# sElect: A Lightweight Verifiable Remote Voting System

Ralf Küsters[1], Johannes Müller[1], and Enrico Scapin[1] Tomasz Truderung[2]

[1] University of Trier, {kuesters, muellerjoh, scapin}@uni-trier.de
[2] Polyas GmbH, ttruderung@gmail.com

**Abstract.** Modern remote electronic voting systems, such as the prominent Helios system, are designed to provide vote privacy and verifiability, where, roughly speaking, the latter means that voters can make sure that their votes were actually counted. In this paper, 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 and is designed to be particularly simple and lightweight in terms of its structure, the cryptography it uses, and the user experience. 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 prove that this system provides a good level of privacy, verifiability, and accountability for low-risk elections.

## 1 Introduction

E-voting systems are used in many countries for national or municipal elections as well as for elections within associations, societies, and companies. There are two main categories of such systems. In the first category, voters vote in polling stations using electronic voting machines, such as direct recording electronic voting systems or scanners. In the second category, called remote electronic voting, voters vote over the Internet using their own devices (e.g., desktop computers or smartphones). In addition, there are hybrid approaches, where voters, via an additional channel, e.g., mail, are provided with codes which they use to vote (code voting).

E-voting systems are complex hardware/software systems and as in all such systems programming errors can hardly be avoided. In addition, these systems might deliberately be tampered with when deployed in elections. This means that voters when using e-voting systems, in general, do not have any guarantee that their votes were actually counted and that the published result is correct, i.e., reflects the actual voters' choices. In fact, many problems have been reported (see, e.g., [1, 38]). Therefore, besides vote privacy, modern e-voting systems strive for what is called *verifiability*. This security property requires that voters are able to check the above, i.e., proper counting of their own votes and integrity of the overall result, even if voting machines/authorities are (partially) untrusted.

Several such e-voting systems have been proposed in the literature, including, for example, such prominent systems as Helios [4], Prêt à Voter [35], STAR-Vote [7], and Remotegrity [40]. Some systems, such as Civitas [14] and Scantegrity [12], are designed to, in addition, even achieve *coercion-resistance*, which requires that vote selling and voter coercion is prevented. Several of these systems have been used in binding elections (see, e.g., [5, 12, 17]). In this paper, we are interested in remote electronic voting, which is meant to enable the voter to vote via the Internet.

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 work, 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 work 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. More specifically, the main contributions of this paper are as follows.

**Contributions of this paper.** We present a new, particularly lightweight remote e-voting system, called *sElect* (secure/simple elections), which we implemented as a platform independent web application and for which we perform a detailed cryptographic security analysis w.r.t. privacy of votes as well as verifiability and accountability. The system combines several concepts, such as verification codes (see, e.g., [19]) and Chaumian mix nets [13], 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.

We briefly sketch the main characteristics of sElect, including several novel and unique features and concepts which should be beneficial also for other systems. Besides the technical account of sElect provided in the following sections, a general discussion on sElect, including its limitations, is also provided in Section 8.

*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 7). In a user study carried out in [3] 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, 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.

*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. As further discussed in Section 8, this simple procedure has several obvious benefits. For example, it reduces trust assumptions concerning the voter's computing platform (for fully automated verification the voter's computing platforms needs to be fully trusted). Also voter's 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. On the negative side, such codes open the way for voter coercion (see also Section 8).

*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 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 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 (but also some disadvantages), as discussed in Section 8.

*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 come with a rigorous cryptographic analysis, this is a valuable feature by itself.

Our cryptographic analysis, carried out in Sections 4, 5, and 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.[3] In contrast, sElect demonstrates that one can achieve (a certain level of) end-to-end verifiability, as

[3] As pointed out in [30], this combination does not guarantee end-to-end verifiability, though.

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.

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.

**Structure of the paper.** In Section 2, we describe sElect in detail on a conceptual level. Verifiability, accountability, and privacy of sElect are then analyzed in Sections 4, 5, and 6, respectively, based on the model of sElect provided in Section 3. Details of our implementation of sElect are presented in Section 7, with a detailed discussion of sElect and related work provided in Section 8. We conclude in Section 9. Full details and proofs can be found in the appendix of this paper; see [2] for the implementation and an online demo of sElect.

## 2 Description of sElect

In this section, we present the sElect voting system on the conceptual level. Its implementation is described in Section 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[4] 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. First, whenever we say that a party produces a signature on some message $m$, this implicitly means that the signature is in fact computed on the tuple $(elid, tag, m)$, where $elid$ 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 $v_1, \ldots, v_n$* and their *voter supporting devices (VSDs) $vsd_1, \ldots, vsd_n$*, an *authentication server $AS$*, *$m$ mix servers $M_1, \ldots, M_m$*, and a *voting authority $VA$*. 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

[4] 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.

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 (see Appendix A for a concrete instantiation).

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 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 VA.

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 $(sk_j, pk_j)$. 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 (or more generally, make a choice) $m_i$. In the latter case, the voter indicates her choice $m_i$ to the VSD. In addition, for verification purposes, a *verification code* $n_i$ is generated (see below), which the voter is supposed to write down/store. At the end of the election, the choice/verification code pairs of all voters who cast a vote are supposed to be published so that every voter can check that her choice/verification code pair appears in the final result, and hence, that her vote was actually counted. The verification code is a concatenation $n_i = n_i^{voter} \| n_i^{vsd}$ of two nonces. The first nonce, $n_i^{voter}$, 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 7, this nonce is a nine character string chosen by the voter). 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, $n_i^{vsd}$, is generated by the VSD itself, the *VSD generated nonce*. Now, when the verification code is determined, the VSD encrypts the voter's choice $m_i$ and the verification code $n_i$, i.e., the choice/verification code pair $\alpha_m^i = (m_i, n_i)$, under the last mix server's public key $pk_m$ using random coins $r_m^i$, resulting in the ciphertext $\alpha_{m-1}^i = \mathrm{Enc}_{pk_m}^{r_m^i}((m_i, n_i))$. Then, the VSD encrypts $\alpha_{m-1}^i$ under $pk_{m-1}$ using the random coins $r_{m-1}^i$, resulting in the ciphertext $\alpha_{m-2}^i = \mathrm{Enc}_{pk_{m-1}}^{r_{m-1}^i}(\alpha_{m-1}^i)$, and so on. In the last step, it obtains

$$\alpha_0^i = \mathrm{Enc}_{pk_1}^{r_1^i}(...(\mathrm{Enc}_{pk_m}^{r_m^i}(m_i, n_i))...).$$

The VSD submits $\alpha_0^i$ 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 $\alpha_0^i$; 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 $C_0$ containing all the cast valid ballots and the list *LN* containing the identifiers of all voters who cast a valid ballot. It is expected that the list *LN* is at least as long as $C_0$ (otherwise *AS* will be blamed for misbehavior).

**Mixing phase.** The list of ciphertexts $C_0$ posted by the authentication server is the input to the first mix server $M_1$, which processes $C_0$, as described below, and posts its signed output $C_1$ on the bulletin board. This output is the input to the next mix server $M_2$, and so on. We will denote the input to the $j$-th mix server by $C_{j-1}$ and its output by $C_j$. The output $C_m$ of the last mix server $M_m$ is the output of the mixing stage and, at the same time, the output of the election. It is supposed to contain the plaintexts $(m_1, n_1), ..., (m_n, n_n)$ (containing voters' choices 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 $C_{j-1}$ 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 $C_{j-1}$ under its private key $sk_j$, removes duplicates, and orders the result lexicographically. If an entry in $C_{j-1}$ 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 $C_j$.

**Verification phase.** After the final result $C_m$ 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 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 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 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 $n_i^{voter}$, appears next to her choice in the final result list. If this is the case, the voter would be convinced that her vote was counted (see also Section 4). A voter $v_i$ who decided to abstain from voting may check the list $LN$ to make sure that her name (identifier) is not listed there.[5] 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 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. A VSD $vsd_i$ checks whether the originally submitted plaintext $(m_i, n_i)$ appears in $C_m$. 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 a VSD $vsd_i$ which obtained such an acknowledgement and cannot find the plaintext $(m_i, n_i)$ in $C_m$. First, the VSD $vsd_i$ checks whether the ballot $\alpha_0^i$ is listed in the published result $C_0$ of the authentication server $AS$. If this is not the case, the VSD $vsd_i$ anonymously publishes the acknowledgement obtained from $AS$ on the bulletin board $B$ which proves that $AS$ misbehaved (recall that such an acknowledgement contains a

[5] Variants of the protocol are conceivable where a voter signs her ballot and the authentication 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.

signature of $AS$ on the ballot $\alpha_0^i$). Otherwise, i.e., if $\alpha_0^i$ is in $C_0$, the VSD checks whether $\alpha_1^i$ is listed in the published result $C_1$ of the first mix server $M_1$. If $C_1$ contains $\alpha_1^i$, the VSD $vsd_i$ checks whether $\alpha_2^i$ can be found in the published result $C_2$ of the second mix server $M_2$, and so on. As soon as the VSD $vsd_i$ gets to the first mix server $M_j$ which published a result $C_j$ that does not contain $\alpha_j^i$ (such a mix server has to exist), the VSD anonymously sends $(j, \alpha_j^i, r_j^i)$ to the bulletin board $B$. This triple demonstrates that $M_j$ misbehaved: the encryption of $\alpha_j^i$ under $pk_j$ with randomness $r_j^i$ yields $\alpha_{j-1}^i$, and hence, since $\alpha_{j-1}^i$ is in the input to $M_j$, $\alpha_j^i$ 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 $\alpha_j^i$ contains the voter's choice as a plaintext. In practice, the voter could, for example, use a trusted proxy server, the Tor network, or some anonymous e-mail service.

We say that a voter $v_i$ *accepts* the result of an election if neither the voter $v_i$ nor her VSD $vsd_i$ output a complaint. Otherwise, we say that $v_i$ *rejects* the result.

*Remark 1.* Note that the procedures for ballot casting and mixing are very simple. In particular, a mix server needs to carry out only $n$ decryptions. Using standard hybrid encryption based on RSA and AES, it amounts to $n$ RSA decryption steps ($n$ modular exponentiations) and $n$ 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.

## 3 Modeling

In this section, we formally model the sElect voting protocol, with full details provided in Appendix B. This model is the basis for our security analysis of sElect carried out in the following sections. The general computational model that we use follows the one in [28, 30]. This model introduces the notions of processes, protocols, instances, and properties, which we briefly recall before modeling sElect.

**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. In a run of a process, at any time 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. A 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 $\pi$ as $\pi = p_1 \parallel \cdots \parallel p_l$, where $p_1 \ldots, p_l$ are programs. If $\pi_1$ and $\pi_2$ are processes, then $\pi_1 \parallel \pi_2$ 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 $\pi$ where all programs are given the security parameter $\ell$ is denoted by $\pi^{(\ell)}$. The processes we consider are such that the length of a run is always polynomially bounded in $\ell$. Clearly, a run is uniquely determined by the random coins used by the programs in $\pi$.

**Protocol.** A *protocol P* specifies a set of agents (also called parties or protocol participants) and a set of channels these agents can communicate over. Moreover, $P$ specifies, for every agent $a$, a set $\Pi_a$ of all programs the agent $a$ may run and a program $\hat{\pi}_a \in \Pi_a$, *the honest program of a*, i.e., the program that $a$ runs if $a$ is honest, and hence, follows the protocol.

**Instance.** Let $P$ be a protocol with agents $a_1, \dots, a_n$. An *instance of P* is a process of the form $\pi = (\pi_{\mathsf{a}_1} \parallel \dots \parallel \pi_{\mathsf{a}_n})$ with $\pi_{\mathsf{a}_i} \in \Pi_{a_i}$. An agent $\mathsf{a}_i$ is called *honest* in the instance $\pi$, if $\pi_{\mathsf{a}_i} = \hat{\pi}_{\mathsf{a}_i}$. A *run of P* (with security parameter $\ell$) is a run of some instance of $P$ (with security parameter $\ell$); we consider the instance to be part of the description of the run. An agent $\mathsf{a}_i$ is honest in a run $r$, if $r$ is a run of an instance of $P$ with honest $\mathsf{a}_i$.

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

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

**Modeling of sElect.** The sElect system can be modeled in a straightforward way as a protocol $P_{sElect} = P_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{vsd}, p^{verif}_{abst})$ in the above sense, as detailed next. By $n$ we denote the number of voters and their voter supporting devices, and by $m$ the number of mix servers. By $\mu$ we denote a probability distribution on the set of candidates/choices, including abstention. An honest voter makes her choice according to this distribution.[6] This choice is provided to her VSD and is called the *actual choice* of the voter. By $p^{verif}_{voter} \in [0,1]$ 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. By $p^{verif}_{vsd} \in [0,1]$ we denote the probability that an honest VSD of a voter who does not abstain from voting is triggered to verify the result. By $p^{verif}_{abst} \in [0,1]$ we denote the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $LN$ output by the authentication server. Note that the set of valid choices (candidates) is implicitly given by $\mu$. We assume that the choices are represented by messages of the same length.

The set of agents of $P_{sElect}$ consists of all agents described in Section 2, i.e., the bulletin board $B$, $n$ voters $v_1, \dots, v_n$, $n$ VSDs $vsd_1, \dots, vsd_n$, the authentication servers $AS$, $m$ mix servers $M_1, \dots, M_m$, and in addition, a scheduler $S$. The latter party will play the role of the voting authority $VA$ 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

[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.

these channels, but dishonest agents might. The honest programs $\hat{\pi}_{\mathsf{a}}$ of honest agents are defined in the obvious way according to the description of the agents in Section 2. We assume that the scheduler and the bulletin board are honest. Technically, this means that the set of programs $\Pi_{\mathsf{a}}$ of each of these agents contains only one program, namely, the honest one. All other agents can possibly be dishonest. For these agents, the sets $\Pi_{\mathsf{a}}$ of their programs contain all 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 (see also Section 8).

## 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 a generic definition of end-to-end verifiability presented in [28]. Importantly, verifiability is ensured without having to trust any of the VSDs 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 8).

