parkes, Michael O. Rabin, Stuart M. Shieber, and Christopher Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. In *Proceedings of the 8th International Conference on Electronic Commerce: The new e-commerce - Innovations for Conquering Current Barriers, Obstacles and Limitations to Conducting Successful Business on the Internet, 2006, Fredericton, New Brunswick, Canada, August 13-16, 2006*, pages 70–81, 2006.

[RBH+10] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. The Prêt à Voter Verifiable Election System. Technical report, 2010.

[RRI16] Peter Y. A. Ryan, Peter B. Rønne, and Vincenzo Iovino. Selene: Voting with transparent verifiability and coercion-mitigation. In *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ*

Church, Barbados, February 26, 2016, Revised Selected Papers, pages 176–192, 2016.

[Sca18] Enrico Scapin. *Implementation-level analysis of cryptographic protocols and their applications to e-voting systems*. PhD thesis, University of Stuttgart, Germany, 2018.

[SFC15] Ben Smyth, Steven Frink, and Michael R. Clarkson. Computational Election Verifiability: Definitions and an Analysis of Helios and JCJ. Number 2015/233, 2015.

[SFD+14] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 703–715, 2014.

[SK95] Kazue Sako and Joe Kilian. Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth. In *EUROCRYPT 1995*, pages 393–403, 1995.

[Smi07] Warren D. Smith. Three Voting Protocols: ThreeBallot, VAV, and Twin. In *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007*, 2007.

[SP15] Alan Szepieniec and Bart Preneel. New Techniques for Electronic Voting. *USENIX Journal of Election Technology and Systems (JETS)*, 3(2):46 – 69, 2015. Cryptology ePrint Archive, Report 2015/809.

[ST06] Berry Schoenmakers and Pim Tuyls. Efficient binary conversion for paillier encrypted values. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 522–537, 2006.

[SV15] Berry Schoenmakers and Meilof Veeningen. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, pages 3–22, 2015.

[Tod08] Todd R. Weiss. http://www.computerworld.com/s/article/9118204/Princeton_report_rips_N.J._e_voting_machines_as_easily_hackable_, October 27th 2008.

[WH] Janna-Lynn Weber and Urs Hengartner. Usability Study of the Open Audit Voting System Helios.

[WWH+10] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security analysis of India's electronic voting machines. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 1–14, 2010.

[WWIH12] Scott Wolchok, Eric Wustrow, Dawn Isabel, and J. Alex Halderman. Attacking the Washington, D.C. Internet Voting System. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 114–128, 2012.

Acknowledgements

This thesis would not have been written without the support and advice of several people whom I would like to thank.

First of all, I am indebted to my advisor Ralf Küsters for supporting me during the past four years and for giving me the opportunity to continue my studies as an external student. Ralf's passion for scientific excellence has greatly inspired me.

Besides my advisor, I thank Tomasz Truderung, especially for guiding me in the first year of my studies.

I am also grateful to all the other co-authors of those scientific publications on which this thesis is based. It has been a pleasure working with you.

Finally, I thank all research group mates in Trier and Stuttgart for the joyful time and thrilling darts competitions.