For brevity of presentation, we state a simple domain specific instantiation of the general end-to-end verifiability definition in [28]. This definition is centered around the *goal* that a system is supposed to achieve. Informally speaking, according to [28], end-to-end verifiability postulates that if some parties (such as voting authorities) deviate from the protocol in a "serious" way, then this deviation is noticed by honest participants (such as voters or external observers) with high probability. Misbehavior is considered serious if the *goal* of the protocol (which may be different for different domains) is violated.

We start by introducing the goal for voting protocols.

### 4.1 Goal for Voting Protocols

In what follows, we assume that the result of the election is simply a multiset of choices, as is the case for sElect. This multiset contains also vote abstention. Therefore, the number of elements in this multiset always equals the total number of voters $n$ in our modeling of sElect (see Section 3).

Formally, for a (voting) protocol $P$, a *goal* is set of runs of $P$. The following definition, with further explanation provided below, precisely defines the goal $\gamma_k$ for voting. First, recall from Section 3 that an honest voter[7] $v_i$ first chooses a candidate $m_i$ (the *actual choice* of $v_i$) and then inputs the candidate to her VSD. The VSD is supposed to create and cast a ballot containing this choice.

[7] Also recall from Section 3 the definition of honest agents in runs of protocols and instances of protocols.

**Definition 1 (Goal $\gamma_k$).** *Let r be a run of some instance of a protocol with $n_h$ honest voters and $n_d = n - n_h$ dishonest voters. Let $C_h = \{c_1, \ldots, c_{n_h}\}$ be the multiset of actual choices of the honest voters in this run, as described above (recall that the choices also contain abstentions). We say that $\gamma_k$* is satisfied in *r (or $r \in \gamma_k$), if the published result of the election is a multiset $\{\tilde{c}_1, \ldots \tilde{c}_n\}$ which contains at least $n_h - k$ elements of the multiset $C_h$; if no election result is published in r, then $\gamma_k$ is not satisfied in r.*

The above definition says that in a run *r* the goal $\gamma_k$ is satisfied if in the published result all votes of honest voters are included, except for at most *k* votes, and for every dishonest voter at most one choice is included. In particular, for $k = 0$, $\gamma_k$ guarantees that all votes of the honest voters are counted and at most one vote of every dishonest voter. We refer the reader to [29] for more discussion on $\gamma_k$.

### 4.2 Definition of Verifiability

As mentioned at the end of Section 2, every voter either *accepts* or *rejects* a protocol run, where a voter accepts if neither the voter nor her VSD outputs a complaint according to the description in Section 2; otherwise the voter rejects.

Now, the intuition behind the notion of verifiability is that, whenever the goal of the protocol is violated, then with high probability some voters will notice it and reject the run. Conversely, the probability that the goal is violated and yet all the voters accept should be small. In the following definition, we bound this probability by a constant $\delta$.

**Definition 2 (Verifiability).** *An e-voting protocol P provides $\delta$-verifiability with tolerance k if, for every instance $\pi$ of P, the probability that in a run r of $\pi$*

*(a) the goal $\gamma_k$ is violated in r (that is $r \notin \gamma_k$), and yet*

*(b) all voters accept r*

*is $\delta$-bounded (i.e., bounded by $\delta$ plus some negligible function, as defined in Section 3).*

### 4.3 Analysis

In this section, we state the level of verifiability being offered by sElect according to Definition 2. As already pointed out above, to achieve this verifiability level we only have to assume that a voter has access to the final result. We do not need any other trust assumptions. In particular, the mix servers, the authentication server, and all VSDs can be dishonest.

The verifiability level of sElect depends on whether or not *clashes* occur, i.e., whether two or more honest voters chose the same nonce. We denote the probability of having at least one clash by $p_{clash}$ and define $p_{noclash} = 1 - p_{clash}$. 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 made the same choice 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* [31].

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

**Theorem 1 (Verifiability).** *The sElect protocol $P_{sElect}(n, m, \mu, p_{voter}^{verif}, p_{vsd}^{verif}, p_{abst}^{verif})$ provides $\delta^k(p_{voter}^{verif}, p_{abst}^{verif})$-verifiability w.r.t. the goal $\gamma_k$, where*

$$\delta^k(p_{voter}^{verif}, p_{abst}^{verif}) = p_{noclash} \cdot \left(1 - \min\left(p_{voter}^{verif}, p_{abst}^{verif}\right)\right)^{k+1} + p_{clash}.$$

The theorem says that the probability that more than $k$ votes of honest voters are manipulated, i.e., changed, dropped, or added for honest voters who abstained (ballot stuffing), but still no voter complaints, and hence, rejects the run, is bounded by $\delta^k(p_{voter}^{verif}, p_{abst}^{verif})$.

The formal proof of Theorem 1 is provided in Appendix C. The intuition behind the definition of $\delta^k(p_{voter}^{verif}, p_{abst}^{verif})$ is simple. 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. So, in order to manipulate more than $k$ honest votes, and hence, violate $\gamma_k$, at least $k+1$ honest voters 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, $\delta^k(p_{voter}^{verif}, p_{abst}^{verif})$ 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., [11, 10]) 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.

## 5 Accountability

While verifiability requires that manipulation can be detected, roughly speaking, accountability in addition requires that misbehaving parties can be blamed.

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 trusting 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. The judge cannot tell whether a voter makes false claims or some servers actually misbehaved.

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 proposed in [28], which we instantiate for sElect. The detailed formal accountability result and full proofs are given in Appendix D. Here, we only describe the most important aspects of this result.

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.

## 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.

We prove our privacy result under the assumption that one of the mix servers is honest. Clearly, if all the mix servers were dishonest, privacy could not 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 of the voter in plaintext. In our formal analysis of privacy below, we therefore consider the voter and the VSD to be one entity. In addition, we assume that honest voters (VSDs) can successfully cast their ballots, i.e., when a voter casts a ballot, then the authentication server returns a valid acknowledgment. As discussed in Sections 2, not obtaining such acknowledgments is a general problem in remote voting systems as servers could always ignore messages from voters; voters can complain in such cases.

More specifically, we prove the privacy result for the modified protocol $P^j_{sElect} = P^j_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$ which coincides with $P_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{vsd}, p^{verif}_{abst})$, except for the following three changes. First, as mentioned before, we now consider the voter and the VSD to be one agent. We therefore also consider only one probability of performing the verification procedure, which we denote by $p^{verif}_{voter}$. Second, the set $\Pi_{M_j}$ of programs of the $j$-th mix server $M_j$ contains only the honest program of $M_j$, modeling that in all instances of this protocol $M_j$ is honest. Third, as discussed above, the set of programs $\Pi_{AS}$ of the authentication server $AS$ consists only of those programs that respond with valid acknowledgments when honest VSDs cast their ballots; we stress that otherwise the programs of $AS$ can perform arbitrary (dishonest) actions, e.g., drop the voter's ballot nevertheless.

Roughly speaking, our privacy result says that no adversary is able to distinguish whether some voter (called the *voter under observation*) voted for candidate $c$ or $c'$, where the voter under observation runs her honest program.

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

### 6.1 Semi-Honest Adversaries

An adversary who controls the first mix 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 as formalized below.

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 (see Section 4). Hence, in the above attack where $k$ is big, the probability of the adversary to be caught would be very close to 1 (see also the discussion in Section 6.3). 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 could consider dropping some small number of votes, for which the risk of being caught is not that huge, in order to weaken privacy to some degree. To analyze this trade-off, we now introduce the notion of $k$-semi-honest adversaries. Intuitively, a $k$-semi-honest adversary manipulates, i.e., drops or changes, at most $k$ entries of honest voters in a protocol run; apart from this restriction, such an adversary can perform any adversarial action. Jumping ahead, we show in Section 6.3 that for sElect $k$ must be quite high to weaken privacy even by a small amount. So altogether, dropping/changing votes of honest voters in order to break privacy is not a reasonable thing to do for an adversary who avoids being caught cheating.

We now formulate $k$-semi-honest adversaries for the protocol $P^j_{sElect}$ (see above). However, the general concept should be applicable to other protocols as well.

To define $k$-semi-honest adversaries, we consider the set $\gamma'_k$ of runs of $P^j_{sElect}$ which is defined similarly to $\gamma_k$ (see Section 4.1) but is concerned only with honest voters who actually cast a ballot. Then, for a $k$-semi-honest adversary we require that running this adversary with $P^j_{sElect}$ yields a run in $\gamma'_k$.

Formally, $\gamma'_k$ is defined as follows. Let $r$ be a run of some instance of $P^j_{sElect}$ and let $C'_h = \{(c_1, n_1), \ldots, (c_{l'}, n_{l'})\}$ be the multiset of vote-nonce pairs in the ballots successfully cast by honest voters in $r$, where $c_i$ is the actual choice of such an honest voter and $n_i$ is the verification code.[8] We say that $\gamma'_k$ *is satisfied in* $r$ (or $r \in \gamma'_k$) if the list of published vote-nonce pairs in $r$ (the output $C_m$ of the last mix server), as a multiset, contains at least $l' - k$ elements of the multiset $C'_h$, where $l'$ is the number of elements of $C'_h$; if no election result is published in $r$, and hence, no vote-nonce pairs, then $\gamma'_k$ is not satisfied in $r$.

[8] Recall the definition of actual choices of honest voters from Section 4.1. Also note that with overwhelming probability, the multiset $C'_h$ does not contain duplicates as the verification codes will be different with overwhelming probability.

**Definition 3** (**$k$-semi-honest adversaries).** *We say that an adversary is* $k$-semi-honest in a run *$r$ (of $P^j_{sElect}$), if the property $\gamma'_k$ is satisfied in this run.*[9] *An adversary (of an instance $\pi$ of $P^j_{sElect}$) is* $k$-semi-honest *if it is $k$-semi-honest with overwhelming probability (over the set of runs of $\pi$).*

The following result shows that, under any circumstances, not being $k$-semi-honest involves a high and predictable risk of being blamed (which means that some VSD outputs valid evidence for blaming the adversary). More specifically, it demonstrates that whenever the adversary is not $k$-semi-honest, the probability that he will be caught is at least $1-(1-p^{verif}_{voter})^{k+1}$.

To state this result, we use the following notation. Recall that a run $r$ of an instance $\pi$ of $P^j_{sElect}$ is determined by the random coins the dishonest parties in $\pi$ (the adversary) and the honest parties use. Let $\omega$ denote the random coins used in $r$. We can represent $\omega$ as $\langle \omega', \omega_v \rangle$ where $\omega_v$ are the random coins used by the honest voters to determine whether they check their verification codes (see Section 2, the verification phase) and $\omega'$ contains the remaining part of $\omega$. Note that $\omega'$ completely determines the run of the protocol up to the verification phase. In particular, $\omega'$ determines the output of the last mix server and it determines whether the goal $\gamma'_k$ is satisfied or not ($\gamma'_k$ does not depend on $\omega_v$). Let us interpret $\omega'$ as an event, i.e., a set of runs of $P^j_{sElect}$ where the random coins are partially fixed to be $\omega'$ and $\omega_v$ is arbitrary. Then there are two possible cases. Either the adversary is $k$-semi-honest in all runs of $\omega'$, and hence, $\omega' \subseteq \gamma'_k$, or the adversary is not $k$-semi-honest in all runs of $\omega'$, i.e., $\omega' \cap \gamma'_k = \emptyset$.

**Lemma 1.** *Let $\pi$ be an instance of $P^j_{sElect}$. For all (but negligibly many) $\omega'$ such that the adversary in $\pi$ is not k-semi-honest in $\omega'$, we have that*

$$\mathsf{Pr}\left[IB \mid \omega'\right] \geq 1-(1-p^{verif}_{voter})^{k+1},$$

*where IB denotes the event that individual blame is assigned, i.e., one of the voters (VSDs) outputs valid evidence for a dishonest server.*

This lemma, with the proof provided in Appendix E, can be interpreted as follows. Whenever an adversary (controlling the dishonest servers) has produced an election output where he dropped/manipulated more than $k$ vote-nonce pairs of honest voters, then he knows that he, i.e., some of the dishonest servers, will be caught and blamed (i.e., evidence for blaming the dishonest server will be produced) with a probability of at least $1-(1-p^{verif}_{voter})^{k+1}$. This risk is enormous even for quite modest $k$ and realistic probabilities $p^{verif}_{voter}$ (see also below). So, unless an adversary does not care being caught, not being $k$-semi honest is not reasonable. As argued below, increasing $k$ does not buy the adversary much in weakening privacy, but dramatically increases his risk of being caught.

[9] Recall that a run is a run of some instance of $P^j_{sElect}$ and that the adversary consists of the dishonest agents in this instance.

### 6.2 Definition of Privacy

In our analysis of the sElect system, we use the definition of privacy for e-voting protocols proposed in [30], but where adversaries are restricted to be *k*-semi-honest. As opposed to simulation-based definitions (see, for instance, [24]) and related game-based definitions (e.g., [9]) which take a binary view on privacy and reject protocols that do not provide privacy on the ideal level, the definition of [30] allows one to *measure* the level of privacy a protocol provides. 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.

As briefly mentioned above, following [30], we formalize privacy of an e-voting protocol as the inability of an adversary to distinguish whether some voter $v$ (the voter under observation), who runs her honest program, voted for a candidate $c$ or $c'$.

To define this notion formally, we first introduce the following notation. Let $P$ be an (e-voting) protocol in the sense of Section 3 with voters, authorities, etc. Given a voter $v$ and a choice $c$, the protocol $P$ induces a set of instances of the form $(\hat{\pi}_v(c) \parallel \pi^*)$ where $\hat{\pi}_v(c)$ is the honest program of the voter $v$ under observation which takes $c$ as the candidate for whom $v$ votes and where $\pi^*$ is the composition of programs of the remaining parties. In the case of sElect, $\pi^*$ would include the scheduler, the bulletin board, all other voters, the authentication server, and all mix servers.

Let $\mathsf{Pr}[(\hat{\pi}_v(c) \parallel \pi^*)^{(\ell)} \mapsto 1]$ denote the probability that the adversary, i.e., the dishonest agents in $\pi^*$, writes the output 1 on some dedicated channel in a run of $\hat{\pi}_v(c) \parallel \pi^*$ with security parameter $\ell$ and some candidate $c$, where the probability is taken over the random coins used by the agents in $\hat{\pi}_v(p) \parallel \pi^*$.

Now, we define privacy with respect to *k*-semi-honest adversaries.

**Definition 4.** *Let $P$ be a protocol with a voter under observation $v$ and let $\delta \in [0,1]$. We say that $P$* with $l$ honest voters achieves $\delta$-privacy w.r.t. $k$-semi-honest adversaries*, if*

$$\left|\mathsf{Pr}[(\hat{\pi}_v(c) \parallel \pi^*)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_v(c') \parallel \pi^*)^{(\ell)} \mapsto 1]\right| \tag{1}$$

*is $\delta$-bounded as a function of the security parameter $\ell$, for all candidates $c,c'$ ($c,c' \neq$ $\mathsf{abstain}$) and all programs $\pi^*$ of the remaining parties such that at least $l$ voters are honest in $\pi^*$ (excluding the voter under observation $v$) and such that the adversary (the dishonest parties in $\pi^*$) is $k$-semi-honest.*

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

Since $\delta$ typically depends on the number $l$ of honest voters, privacy is formulated w.r.t. this number. Note that a smaller $\delta$ means a higher level of privacy. However, even for the ideal e-voting protocol, where voters privately enter their votes and the adversary sees only the election outcome, $\delta$ cannot be 0: there is, for example, 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. We denote the level of privacy of the ideal protocol by $\delta^{id}_{l,\mu}$, where $l$ is the number of honest voters and $\mu$ the probability distribution used by the honest voters to determine their choices (see Appendix F for an explanation of how $\delta^{id}_{l,\mu}$ is calculated).

### 6.3 Analysis

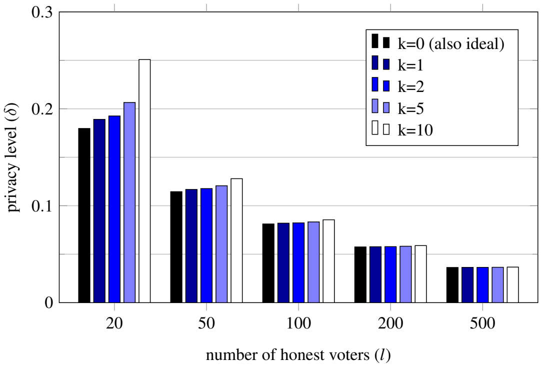


**Fig. 1.** Privacy level $\delta_{l-k}$ for sElect with $k$-semi-honest adversary, for different number of honest voters $l$ and different $k$. The honest voters vote for two candidates, with probabilities 0.4 and 0.6. Note that the case $k = 0$ also equals the ideal case.

We now prove that sElect provides a high level of privacy w.r.t. $k$-semi-honest adversaries and in case (at least) one mix server is honest. Where "high level of privacy" means that $\delta$-privacy is provided for a $\delta$ that is very close to the ideal one mentioned above.

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 $\mu$ the probability of abstention is zero.

We have the following formal privacy result for sElect. The proof is provided in Appendix F, where we reduce the privacy game for sElect with $l$ honest voters, as specified in Definition 4, to the privacy game for the ideal voting system with $l-k$ voters, using a sequence of games.

**Theorem 2 (Privacy).** *The protocol $P^{j}_{sElect}(n,m,\mu,p^{verif}_{voter},\, p^{verif}_{abst})$, with $l$ honest voters achieves $\delta^{id}_{l-k,\mu}$ privacy w.r.t. $k$-semi-honest adversaries, with $\delta^{id}_{\nu,\mu}$ as defined in Section 6.2.*

In Figure 1, we present some selected values of $\delta^{id}_{l-k,\mu}$ which, by the above theorem, express the privacy level of sElect when $k$-semi-honest adversaries are considered. As can be seen from Figure 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, even if we take $p = 0.2$ for the verification rate (which is much less than the verification rates obtained in our mock elections, see Section 7), the risk is 36% for $k = 2$, 67% for $k = 5$, and 89% for $k = 10$; with $p = 0.5$ similar to our mock elections, we obtain 75% for $k = 2$, 97% for $k = 5$, and $\approx$ 100% for $k = 10$. This means that unless adversaries do not care being caught at all, privacy cannot be broken.

## 7 Implementation of sElect

In this section, we shortly describe our prototypical implementation of sElect. A more detailed overview is given in Appendix A. 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 (a full-fledged usability study is out of the scope of this paper and left for future work).

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 8). 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 2 within the browser and without further interaction with the voting booth server.[10]

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; see Appendix A for the details of authentication. 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

[10] 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.

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 2 (see also Section 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 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 4, 5, and 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.

## 8 Related Work and Discussion

In what follows, we first briefly mention further related work and then discuss features and limitations of sElect.

### 8.1 Related work

The basic idea of combining the choice of a voter with an individual verification code has already been mentioned in an educational voting protocol by Schneier [37].

The F2FV boardroom voting protocol [6] is based on the concept of verification codes too. In that protocol, it is assumed that all voters are located in the same room and use their devices in order to submit their vote-nonce pairs as plaintexts to the bulletin board. As pointed out in [6], F2FV is mainly concerned with verifiability, but not with privacy.

Several remote e-voting protocols have been proposed in the literature (see also the introduction), with Helios [4] being the most prominent one.

In Helios, a voter, using her browser, submits a ballot (along with specific zero-knowledge proofs) to a bulletin board. Afterwards, in the tallying phase, the ballots on the bulletin board are tallied in a universally verifiable way, using homomorphic tallying and verifiable distributed decryption. Helios uses so-called Benaloh challenges to ensure that browsers encrypt the actual voters' choices (cast-as-intended). For this purpose, the browser, before submitting the ballot, asks whether the voter wants to audit or cast the ballot. In the former case, the browser reveals the randomness used to encrypt the voter's choice. After that, the voter should copy/paste this information to another (then trusted) device to check that the ballot actually contains the voter's choice. The voter is also supposed to check that her ballot appears on the bulletin board, which together with the homomorphic tallying and verifiable distributed decryption then implies that the voter's vote is counted.

Helios-C [16] is a modification of Helios where a registration authority creates public/private key pairs for all voters. Voters sign their ballots in order to prevent ballot stuffing even if the bulletin board is dishonest.

### 8.2 Discussion

We now provide a more detailed discussion of the main features of sElect, which were already mentioned in the introduction, including limitations of the systems.

*Fully automated verification.* Fully automated verification, put forward in this paper, is a main and unique feature of sElect, which would also be very useful for other systems, such as Helios. This kind of verification is performed without any interaction required from the voter, and hence, is completely transparent to the user. In particular, the voter does not have to perform any cumbersome or complex task, which thus eases the voter's experience. This, and the fact that fully automated verification is triggered when the voter visits the voting booth again (to later look up the election result on the bulletin board), should also help to improve verification rates, as hinted at by our two small mock elections. Moreover, this kind of verification importantly also provides a high-level of accountability, as we proved (see Section 5).

Obviously, for fully automated verification we need to assume that (most of) the VSDs can be trusted. Recall from Section 7 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.

As mentioned, 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.[11] Also, as mentioned in Section 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 below and our analysis in Section 4).

It seems that even for high-stake and high-risk elections some kind of fully automated verification might be better than completely relying on actions performed by the voter,

[11] 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., [20]).

as is the case for all other remote e-voting systems. So, other systems should profit from this approach as well.[12]

*Voter-based verification (human verifiability).* The level of verifiability provided by voter-based verification (manual checking of voter-generated verification codes) has been analyzed in detail in Section 4.

On the positive side, voter-based verification provides a quite good level of verifiability, with the main problem being clashes (as discussed in Section 4.3). 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. Moreover, trust on the voter's computing platform (hardware, operating system, browser), which is the other part of her VSD, is reduced significantly with voter-based verification: in order to hide manipulations, the voter's computing platform would have to present a fake election outcome to the voter. As mentioned before, our underlying assumption is that (for low-risk elections) such targeted attacks are not performed on the voter's computing platform. (Of course, voters also have the option to look up the election result using a different device.)

Voter-based verification is also very easy for the voter to carry out and the voter easily grasps its purpose. In particular, she can be convinced that her vote was actually counted without understanding details about the system, e.g., the meaning and workings of universally verifiable mix nets or verifiable distributed decryption. 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.[13]

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

[12] For high-risk elections one might have to take extra precautions for secretly storing the voter's receipt in the voter's browser or on her computer.

[13] 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 by several usability studies (see,

e.g., [3, 22, 23, 33, 34, 39]). 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.

is published, and hence, once the result is published, a coercer can see how the voter voted.[14]

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.

*Simple cryptography and design.* Unlike other modern remote e-voting systems, sElect employs only the most basic and standard cryptographic operations, namely, public key encryption and digital signatures, while all other verifiable remote e-voting systems use more sophisticated cryptographic operations, such as zero-knowledge proofs, verifiable distributed decryption, universally verifiable mix nets, etc. The overall design and structure of sElect is simple as well. As already mentioned in the introduction, the motivation for our design choices were twofold: Firstly, we wanted to investigate what level of security (privacy, verifiability, and accountability) can be achieved with only the most basic cryptographic primitives and a simple and user-friendly design. Secondly, using only the most basic cryptographic primitives has several advantages: i) 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 [3]. ii) The implementation of sElect is also quite efficient (see Section 2). iii) 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.[15] We note that in [25, 26] very complex non-remote voting systems were recently proposed to obtain security without such assumptions. iv) Post-quantum cryptography could easily be used with sElect, because one could employ appropriate public key encryption schemes and signature schemes. v) 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

[14] In very recent work, a mitigation for this problem has been considered [36], but this approach assumes, among others, a public-key infrastructure for all voters.

[15] 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.

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).

Since sElect employs Chaumian mix nets, 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.

*Bulletin board.* We finally note that in our security analysis of sElect and also in its implementation, we consider an (honest) bulletin board. This has been done for simplicity and is quite common; for example, the same is done in Helios. The key property required is that every party has access to the bulletin board and that it provides the same view to everybody. This can be achieved in different ways, e.g., by distributed implementations and/or observers comparing the (signed) content they obtained from bulletin boards (see, e.g., [18]); such approaches are orthogonal to the rest of the system, though.

## 9 Conclusion

We proposed a new practical voting system, sElect, which is intended for low-risk elections. It provides a number of new features and compared to existing modern remote voting systems is designed to be particularly simple and lightweight in terms of its structure, the cryptography it uses, and the user experience.

One of the unique features of sElect is its fully automated verification procedure (VSD-based verification), which allows for seamless verification without voter interaction and provides a good level of accountability, under the assumption that the voter's VSD is honest. Moreover, fully automated verification is linked with the act of looking up the election outcome, which should further increase verification rates.

sElect also supports voter-based verification which provides a very simple and easy to grasp manual verification mechanism (human verifiability) and which mitigates the trust in the VSD.

We provided a detailed cryptographic analysis of the level of verifiability, accountability, and privacy sElect offers. Along the way, we introduced the new concept of $k$-semi honest adversaries and showed that the level of privacy sElect provides is close to ideal for the class of $k$-semi-honest adversaries. We also show that while increasing $k$ (i.e., the number of dropped/manipulated votes) buys almost nothing in terms of breaking privacy, the risk of being caught increases drastically, and hence, unless an adversary does not care being caught at all, privacy cannot be broken. Our security analysis of sElect is a valuable feature by itself, as rigorous cryptographic analysis of practical systems is rare, and it moreover shows that even with very simple cryptographic means, one can achieve a relatively good level of security.

Altogether, sElect provides a new balance between simplicity, convenience, and security. It is an interesting new option for low-risk remote electronic voting. Some of its new features can probably also be integrated into other systems or might inspire new designs. While we carried out two small mock elections with sElect, mainly to get first feedback on VSD-based verification rates, relevant future work includes to perform a systematic and broad usability study and to try out sElect in bigger and real-world elections.

**Acknowledgements.** This work was partially supported by *Deutsche Forschungsgemeinschaft* (DFG) under Grant KU 1434/6-3 within the priority programme 1496 “Reliably Secure Software Systems – RS$^3$”.

## References

1. http://www.computerworld.com/s/article/9233058/Election_watchdogs_keep_wary_eye_on_paperless_e_voting_systems, October 30th 2012.
2. Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: Implementation, 2015. Source code available at https://github.com/escapin/sElect, online demo at https://select.uni-trier.de.
3. C. Acemyan, P. Kortum, M. Byrne, D. Wallach. Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for Helios, Prêt à Voter, and Scantegrity II. *USENIX Journal of Election Technology and Systems (JETS)*, 2014.
4. Ben Adida. Helios: Web-based Open-Audit Voting. In *USENIX 2008*, pages 335–348. USENIX Association, 2008.
5. Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jaques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *(EVT 2009)*, 2009.
6. Mathilde Arnaud, Véronique Cortier, and Cyrille Wiedling. Analysis of an Electronic Boardroom Voting System. In *VOTE-ID*, volume 7985 of *LNCS*, pages 109–126. Springer, 2013.
7. S. Bell, J. Benaloh, M. Byrne, D. DeBeauvoir, B. Eakin, G. Fischer, Ph. Kortum, N. McBurnett, J. Montoya, M. Parker, O. Pereira, Ph. Stark, D. Wallach, and M. Winn. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. *USENIX Journal of Election Technology and Systems (JETS)*, 1:18–37, August 2013.
8. D. Bernhard, V. Cortier, D. Galindo, O. Pereira, and B. Warinschi. A comprehensive analysis of game-based ballot privacy definitions. Technical Report 2015/255, Cryptology ePrint Archive, 2015. To appear in S&P 2015.
9. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, 2012.
10. J. Bonneau, and S. Preibusch, and R. Anderson. A birthday present every eleven wallets? The security of customer-chosen banking PINs, In *Financial Cryptography and Data Security*, volume 7397 of *LNCS*, pages 25–40. Springer, 2012.
11. J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords, In *2012 IEEE Symposium on Security and Privacy (S&P)*, pages 538–552. IEEE, 2012.
12. R. Carback, D. Chaum, J. Clark, adn J. Conway, E. Essex, P.S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P.L. Vora. Scantegrity II Municipal Election at Takoma Park: The First E2E Binding governmental Elecion with Ballot Privacy. In *USENIX 2010*. USENIX Association, 2010.
13. David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
14. M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *S&P 2008*, pages 354–368. IEEE Computer Society, 2008.
15. Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. *Journal of Computer Security*, 21(1):89–148, 2013.

16. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *ESORICS 2014*, volume 8713 of *LNCS*, pages 327–344. Springer, 2014.
17. Chris Culnane, Peter Y. A. Ryan, Steve Schneider, and Vanessa Teague. vVote: a Verifiable Voting System (DRAFT). *CoRR*, abs/1404.6822, 2014. Available at http://arxiv.org/abs/1404.6822.
18. C. Culnane, S. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *CSF 2014*, pages 169–183. IEEE Computer Society, 2014.
19. Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic Protocols. In *STOC 1982*, pages 383–400. ACM, 1982.
20. G. Grewal, M. Ryan, L. Chen, M. Clarkson Du-Vote: Remote Electronic Voting with Untrusted Computers. In *CSF 2015*, pages 155–169. IEEE Computer Society, 2015.
21. J. Heather, P. Y. A. Ryan, and V. Teague. Pretty Good Democracy for More Expressive Voting Schemes. In *ESORICS 2010*, volume 6345 of *LNCS*, pages 405–423. Springer, 2010.
22. Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability Analysis of Helios - An Open Source Verifiable Remote Electronic Voting System. In *EVT/WOTE'11*. USENIX Association, 2011.
23. F. Karayumak, M. Kauer, M. Olembo, T. Volk, M. Volkamer. User study of the improved Helios voting system interfaces. In *STAST 2011*, pages 37–44. IEEE Computer Society, 2011.
24. Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 607–625. Springer, 2012.
25. A. Kiayias, T. Zacharias, B. Zhang. End-to-End Verifiable Elections in the Standard Model. In *EUROCRYPT 2015*, volume 9057 of *LNCS*, pages 468–498. Springer, 2015.
26. A. Kiayias, T. Zacharias, B. Zhang. DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles. In *CCS 2015*, pages 352–363. ACM, 2015.
27. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. A Game-based Definition of Coercion-Resistance and its Applications. In *CSF 2010*, pages 122–136. IEEE Computer Society, 2010.
28. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *CCS 2010*, pages 526–535. ACM, 2010.
29. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountabiliy: Definition and Relationship to Verifiability. Technical Report 2010/236, Cryptology ePrint Archive, 2010. http://eprint.iacr.org/2010/236.
30. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *S&P 2011*, pages 538–553. IEEE Computer Society, 2011.
31. Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *S&P 2012*, pages 395–409. IEEE Computer Society, 2012.
32. 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.
33. S. Neumann, M. Olembo, K. Renaud, M. Volkamer. Helios Verification: To Alleviate, or to Nominate: Is That the Question, or Shall we Have Both?. In *EGOVIS 2014*, volume 8650 of *LNCS*, pages 246–260. Springer, 2014.
34. Maina M. Olembo, Steffen Bartsch, and Melanie Volkamer. Mental Models of Verifiability in Voting. In *Vote-ID 2013*, volume 7985 of *LNCS*, pages 142–155. Springer, 2013.
35. P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. The Prêt à Voter Verifiable Election System. Technical report, Universities of Luxembourg and Surrey, 2010. http://www.pretavoter.com/publications/PretaVoter2010.pdf.
36. P. Y. A. Ryan, P. B. Roenne, and V. Iovino. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. Cryptology ePrint Archive, Report 2015/1105.

37. B. Schneier Applied Cryptography. John Wiley& sons, New York, 1996.
38. D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman. Security Analysis of the Estonian Internet Voting System. In *CCS 2014*, pages 703–715. ACM, 2014.
39. J. Weber, U. Hengartner Usability Study of the Open Audit Voting System Helios. http://www.jannaweber.com/wp-content/uploads/2009/09/858Helios.pdf.
40. F. Zagórski, R. Carback, D. Chaum, J. Clark, A. Essex, and P. L. Vora. Remotegrity: Design and Use of an End-to-End Verifiable Remote Voting System. In *ACNS 2013*, volume 7954 of *LNCS*, pages 441–457. Springer, 2013.

# A Implementation of sElect

In this section, we provide an overview of our implementation of sElect (see [2] for the code). 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 the fully automated verification (a full-fledged usability study is out of the scope of this paper and left for future work).

We have implemented sElect as a web application. That is, voters simply use a browser to vote, without the need to install other software, browser extensions, or plugins, and hence, they can vote on many platforms (desktop computers, smartphones, etc.) We have implemented the core of the mix servers in Java, while the remaining parts of the servers are implemented in node.js. The cryptographic core of the system, and the code of the mix server in particular, is very simple, which, as already mentioned in the introduction, is a valuable feature of sElect. In fact, the bulk of the code has to do with user interfaces and networking.

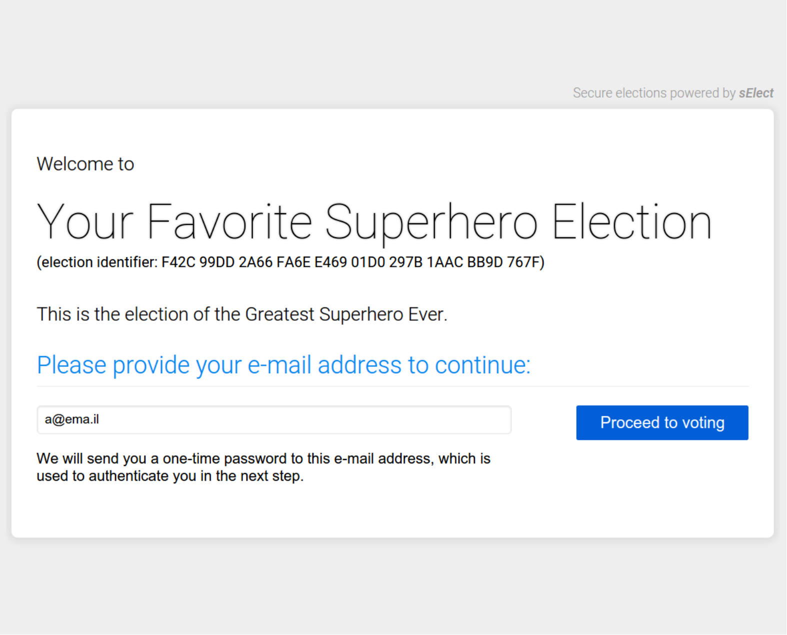
In order to vote, voters visit a web site which serves what we call a voting booth. Except for serving static files (html/JavaScript), the voting booth server does not play any role in the voting process. All the computations, including in particular ballot creation and verification of acknowledgements, are carried out locally on the voters' machine within the browser. Votes only leave the browser encrypted (as ballots), to be submitted to the authorization server.

While for the mock elections, only one server serving the voting booth was set up, the idea would be that a voter can choose a voting booth of any organization/company it trusts or even set up its own voting booth (server).

In our implementation, we use only one authorization method: one time passwords. These are sent to the voters' e-mail addresses when voters initiate the voting process. However, it would not be a problem to support different authorization methods. Also, in the current implementation of the system, the voting booth is involved in the authorization process, as described below. For example, so far the voter enters her email address into the voting booth. However, one could as well separate the voting booth from voter authentication completely so that the voting booth does not learn the voter's identity.

**Details of the implementation.** We now describe our implementation of sElect in more detail, with screenshots illustrating the user experience.

When a user opens the voting booth (in a browser), she is asked for her e-mail address (Figure 2). The voting booth forwards this e-mail address to the authentication server which (if the voter is eligible) generates a one-time password for the voter and sends it to



Welcome to

Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

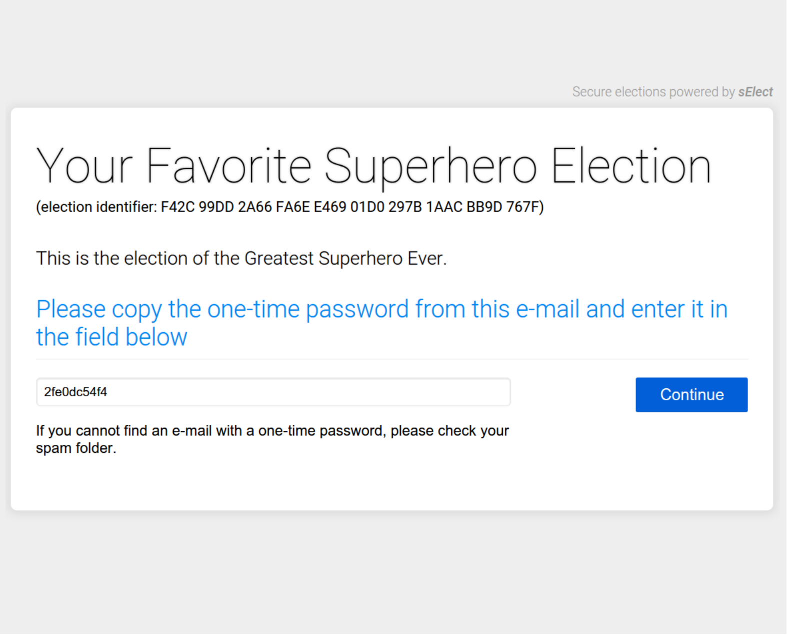
This is the election of the Greatest Superhero Ever.

Please provide your e-mail address to continue:

a@ema.il

We will send you a one-time password to this e-mail address, which is used to authenticate you in the next step.

**Fig. 2.**



Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

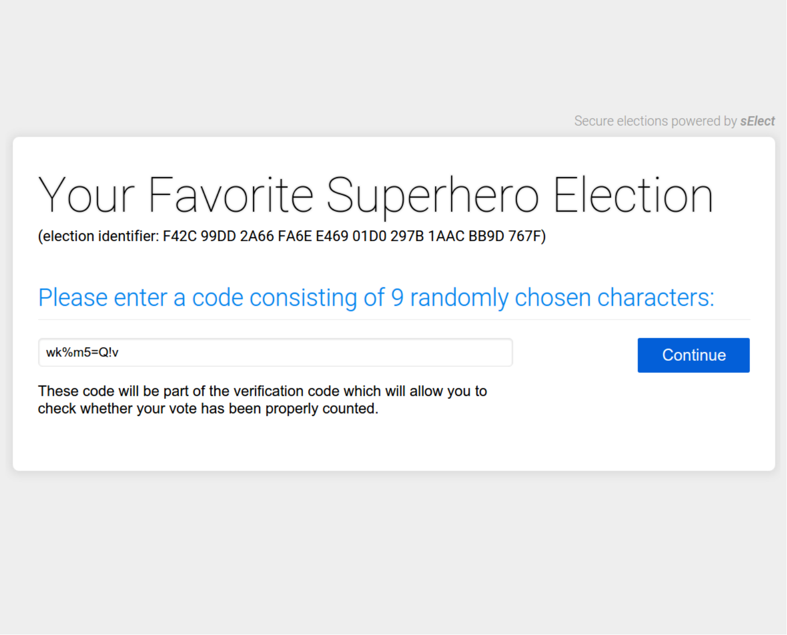
This is the election of the Greatest Superhero Ever.

Please copy the one-time password from this e-mail and enter it in the field below

2fe0dc54f4

If you cannot find an e-mail with a one-time password, please check your spam folder.

**Fig. 3.**


Secure elections powered by ***sElect***

Your Favorite Superhero Election

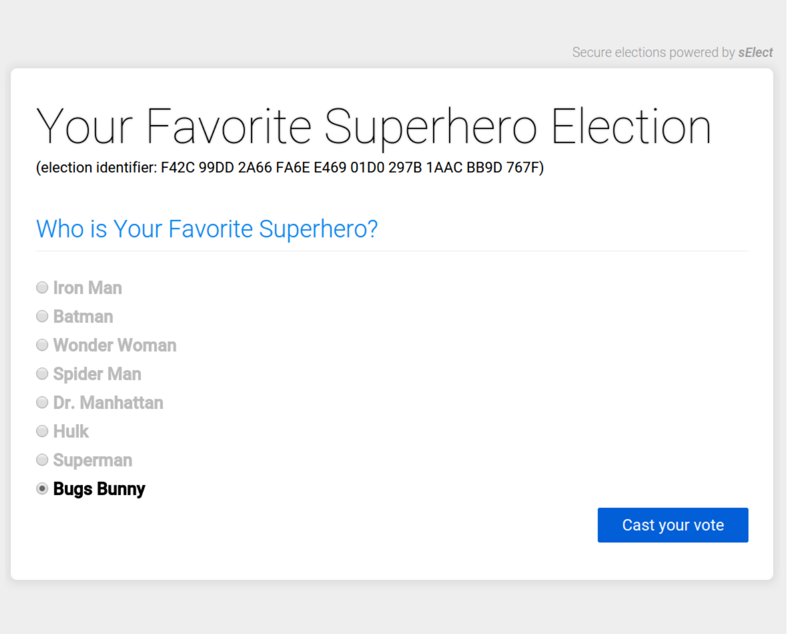
(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

Please enter a code consisting of 9 randomly chosen characters:

wk%m5=Q!v

These code will be part of the verification code which will allow you to check whether your vote has been properly counted.

**Fig. 4.**


Secure elections powered by ***sElect***

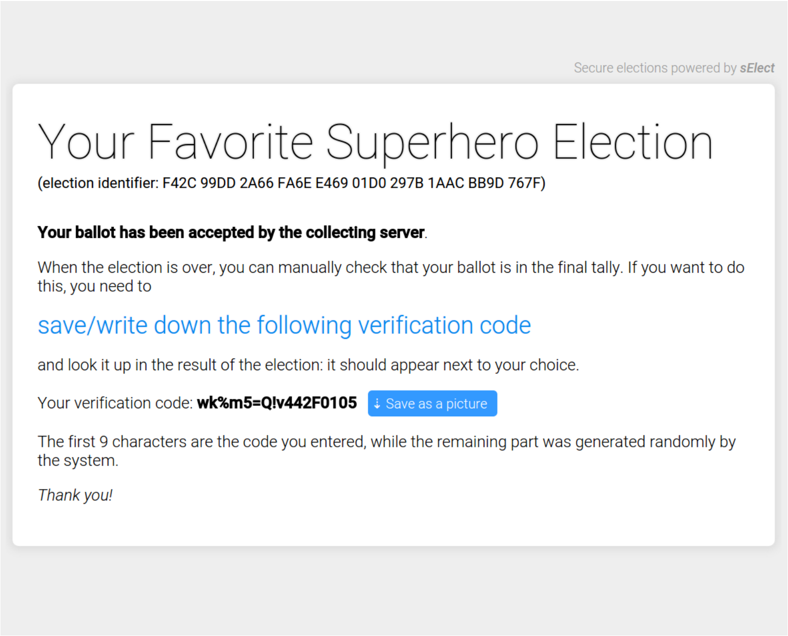
Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

Who is Your Favorite Superhero?

○ Iron Man
○ Batman
○ Wonder Woman
○ Spider Man
○ Dr. Manhattan
○ Hulk
○ Superman
● Bugs Bunny

**Fig. 5.**


Secure elections powered by *sElect*

Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

**Your ballot has been accepted by the collecting server**.

When the election is over, you can manually check that your ballot is in the final tally. If you want to do this, you need to

save/write down the following verification code

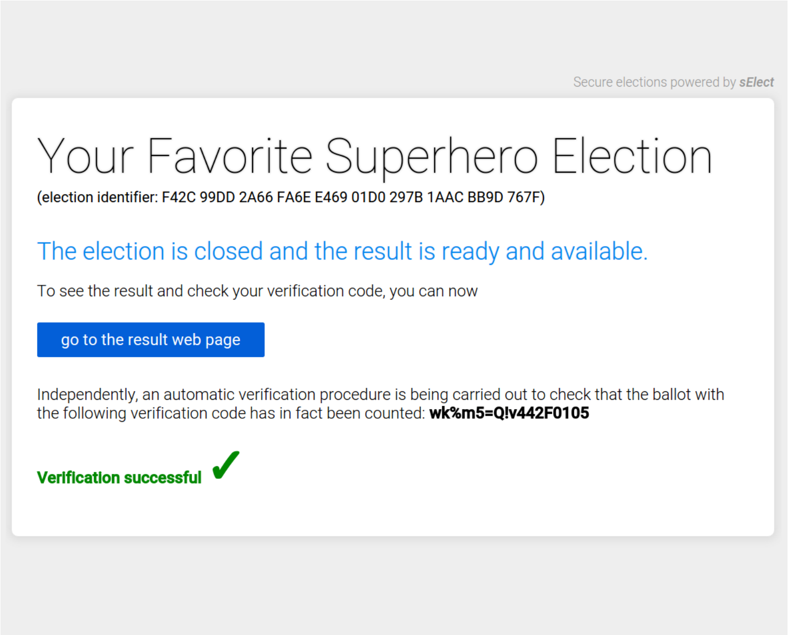
and look it up in the result of the election: it should appear next to your choice.

Your verification code: **wk%m5=Q!v442F0105**

The first 9 characters are the code you entered, while the remaining part was generated randomly by the system.

*Thank you!*

**Fig. 6.**


Secure elections powered by *sElect*

Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

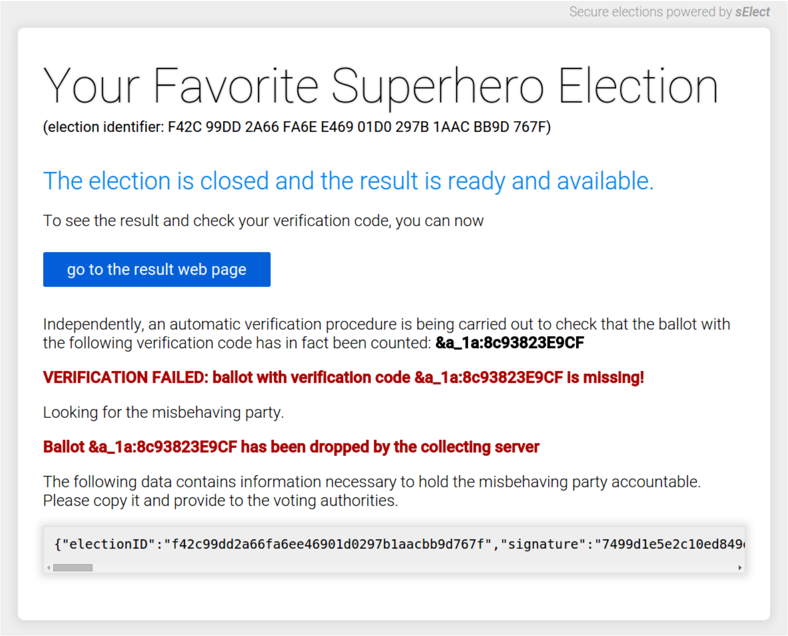
The election is closed and the result is ready and available.

To see the result and check your verification code, you can now

Independently, an automatic verification procedure is being carried out to check that the ballot with the following verification code has in fact been counted: **wk%m5=Q!v442F0105**

**Verification successful** ✓

**Fig. 7.**



Secure elections powered by *sElect*

Your Favorite Superhero Election

(election identifier: F42C 99DD 2A66 FA6E E469 01D0 297B 1AAC BB9D 767F)

The election is closed and the result is ready and available.

To see the result and check your verification code, you can now

Independently, an automatic verification procedure is being carried out to check that the ballot with the following verification code has in fact been counted: **&a_1a:8c93823E9CF**

**VERIFICATION FAILED: ballot with verification code &a_1a:8c93823E9CF is missing!**

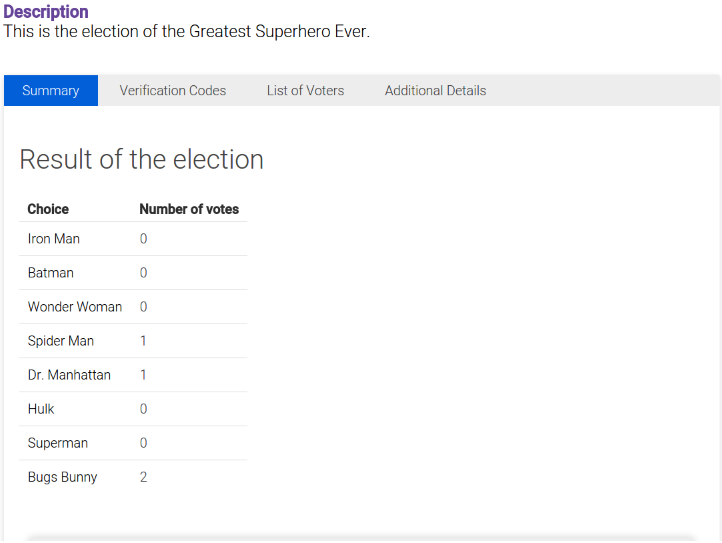
Looking for the misbehaving party.

**Ballot &a_1a:8c93823E9CF has been dropped by the collecting server**

The following data contains information necessary to hold the misbehaving party accountable.
Please copy it and provide to the voting authorities.

```
{"electionID":"f42c99dd2a66fa6ee46901d0297b1aacbb9d767f","signature":"7499d1e5e2c10ed849
```

**Fig. 8.**

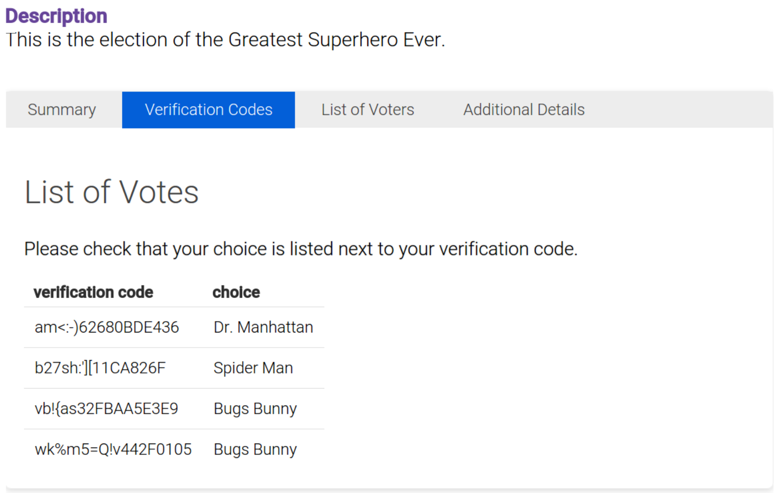


**Description**
This is the election of the Greatest Superhero Ever.

Result of the election

| Choice | Number of votes |
|---|---|
| Iron Man | 0 |
| Batman | 0 |
| Wonder Woman | 0 |
| Spider Man | 1 |
| Dr. Manhattan | 1 |
| Hulk | 0 |
| Superman | 0 |
| Bugs Bunny | 2 |

**Fig. 9.**



**Description**

This is the election of the Greatest Superhero Ever.

List of Votes

Please check that your choice is listed next to your verification code.

| verification code | choice |
|---|---|
| am<:-)62680BDE436 | Dr. Manhattan |
| b27sh:'][11CA826F | Spider Man |
| vb!{as32FBAA5E3E9 | Bugs Bunny |
| wk%m5=Q!v442F0105 | Bugs Bunny |

**Fig. 10.**

her e-mail address. The user is then supposed to enter this one-time password (copy and paste from her e-mail) to the voting booth (Figure 3). Then, the user is asked to provide a random code of nine characters, which will be used as part of the verification code (Figure 4). Next, the user is prompted by the voting booth to make her choice (Figure 5). Then, the voting booth, in the background, generates a random verification code, concatenates it with the code entered by the voter, and creates a ballot. This ballot is then, along with the one-time password, sent by the voting booth to the authentication server (also called collecting server in our implementation). The server is supposed to reply with an acknowledgement which is verified by the voting booth. After that, the verification code is displayed to the voter, who can then copy her verification code or save it as a picture (Figure 6). Independently, the verification code along with the full receipt (the data necessary to blame misbehaving parties in case something should go wrong) is stored in the browser's local storage, an HTML5 feature for storing data within the user's browser.[16] We note that data is stored in the local storage by origin.[17] By the Same Origin Policy (SOP), only JavaScript running under that origin can access this

[16] We emphasize that we deliberately wanted to keep the user interface very simple. Therefore, only the verification code is shown to the voter (a concept voters should understand). The rest of the receipt, which is used for accountability purposes, is stored and checked only by the voting booth on the voter's browser.

[17] An origin is defined by a domain name plus the information whether the connection to this domain is via HTTP or HTTPS.

data. In our case, the idea is that a voting booth runs in its own (HTTPS) origin, and hence, only (the JavaScript loaded from) this voting booth can access the receipt stored in the user's browser.

When the election is over, the voter is prompted to open her voting booth again. In our deployment, e-mails were sent to the voters informing them that the result of the election was ready and that the voter can see the result and check her verification code following a link to her *voting booth*. When the voter opens the voting booth in this phase, it fetches the information stored in the browser's local storage, which should contain the full receipt, and the result of the election from the bulletin board, and then verifies signatures and makes sure that the verification code is listed in the final result along with the chosen candidate. If this is the case, the voter is informed that her vote has been counted correctly (Figure 7). Otherwise, the evidence for blaming a (dishonest) party is generated and the voter is informed that the verification procedure failed. In particular, the complaint singles out the misbehaving party and provides evidence of the misbehavior. For instance, in Figure 8 the authentication/collecting server has been singled out as the misbehaving party. In addition to this fully automated check (carried out as soon the voter visits her voting booth), the voter is given the opportunity to visit the bulletin board, where she can see the result (Figure 9) and manually check that her verification code is listed next to her choice (Figure 10).

**Discussion.** One central point of the design of sElect is the very simple voter-based verification procedure (human verifiability), where the voter is asked to check whether her verification code and choice appear in the result. Another central point of the design is also the fully automated verification, which, in addition, is coupled with the act of looking up the election outcome: while voters are typically interested in the election outcome, less voters might be interested in the verification process. Since the verification process is triggered and carried out fully automatically without any effort by the voter, almost all voters who check the election outcome seamlessly and automatically also perform the verification (see also the remark below). The voter-based (manual) verification can be seen as an orthogonal mechanism, which does not assume trust in other parties or devices (except that the voter needs to be able to look at the election outcome). This gives the voters direct understanding that their votes were actually counted.

Under some circumstances, for example, if voters clean their browsing data, which may include the local storage, or if they use different browsers to cast their votes and to verify the result, the automated verification procedure cannot be carried out. Note that, even in this case, the voter can perform the manual verification. Also, note that to guarantee a high level of accountability/verifiability and privacy for sElect, we do not require that 100% of the voters check the result; much less suffices to make manipulation very risky, as proven in Sections 4, 5, and 6. With our mock elections (see below) we provide first estimates about the verification ratio for fully automated verification when the system is used in practice.

**Mitigating coercion resistance.** The sElect voting system, just as Helios, was not designed to provide coercion resistance and, in fact, in the current implementation, vote selling and voter coercion is quite easy: a voter can simply forward her verification code to a coercer who can use this code to check which candidate the voter voted for.

Still, to mitigate this problem to some extent and to make coercion less easy, one can consider the following variants of sElect.

First, we can consider a variant where voter-based verification is dropped and the verification codes are not shown to the voter (but only stored internally in the browser's local storage). This, of course, means that the verification would be done only automatically by the voting booth; the voter could not carry out manual verification.

One can also consider a variant where the whole receipt, including the verification code (again the voter's part of the code would be dropped), is not computed and stored within the browser but on the server site of the voting booth. In this variant, as opposed to the implemented variant, the voting booth server plays an active role. In particular, it would perform the verification procedure itself (or delegate this to another party).

Note that without trusting the voting booth, coercion resistance and even privacy would be much harder to achieve. We emphasize that the voter is free to choose a voting booth she trusts, and as discussed before, for low-risk and low-coercion elections trusting the voting booth (and the client platform) will in many cases be reasonable.

**Two small mock elections.** To obtain user feedback and, in particular, get 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 our field tests were not meant to be full-fledged and systematic usability studies, which we leave for future work.

The participants in our first mock election were students of our department (who voted for the "Greatest Superhero Ever"): 52 voters cast their ballots and 30 (out of 52) verification codes/receipts were checked automatically by the voting booth.[18] This gives a verification ratio of 57.7%. We can expect that some number of verification codes were checked manually, so the overall verification ratio might be even higher. However, we do not have reliable data about voter-based verification.

The participants of our second mock election were researchers of a national computer science project (who voted on their favorite text editor). In this case, we recorded 22 cast ballots and 13 (out of 22) verification codes/receipts were checked automatically, which gives a verification ratio of 59.1%.[19] Again, the overall verification ratio might be even higher considering possible voter-based verification.

As one can see, in both cases, the verification ratio was quite high. 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 4, 5, and 6.

The hope is 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, when invited, for example by email, to take a look at the election outcome via the voting booth, the verification procedure carried out in the voting booth within the browser will be triggered automatically.

[18] Bugs Bunny won.

[19] Emacs won.

# B Formal Model of sElect

**Set of participants.** In addition to the participants listed in Section 3, we also have a judge $J$ (see Appendix D).

In what follows, we define the set $\Pi_a$ of programs of all agents $a$ in $P_{sElect}$, including their honest program $\hat{\pi}_{\mathsf{a}}$.

**The sets $\Pi_a$ (trust assumptions).** We assume that the scheduler $S$, the judge $J$ and the bulletin board $B$ are honest. Technically, this means that the set of programs of each of these agents contains only one program, namely, the honest one. All other agents can possibly be dishonest. For these agents, the sets of their programs contain all probabilistic polynomial-time programs.

Next, we describe the honest programs $\hat{\pi}_{\mathsf{a}}$ of every agent $a$ in $P_{sElect}$.

**Judge $J$.** The honest program of $J$ carries out the procedure described in Appendix D.2.

**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 choice $m_i$ according to the probability distribution $\mu$. A choice may be either a distinct value $\mathsf{abstain}$, which expresses abstention from voting, or a candidate name (or whatever real choices are possible). If $m_i = \mathsf{abstain}$, then the voter program stops. Otherwise, if $m_i$ is a candidate name, the program continues and produces a random nonce $n_i^{voter}$. She then sends $(m_i, n_i^{voter})$ 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 $m_i$ was $\mathsf{abstain}$ or not. If $m_i$ was $\mathsf{abstain}$, the voter, with probability $p_{abst}^{verif}$, verifies that her name is not listed in the list $LN$ of names output by the authentication server. She files a complaint if this is not the case, as described in Section 2. If $m_i \neq \mathsf{abstain}$, the voter, with probability $p_{voter}^{verif}$, follows the verification procedure to check that her choice/verification code-pair is listed in the final result. If this is not the case, she files a complaint as described in Section 2.

**Voter supporting device $vsd_i$.** When the voter supporting device $vsd_i$ receives a tuple $(m, n)$ by $v_i$, it produces and casts a ballot as described in Section 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 $p_{vsd}^{verif}$, follows the verification procedure to check that $v_i$'s choice/verification code-pair is listed in the final result. If this is not the case, it files a complaint as described in Section 2.

**Authentication server** $AS$**.** The honest authentication server $AS$ carries out the steps described in Section 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 2.

**Scheduler** $S$**.** In every instance of $P_{sElect}$, the honest program $\hat{\pi}_S$ of $S$ plays the role of the master program (in the sense of Section 3). 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 D), 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 $\ell$ 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 $LN$ (containing the names of those eligible voters who cast a valid ballot) and the list $C_0$ (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 $M_m$ (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 $p^{verif}_{abst}$ whether her name appears in the list *LN*, and, if this is not the case, to file a complaint as described in the description, section 2. If the voter did not abstain, she and her VSD are supposed to verify with probability $p^{verif}_{voter}$ and $p^{verif}_{vsd}$, respectively, whether the voter's submitted choice appears in the final result $C_m$, and, if this is not the case, to file a complaint as described in the description, section 2. Afterwards, the scheduler triggers all the dishonest voters.

## C Verifiability Proof

In this section we prove Theorem 1 which expresses the verifiability property of the sElect protocol.

*Proof.* Let *accept* be the property of the sElect protocol $P_{sElect}$ which consists of those runs which are accepted by all voters, in particular all honest voters. Let *no clash* be the property of the sElect protocol $P_{sElect}$ which consists of those runs in which no clashes of voter-generated verification codes occur.

Let $\pi$ be an instance of the protocol $P_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{vsd}, p^{verif}_{abst})$. We have to prove that

$$\Pr\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge no\ clash)\right] \leq p_{noclash} \cdot \left(1 - \min\left(p^{verif}_{voter}, p^{verif}_{abst}\right)\right)^{k+1}$$

holds true.

We denote the set of those honest voters whose choices are not included in the pure result of a run by $V$.[20] Recall that, if $\gamma_k$ is not satisfied in a run, then we have $|V| \geq k+1$. Let $verify_V$ be the property that at least one of the honest voters in $V$ or her associated voter supporting device verifies the final result as described in section 2.

We will first show that

$$\begin{aligned} & \Pr\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge verify_V \wedge no\ clash)\right] \\ \leq\ & \Pr\left[\pi(1^\ell) \mapsto (accept \wedge verify_V \wedge no\ clash)\right] \\ =\ & 0 \end{aligned}$$

holds true. To prove this, we argue that for each voter $v_i$ in $V$ the probability that the voter verifies and also accepts the final result is 0. Let $r$ be a run of $\pi$. Now, let $v_i$ be a voter in $V$. The first possibility for a voter $v_i$ to be in $V$ is that $v_i$ did not submit a ballot

[20] A pure result describes the final result without the choices of the dishonest voters (see Appendix F).

but her actual choice ”abstention” is not included in the pure result. Therefore, her name appears in the list of names *LN* which is published by the authentication server *AS*. If the voter $v_i$ then performs the verification procedure described in section 2, she will observe that her name is in *LN*. Therefore, $v_i$ complains and does not accept the run. The second possibility for a voter $v_i$ to be in $V$ is that the voter $v_i$ created a vote-nonce pair $(m_i, n_i^{voter})$ and gave the tuple to her voter supporting device $vsd_i$ (which is supposed to produce and submit the ballot) but her actual choice $m_i$ is not included in pure result. If $v_i$ then performs the verification procedure described in section 2, she will check whether $(m_i, n_i^{voter} \| x)$ is in the final result $C_m$ for some number $x$. If this is not the case, $v_i$ complains and does not accept the run. Otherwise, if $(m_i, n_i^{voter} \| x)$ is in the final result $C_m$ for some number $x$, there must a voter $v_j$, $i \neq j$, such that $(m_i, n_i^{voter}) = (m_j, n_j^{voter})$. If, however, *no clash* holds true, this case is impossible. To see this, recall that in the definition of $\gamma_k$ only honest voters are considered. Therefore, we can conclude that we have $\mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge verify_V \wedge no\ clash)\right] = 0$.

For all voters $v_i$ and all voter supporting devices $vsd_i$, the probability that the individual verification procedure is not carried out is $\leq 1 - \min\left(p_{voter}^{verif}, p_{abst}^{verif}\right)$. In addition, the decision of each voter whether to verify the final result is (information theoretically and, thus, statistically) independent of, first, the decision of each other voter to verify, second, of whether $\gamma_k$ holds true or not, and third, of whether *no clash* holds true or not. Therefore, we have that

$$\begin{aligned}
& \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge \neg verify_V \wedge no\ clash)\right] \\
\leq\ & \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge \neg verify_V) \wedge no\ clash\right] \\
\leq\ & p_{noclash} \cdot \left(1 - \min\left(p_{voter}^{verif}, p_{abst}^{verif}\right)\right)^{k+1}
\end{aligned}$$

holds true.

By what we have shown above and the fact that

$$\begin{aligned}
& \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept)\right] \\
=\ & \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge no\ clash)\right] \\
& + \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge \neg no\ clash)\right] \\
\leq\ & \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge verify_V \wedge no\ clash)\right] \\
& + \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge accept \wedge \neg verify_V \wedge no\ clash)\right] \\
& + \mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg no\ clash)\right] \\
\leq\ & p_{noclash} \cdot \left(1 - \min\left(p_{voter}^{verif}, p_{abst}^{verif}\right)\right)^{k+1} + p_{clash},
\end{aligned}$$

the claim follows. □

# D Accountability

In this section, we first briefly recall from [28] the general, domain-independent definition of accountability and its instantiation to e-voting. We then use this definition to precisely specify and prove the level of accountability sElect provides.

As demonstrated in [28], verifiability is a weaker form of accountability. For verifiability, one requires only that, if some goal of the protocol is not achieved (e.g., the election outcome does not correspond to how the voters actually voted, see Section 4.1), then the judge will not accept such a run, but he is not required to blame misbehaving parties. Conversely, accountability requires that misbehaving parties are blamed. In practice, as already pointed out in the introduction and other work (see, e.g., [28]), it is very important that an e-voting scheme provides accountability, not only verifiability.

## D.1 Definition of Accountability

The definition of accountability is w.r.t. an agent $J$ of the protocol who is supposed to blame protocol participants in case of misbehavior. The agent $J$, sometimes referred to as a *judge*, can be a "regular" protocol participant or an (external) judge, who is typically provided with additional information by other, possibly untrusted protocol participants.

Informally speaking, accountability requires two conditions to be satisfied:

(i) (*fairness*) $J$ (almost) never blames protocol participants who are honest, i.e., run their honest program.

(ii) (*completeness*) If, in a run, some desired goal of the protocol is not met—due to the misbehavior of one or more protocol participants—then $J$ blames those participants who misbehaved, or at least some of them (see below).

As in the case of verifiability (see Section 4.1) a desired goal for voting protocols would be that the published result of the election corresponds to the actual votes cast by the voters. The completeness condition then guarantees that if in a run of the protocol the published result of the election does not correspond to the actual votes cast by the voters (a fact that must be due to the misbehavior of one or more protocol participants), then one or more participants are held accountable by $J$; by the fairness condition they are *rightly* held accountable.

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

**Verdicts.** A verdict can be output by the judge (on a dedicated output channel) and states which parties are to be blamed (that is, which ones, according to the judge, have misbehaved). In the simplest case, a verdict can state that a specific party $a$ misbehaved (behaved dishonestly). Such an *atomic verdict* is denoted by $\mathsf{dis}(a)$. It is also useful to state more fine grained or weaker verdicts, such as "$a$ or $b$ misbehaved". Therefore, in the general case, we will consider verdicts which are boolean combinations of atomic verdicts. In fact, in our formal analysis of sElect, we use in some cases verdicts of the form $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$ stating that either the $i$-th voter $v_i$ or the authentication server $AS$ misbehaved (but the verdict leaves open, as it might not be clear, which one of them). Given an instance $\pi$ of a protocol $P$, we say that a verdict $\psi$ *is true in* $\pi$, written $\pi \models \psi$, iff the formula $\psi$ evaluates to true with the proposition $\mathsf{dis}(a)$ set to false if $a$ is honest in $\pi$ (as defined in Section 3), and set to true otherwise.

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

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

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

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

**Computational fairness.** An agent $J$ is *computationally fair* in $P$, if, for all instances $\pi$ of $P$, $J$ states false verdicts only with negligible probability, where a verdict $\psi$ is false in a run $r$ of $\pi$ if $\pi \not\models \psi$.

**Definition 5 (Accountability [28]).** *Let $P$ be a protocol with the set of agents $\Sigma$, $J \in \Sigma$, an accountability property $\Phi$ of $P$, and $\delta \in [0,1]$. We say that $J$* ensures $(\Phi, \delta)$-accountability for protocol $P$ *(or $P$* is $(\Phi, \delta)$-accountable w.r.t. $J$*) if*

*(i) (*Fairness*) $J$ is computationally fair in $P$ and*

*(ii) (*Completeness*) for every instance $\pi$ of $P$, the probability* $\mathsf{Pr}\left[\pi(1^\ell) \mapsto \neg(J : \Phi)\right]$ *is $\delta$-bounded as a function of $\ell$.*

In the completeness condition, it is of course desirable that $\delta = 0$, i.e., the probability that $J$ fails to ensure a constraint is negligible. However, this is typically too demanding, as illustrated already in [28], and also by our formal analysis of sElect presented below.

### D.2 Analysis of sElect

To apply the definition of accountability to sElect, we first extend the modeling of this protocol, as given in Appendix B, by adding an additional agent $J$, namely the *judge* whose role it is, as explained above, to state verdicts (blaming misbehaving parties). We also extend the scheduler's program to make sure that the judge is triggered sufficiently often. Additionally, since the (human) voter does not play a role for accountability, we will identify each voter with her voter supporting device. In this model, we will then state the level of accountability provided by sElect for the goal $\gamma_k$ as defined in Section 4.1.

**Unification of voter and VSD.** Recall that in sElect, the VSDs keep all the information necessary to be able to individually blame (potentially) misbehaving servers. In contrast, voters only keep the user generated part of the verification code which, in case their vote-nonce pair is not included in the final result, does not provide evidence as to which party is to be blamed. This means that the voter-based verification does not contribute to accountability. Therefore, in what follows, we consider only the automated verification procedure which is carried out by the VSD and distinct between the voter and her VSD anymore, but 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 $p_{voter}^{verif} \in [0,1]$, and the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $LN$ output by the authentication server by $p_{abst}^{verif} \in [0,1]$.

**Judging procedure of sElect.** We assume that $J$ is honest. (Formally, this means that $\Pi_J$ in the protocol $P_{sElect}$ contains the honest program $\hat{\pi}_J$ of $J$ only.) 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 program $\hat{\pi}_J$, 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 $\text{dis}(S)$, 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 $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$, 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 $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$.

(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 $\alpha$, while $\alpha$ is not listed in the output $C_0$ of $AS$, then the judge blames $AS$ outputting the verdict $\text{dis}(AS)$.

(J5) If, in the verification phase, a valid complaint of the form $(j, \alpha, r)$ is (anonymously) posted, i.e., $\mathsf{Enc}_{pk_j}^{r}(\alpha)$ is in $C_{j-1}$, but $\alpha$ is not in $C_j$, then the judge blames $M_j$ outputting the verdict $\text{dis}(M_j)$.

**Accountability property.** We investigate the accountability level of sElect with respect to the goal $\gamma_k$ as defined in Definition 1. Recall that $\gamma_k$ formalizes the property that malicious participants cannot manipulate or drop more than $k$ honest votes. We will now define an accountability property $\Phi_k$ which covers the goal $\gamma_k$.

Let $\chi_i$ contain all runs of $P_{sElect}$ where (J2) occurs, i.e., the voter complains that she did not get a receipt from $AS$. Similarly, let $\chi_i'$ contain all runs of $P_{sElect}$ where (J3) occurs, i.e., the voter complains that she did not vote, but her identifier is listed in $LN$ published by $AS$. Let $\chi = \bigcup_{i\in\{1,\dots,n\}} \chi_i \cup \chi_i'$.

Now, we define $\Phi_k$ as the accountability property consisting of the following constraints (for $i \in \{1,\dots,n\}$):

$$\begin{aligned}\chi_i &\Rightarrow \mathsf{dis}(v_i) \vee \mathsf{dis}(AS)\\ \chi_i' &\Rightarrow \mathsf{dis}(v_i) \vee \mathsf{dis}(AS)\\ \neg\gamma_k \wedge \neg\chi &\Rightarrow \mathsf{dis}(AS) \mid \mathsf{dis}(M_1) \mid \dots \mid \mathsf{dis}(M_m)\end{aligned}$$

Clearly, this accountability property covers $\neg\gamma_k$ by construction, i.e., if $\gamma_k$ 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 $\Phi_k$ requires that except for the cases where $\chi$ holds, whenever the goal $\gamma_k$ is violated, an individual party is blamed. Conversely, if $\chi_i$ occurs, the judge cannot be sure whether $AS$ or a voter $v_i$ misbehaved. As already discussed in Section 2, this is a very general problem, which applies to virtually any remote voting protocol but for which there are pragmatic solutions. The case $\chi_i'$ 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 $p^{verif}_{voter} \in [0,1]$ is the probability that an honest voter who does not abstain from voting verifies the result, and $p^{verif}_{abst} \in [0,1]$ is the probability that an honest voter who abstains from voting verifies that her name is not listed in the list $LN$ output by the authentication server.

**Theorem 3 (Accountability).** *Let the judge J be as defined above. Then J ensures* $\left(\Phi_k, \delta^k(p^{verif}_{voter}, p^{verif}_{abst})\right)$*-accountability for* $P_{sElect}(n,m,\mu,p^{verif}_{voter},p^{verif}_{abst})$*, where*

$$\delta^k(p^{verif}_{voter}, p^{verif}_{abst}) = \left(1 - \min\left(p^{verif}_{voter}, p^{verif}_{abst}\right)\right)^{k+1}.$$

The proof of this theorem follows from Lemma 2 and Lemma 3 (see below). It is a rather straightforward reduction argument, which uses only EUF-CMA-security of the signature scheme and correctness of the encryption scheme.

This theorem means that the probability that more than $k$ votes of honest voters have been dropped or manipulated, but the judge nevertheless did not blame any party, is at most $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$. If, as discussed above, voters would sign their ballots, then one could get rid of $p^{verif}_{abst}$ in the definition of $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$, as in this case one could require $AS$ to provide a signed ballot for every voter listed in $LN$. 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.

**Lemma 2 (Fairness).** *The judge $J$ is computationally fair in $P_{sElect}(n,m,\mu,p_{voter}^{verif},p_{abst}^{verif})$.*

*Proof.* To prove fairness we will show that, with overwhelming probability, the judge does not post incorrect verdicts.

In a run with property $\chi_i$, the voter $v_i$ complains that she did not receive a valid acknowledgement by $AS$ although she submitted a valid ballot. If the judge $J$ reads such an authenticated complaint, it outputs $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$ (see (J2)). Since the complaint is authenticated and the bulletin board $B$ is honest, the complaint was indeed posted by $v_i$. We consider two cases. First, if the voter is dishonest, the verdict clearly is true. Otherwise, if voter $v_i$ is honest, by the definition of the honest program, the voter submitted a valid ballot but did not receive a valid acknowledgement. In this case, $AS$ must have misbehaved. Therefore, in this case the verdict $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$ holds true as well.

In a run with property $\chi_i'$, the voter $v_i$ complains that she did not vote although her identifier is listed in $LN$ published by $AS$. The list $LN$ is supposed to contain the identifiers of those voters who submitted a vote to $AS$. If the judge $J$ reads such an authenticated complaint, it outputs $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$ (see (J3)). Again, the complaint was indeed posted by $v_i$. Additionally, the list $LN$ was signed by $AS$ with overwhelming probability; otherwise, the process would have been aborted before this complaint. Using analogous reasoning as above, we can infer that the verdict $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$ holds true with overwhelming probability.

According to (J1), the judge states $\mathsf{dis}(S)$ for some server $S$ if the server did not publish a result when expected, or the published data was not in the expected format. Clearly, this verdict is fair because the bulletin board $B$ is honest.

According to (J4), the judge states $\mathsf{dis}(AS)$ if someone posted a complaint in the verification phase which contains a valid acknowledgement of $AS$ for a ballot that does not appear in the list $C_0$ signed by $AS$. We demonstrate that, if $AS$ is honest, then this may happen only with negligible probability. First, the list $C_0$ carries a valid signature of $AS$. Because the honest program of $AS$ never reveals its private keys, $C_0$ must have been produced by $AS$ (with overwhelming probability, by the security of the used signature scheme). Second, the acknowledgement also carries a valid signature of $AS$. Again, it means that (with overwhelming probability) it was produced by $AS$. However, the honest program of the authentication server always puts a ballot for which it signs an acknowledgement in the list $C_0$. Therefore, the case considered here does not occur if $AS$ is honest, except for the negligible set of runs, where one of the signatures was forged.

According to (J5), the judge states $\mathsf{dis}(M_j)$ if someone posted a complaint in the verification phase that contains a triple $(j,\alpha,r)$ for which $\mathsf{Enc}_{pk_j}^r(\alpha)$ is in $C_{j-1}$ while $\alpha$ is not in $C_j$. Since all lists $C_0,\ldots,C_m$ are signed correctly (otherwise the run would have been aborted immediately after the respective publication), each published list was indeed posted by the respective server, except for a set of runs of negligible probability where fake signatures are forged. There are three reasons for an honest mix server not to add the decrypted result of an entry from the input list to the output list. First, if the entry cannot be decrypted. Second, if the decrypted entry is not in the correct format. Third, if the decrypted result is a duplicate of a different entry which is already in the output list. However, none of these reasons can hold true for a triple $(j, \alpha, r)$ as above because the encryption scheme is correct. Therefore, in an overwhelming set of runs the verdict $\mathsf{dis}(M_j)$ is fair.

The above cases cover all possible runs where a verdict is stated by the judge, which completes the proof. □

**Lemma 3 (Completeness).** *For every instance $\pi$ of $P_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$, we have*

$$\mathsf{Pr}\left[\pi(1^\ell) \mapsto \neg(J : \Phi_k)\right] \leq \delta^k(p^{verif}_{voter}, p^{verif}_{abst}) = \left(1 - \min\left(p^{verif}_{voter}, p^{verif}_{abst}\right)\right)^{k+1}$$

*with overwhelming probability as a function of $\ell$.*

*Proof.* In order to prove the lemma, we have to show that the probabilities

$$\mathsf{Pr}\left[\pi(1^\ell) \mapsto (\chi_i \wedge \neg\mathsf{dis}(v_i) \wedge \neg\mathsf{dis}(AS))\right] \tag{2}$$

$$\mathsf{Pr}\left[\pi(1^\ell) \mapsto \left(\chi'_i \wedge \neg\mathsf{dis}(v_i) \wedge \neg\mathsf{dis}(AS)\right)\right] \tag{3}$$

$$\mathsf{Pr}\left[\pi(1^\ell) \mapsto (\neg\gamma_k \wedge \neg\chi \wedge \neg\mathsf{dis}(AS) \wedge \ldots \wedge \neg\mathsf{dis}(M_m))\right] \tag{4}$$

are $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$-bounded for every $i \in \{1, ..., n\}$.

The first two probabilities, (2) and (3), are equal to 0. In fact, if a voter complains in an authenticated way that she did not receive a valid acknowledgement although she submitted a valid ballot (i.e., when $\chi_i$ holds true), or if the voter complains in an authenticated way that she abstained from voting although her name appears in $C_0$ (i.e., when $\chi'_i$ holds true), then, by the definition of the honest programs, the honest bulletin board publishes the respective complaint and the judge outputs the verdict $\mathsf{dis}(v_i) \vee \mathsf{dis}(AS)$.

To complete the proof, we need to show that the probability (4) of the event

$$X = (\neg\gamma_k \wedge \neg\chi \wedge \neg\mathsf{dis}(AS) \wedge \ldots \wedge \neg\mathsf{dis}(M_m))$$

is $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$-bounded.

Let $\beta = \chi_1 \cup \cdots \cup \chi_n$ and $\beta' = \chi'_1 \cup \cdots \cup \chi'_n$ (note that $\neg\chi = \neg\beta \wedge \neg\beta'$). Let $IB$ denote the event that no individual blame is stated by the judge and, 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 $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$-bounded, we will show a stronger fact, namely, we will show that $\mathsf{Pr}\left[\neg IB \wedge \neg\beta' \mid Y\right] \leq \delta^k(p^{verif}_{voter}, p^{verif}_{abst})$ (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 $\alpha^i_m$ does not appear in the result list).

Now, let $\omega$ and $\langle\omega',\omega_v\rangle$ be defined as in Section 6.1. Recall that $\omega'$ completely determines the run of the protocol up to the verification phase. In particular $\omega'$ determines the output of the last mix server and it determines whether the goal $\gamma_k$ is satisfied or not ($\gamma_k$ does not depend on $\omega_v$) and whether $\beta$ is satisfied or not. It means, in particular, that either $\omega' \subseteq Y$ or $\omega' \cap Y = \emptyset$. (As mentioned in Section 6.1, we can consider $\omega'$ to be an event.)

Let $\Omega_Y$ be the set of those $\omega'$ that are inside $Y$. To complete the proof, it is enough to show that, for each $\omega' \in \Omega_Y$ we have

$$\mathsf{Pr}\left[\neg IB \wedge \neg\beta' \mid \omega'\right] \leq \delta^k(p^{verif}_{voter}, p^{verif}_{abst}). \tag{5}$$

Let us recall that $\omega'$ completely determines the run up to the audit coins (which are drawn, when the result is already determined). In particular, $\omega'$ 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 $\mathsf{Pr}\left[\neg IB \wedge \neg\beta' \mid \omega'\right] = 0$. Otherwise, if a result is output, $\omega'$ also determines the set $V_1$ of those honest voters who did not vote, but are listed in $LN$ and the set $V_2$ of those honest voters $v_i$ who cast their ballots, but their vote/verification code pairs $\alpha^i_m$ are not listed in the final result. One can see, by the definition of the goal $\gamma_k$ (which is violated in $\omega'$), that $|V_1| + |V_2| > k$ (otherwise this goal would not be violated).

Now given $V_1$ and $V_2$, it is easy to compute the probability $\neg IB \wedge \neg\beta'$ given $\omega'$: this events happens only when none of the voters in $V_1$ and $V_2$ verifies the result. Note that, indeed, if a voter in $V_1$ verifies the result, she complains and $\beta'$ is automatically satisfied. Similarly, if a voter in $V_2$ 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 $(1-p^{verif}_{abst})^{|V_1|}(1-p^{verif}_{voter})^{|V_2|}$, because the voters in $V_1$ and $V_2$ carry out the verification process with probability $p^{verif}_{abst}$ and $p^{verif}_{voter}$, respectively, independently of anything else (the random coins used in this choices are independent of $\omega'$ and of each other). Recall that $|V_1| + |V_2| > k$. Therefore we have:

$$\begin{aligned}\mathsf{Pr}\left[\neg IB \wedge \neg\beta' \mid \omega'\right] &= (1-p^{verif}_{abst})^{|V_1|}(1-p^{verif}_{voter})^{|V_2|} \\ &\leq (1-\min(p^{verif}_{abst}, p^{verif}_{voter}))^{k+1} \\ &= \delta^k(p^{verif}_{voter}, p^{verif}_{abst}).\end{aligned}$$

This completes the proof. □

## E Individual Blaming of *k*-semihonest Adversaries

In this section, we provide the proof of Lemma 1.

*Proof.* If in the runs of $\omega'$ no result is published, then, by the judging procedure, a server is blamed individually. The probability of this is thus 1.

Otherwise, if in $\omega'$ a result is published and the nonces (verification codes) chosen by the voters are pairwise different (this is obviously the case for all or none of the runs in $\omega'$), then the adversary is not caught cheating only if none of the honest voters whose ballots were dropped verify the result. Note that if the adversary is caught cheating, then he can be blamed individually because, by assumption, honest voters receive an acknowledgment when they cast a ballot from the authentication server. So, when the authentication server misbehaved he can be blamed individually. Also, in any case if one of the mix servers misbehaved they can be blamed individually as well.

Since the runs in $\omega'$ are not $k$-semi-honest, more than $k$ vote-nonce pairs of honest voters must have been dropped/manipulated. So, the probability that none of these voters verify the result is $(1-p^{verif}_{voter})^{k+1}$.

Since the nonces generated by voters (VSDs) for the verification codes have length $\geq \ell$ (security parameter), the probability that nonces chosen by honest voters coincide is negligible. Hence, the above argument holds true for all but negligible many $\omega'$. □

# F Privacy Proof

In this section, we prove Theorem 2 which establishes the privacy level of sElect. This level can be expressed using the privacy level of the ideal voting protocol. Therefore, we formally introduce the ideal voting protocol in the following definition.

**Definition 6 (Ideal voting protocol).** *The ideal voting protocol $\mathsf{P}_{ideal}(n,\mu)$ consists of the following participants. There are n honest voters. We denote the program of an honest voter by $\hat{\pi}_{v_i}$, $1 \leq i \leq n$. There is one honest voting authority with direct channels to each single honest voter. We denote its honest program by $\hat{\pi}_{VAI}$. There is one (dishonest) observer. The set $\Pi_O$ of programs of the observer contains all probabilistic polynomial-time programs. All agents are connected via direct channels.*

*The honest program $\hat{\pi}_{v_i}$ of each voter can be triggered by the voting authority. In this case, $\hat{\pi}_{v_i}$ chooses a candidate according to the distribution $\mu$ and outputs the choice on the direct channel to the voting authority. The honest program $\hat{\pi}_{VAI}$ of the voting authority can be triggered by the observer which plays the role of the master program. In this case, $\hat{\pi}_{VAI}$ successively triggers all honest voters and collects all choices that are submitted via direct channels from the honest voters. When every honest voter has been triggered, and hence, all votes have been collected, the voting authority randomly permutates all collected votes and sends the resulting list to the observer.*

*Altogether, an instance of $\mathsf{P}_{ideal}(n,\mu)$ is of the form $\pi_O \parallel \hat{\pi}_{v_1} \parallel \cdots \parallel \hat{\pi}_{v_1} \parallel \hat{\pi}_{VAI}$ with $\pi_O \in \Pi_O$. We also consider instances of the form $\pi_O \parallel \hat{\pi}_n(c) \parallel \hat{\pi}_{VAI}$ where $\hat{\pi}_n(c)$ is the composition of $n-1$ honest voters plus an honest voter under observation. This voter votes for c instead of making her choice according to $\mu$.*

We now recall the level of privacy $\delta^{id}_{l,\mu}$ of the above ideal voting system from [30]. As we will see, this level depends on the number $l$ of honest voters and the probability distribution $\mu$ used by the honest voters to determine their choices.

To define this level, we need the following terminology. Let $\{c_1, \dots, c_k\}$ be the set of valid choices. Since the adversary knows the choices of the dishonest voters, he can subtract these choices from the final result and obtain the so-called *pure result* $r = (r_1, \dots, r_k)$ of the protocol, where $r_i$, $i \in \{1, \dots, k\}$, is the number of votes for $c_i$ in the result, after subtracting the votes of dishonest voters. Note that, if $l$ is the number of honest voters, then $r_1 + \cdots + r_k = l + 1$ ($l$ honest voters plus the voter under observation). We denote by *Res* the set of all pure results. Let $A^i_r$ denote the probability that the choices made by the honest voters yield the pure result $r$, given that the voter under observation submits $c_i$. Note that $A^i_r$ depends on $l$ and $\mu$. Further, let $M_{j,j'} = \{r \in Res : A^j_r \le A^{j'}_r\}$. Now, $\delta^{id}_{l,\mu}$ is obtained according to the following intuition: If the observer, given a pure result $r$, wants to decide whether the observed voter submitted $c_j$ or $c_{j'}$, the best strategy of the observer is to opt for $c_{j'}$ if $r \in M_{j,j'}$, i.e., the pure result is more likely if the voter submitted $c_{j'}$. Now the level of privacy for the ideal voting protocol with $l$ honest voters, as established in [30], is

$$\delta^{id}_{l,\mu} = \max_{j,j' \in \{1,\dots,k\}} \sum_{r \in M_{j,j'}} (A^{j'}_r - A^j_r).$$

By the result of [30] and the definition of privacy we therefore have

$$\left|\Pr[(\pi_O \parallel \hat{\pi}_n(c) \parallel \hat{\pi}_{VAI})^{(\ell)} \mapsto 1] - \Pr[(\pi_O \parallel \hat{\pi}_n(c') \parallel \hat{\pi}_{VAI})^{(\ell)} \mapsto 1]\right| \le \delta^{id}_{n,\mu} \qquad (6)$$

for every observer $\pi_O \in \Pi_O$ and all candidates $c$ and $c'$ ($c, c' \neq \mathsf{abstain}$). This means that in the ideal protocol the advantage of every observer $\pi_O \in \Pi_O$ to correctly guess whether the voter under observation voted for $c$ or $c'$ is bounded by $\delta^{id}_{n,\mu}$.

Recall that, by assumption, for all honest voters in $P^j_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$ 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 2 that for the public-key encryption scheme we require that for every public-key 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

$$\alpha^i_{j'} = \mathrm{Enc}^{r^i_{j'}}_{pk_{j'}}(...(\mathrm{Enc}^{r^i_m}_{pk_m}(m_i, n_i))...)$$

computed by an honest voter $v_i$ for $M_{j'}$ must have the same size for all honest voters. Hence, there exists a function $\eta_{j'}$ in the security parameter such that for every instance $\pi^{(\ell)}$ of $P^j_{sElect}(n, m, \mu, p, q)$ and for every honest voter $v_i$ in $\pi^{(\ell)}$ and every run of $\pi^{(\ell)}$ the size $|\alpha^i_{j'}|$ of $\alpha^i_{j'}$ is $\eta_{j'}(\ell)$. In what follows, we simply write $\eta^i_{j'} = \eta^i_{j'}(\ell)$. In order to determine $\eta_{j'}$ one can take an arbitrary candidate and an arbitrary nonce of correct size and encrypt the pair under the public keys $pk_m, ..., pk_{j'}$.

Recall that in order to prove the theorem for the protocol $P^j_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{abst})$ with the voter $v$ under observation we have to show that

$$\left|\Pr[(\hat{\pi}_v(c) \parallel \pi^*)^{(\ell)} \mapsto 1] - \Pr[(\hat{\pi}_v(c') \parallel \pi^*)^{(\ell)} \mapsto 1]\right| \qquad (7)$$

is $\delta^{id}_{l-k,\mu}$-bounded as a function of the security parameter $\ell$, for all candidates $c, c'$ ($c, c' \neq \mathsf{abstain}$) and all programs $\pi^*$ of the remaining parties such that at least $l$ voters are honest in $\pi^*$ (excluding the voter under observation $v$) and such that the adversary (the dishonest parties in $\pi^*$) is $k$-semi-honest.

We can split up the composition $\pi^*$ in its honest and its (potentially) dishonest part. Let $HV$ be the set of all honest voters (without the voter under observation) and $\hat{\pi}_{HV}$ be the composition of their honest programs. Recall that the judge $J$, the scheduler $S$, the bulletin board $B$, the voting authority $VA$, and the $j$-th mix server $M_j$ are honest. Therefore, the honest part, which we denote by $\hat{\pi}_H = \hat{\pi}_J \parallel \hat{\pi}_{VA} \parallel \hat{\pi}_B \parallel \hat{\pi}_S \parallel \hat{\pi}_{M_j} \parallel \hat{\pi}_{HV}$, consists of the honest programs $\hat{\pi}_J, \hat{\pi}_{VA} \hat{\pi}_B, \hat{\pi}_S, \hat{\pi}_{M_j}, \hat{\pi}_{HV}$ of the judge $J$, the bulletin board $B$, the scheduler $S$, the honest mix server $M_j$, and the honest voters $HV$, respectively. By $\hat{\pi}_H(c)$ we will denote the composition of all honest program including the program of the voter under observation, i.e., $\hat{\pi}_H(c) = \hat{\pi}_H \parallel \hat{\pi}_v(c)$. All remaining programs are subsumed by the adversarial process $\pi_A$. This means that we can write $\hat{\pi}_v(c) \parallel \pi^*$ as $\hat{\pi}_H(c) \parallel \pi_A$.

Recall that, by assumption, there are two restrictions imposed on the adversary $\pi_A$. First, the adversary is $k$-semi-honest. Second, the set of programs $\Pi_{AS}$ of the authentication server $AS$ consists only of those programs that respond with valid acknowledgments when honest voters cast their ballots; otherwise the programs of $AS$ can perform arbitrary (dishonest) actions. In particular, this is the case for the authentication server within $\pi_A$.

In order to prove the result, we use a sequence of games. We fix a candidate $c$ and start with Game 0 which is simply the process $\hat{\pi}_v(c) \parallel \pi^* = \hat{\pi}_H(c) \parallel \pi_A$. Step by step, we transform Game 0 into Game 4 which is the composition $\hat{\pi}_{H_4}(c) \parallel \pi_A$ for some process $\hat{\pi}_{H_4}(c)$ and the same adversarial process $\pi_A$. Game 4 will be proven indistinguishable from Game 0 from the adversary's point of view, which means that

$$\left|\mathsf{Pr}[(\hat{\pi}_H(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{H_4}(c) \parallel \pi_A)^{(\ell)} \mapsto 1]\right| \tag{8}$$

is negligible as a function of $\ell$ for a fixed candidate $c$. On the other hand, it will be straightforward to show that in Game 4 for arbitrary candidates $c$ and $c'$ ($c, c' \neq \mathsf{abstain}$)

$$\left|\mathsf{Pr}[(\hat{\pi}_{H_4}(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{H_4}(c') \parallel \pi_A)^{(\ell)} \mapsto 1]\right| \tag{9}$$

is bounded by $\delta^{id}_{l-k,\mu}$ because $\hat{\pi}_{H_4}(c)$ and $\hat{\pi}_{H_4}(c')$ use the ideal voting functionality for $l-k$ honest voters. Using the triangle inequality we can therefore deduce that

$$\left|\mathsf{Pr}[(\hat{\pi}_H(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_H(c') \parallel \pi_A)^{(\ell)} \mapsto 1]\right| \tag{10}$$

is $\delta^{id}_{l-k,\mu}$-bounded for all candidates $c$ and $c'$.

**Game 0.** In what follows we write $\hat{\pi}_{H_0}(c)$ for $\hat{\pi}_H(c)$ and consider $\hat{\pi}_{H_0}(c)$ as one atomic process (one program) and not as a composition of processes.[21] Now, Game 0 is the process $\hat{\pi}_{H_0}(c) \parallel \pi_A$. $\triangle$

In the first step, we construct Game 1 which will be proven indistinguishable from Game 0 in Claim F based on the IND-CCA2-security of the public-key encryption

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

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 $\hat{\pi}_{H_0}(c)$ in the following way in order to obtain $\hat{\pi}_{H_1}(c)$. Apart from the modifications below, $\hat{\pi}_{H_0}(c)$ and $\hat{\pi}_{H_1}(c)$ are identical.

*Ballot creation (simulated):* Recall that, in order to create her ballot $\alpha_0^i$, an honest voter $v_i$ first chooses a candidate (either $c$, if under observation, or according to $\mu$, otherwise) and a nonce $n_i$, and then encrypts the tuple under the public keys of the mix servers, starting with the public key $pk_m$ of the last mix server and then going to the public key $pk_1$ of the first mix server.

To simulate the process $\hat{\pi}_{v_i}$ of an arbitrary honest voter $v_i$, the process $\hat{\pi}_{H_1}(c)$ follows $\hat{\pi}_{v_i}$ until the encryption of $v_i$'s choice under the public key $pk_{j+1}$ of the mix server $M_{j+1}$: $\hat{\pi}_{H_1}(c)$ first chooses a candidate and a nonce as before and encrypts it with the public keys $pk_m, ..., pk_{j+1}$ of the mix servers behind the honest mix server $M_j$ to obtain $\alpha_j^i$ (which is supposed to be the output of $M_j$). Now, however, $\hat{\pi}_{H_1}(c)$ does not encrypt $\alpha_j^i$ (containing the choice) further. Instead, $\hat{\pi}_{H_1}(c)$ encrypts a random string of length $\eta_j$ under the remaining public keys $pk_j, pk_{j-1}, ..., pk_1$ to obtain the ciphertexts $\alpha_{j-1}^i, ..., \alpha_0^i$, where $\eta_j$ is defined as above. The pair $\alpha_j^i, \alpha_{j-1}^i$ is logged by $\hat{\pi}_{H_1}(c)$ for replacement later on. After that and before simulating the process $\hat{\pi}_{M_j}$ of the honest mix server $M_j$, $\hat{\pi}_{H_1}(c)$ and $\hat{\pi}_{H_0}(c)$ are identical. This means that the ciphertext $\alpha_0^i$ encrypting $0^{\eta_j}$ is supposed to fake the ballot of $v_i$.

*Honest mixing (simulated):* $\hat{\pi}_{H_1}(c)$ simulates $\hat{\pi}_{M_j}$ in the following way. Let $C_{j-1}$ 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 $\alpha_{j-1}^i$ is in $C_{j-1}$ (recall that ciphertexts can be dropped or manipulated by the adversary), $\hat{\pi}_{H_1}(c)$ adds $\alpha_j^i$ to its output $C_j$ (which is supposed to fake the output of the honest mix server $M_j$). Apart from this, $\hat{\pi}_{H_1}(c)$ follows $\hat{\pi}_{M_j}$. In particular, if the input to $M_j$ contains a ciphertext $z$ which is not logged before as $\alpha_{j-1}^i$, then this ciphertext is decrypted (using the decryption key of $M_j$) and, if successful, added to the output of $M_j$. $\triangle$

Game 1 and Game 2 are completely identical with the difference being that the simulator (i.e., $\hat{\pi}_{H_1}(c)$ in Game 1 and $\hat{\pi}_{H_2}(c)$ 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 F we will show that this can only happen with negligible probability if the adversary is $k$-semi-honest. Therefore, Game 1 and Game 2 can be proven computationally indistinguishable as stated in Claim F.

**Game 2.** The process $\hat{\pi}_{H_2}(c)$ is completely identical to $\hat{\pi}_{H_1}(c)$ with only one modification: $\hat{\pi}_{H_2}(c)$ halts if there are less than $l-k$ ciphertexts associated to the honest voters in the input $C_{j-1}$ of the (simulated) honest mix server $M_j$. In this case, $\hat{\pi}_{H_2}(c)$ halts when it is triggered the first time after the publication of $C_{j-1}$. $\triangle$

We modify $\hat{\pi}_{H_2}(c)$ 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 F.

**Game 3.** For Game 3, we modify $\hat{\pi}_{H_2}(c)$ in the following way in order to obtain $\hat{\pi}_{H_3}(c)$. Apart from the modifications below, $\hat{\pi}_{H_2}(c)$ and $\hat{\pi}_{H_3}(c)$ are identical.

*Ballot creation (simulated):* Let $v_i$ be an arbitrary honest voter. In contrast to $\hat{\pi}_{H_2}(c)$, $\hat{\pi}_{H_3}(c)$ does not pick a candidate when creating the ballot $\alpha_0^i$ and therefore does not encrypt the candidate under the public keys $pk_m, ..., pk_{j+1}$ to obtain $\alpha_j^i$. Instead, $\hat{\pi}_{H_3}(c)$ only encrypts a randomly chosen bit string of length $\eta_j$ under the first $j$ public keys $pk_j, pk_{j-1}, ..., pk_1$ in reverse order to build and publish the fake ballot $\alpha_0^i$ for the voter $v_i$ as in $\hat{\pi}_{H_2}(c)$. The pair $(\alpha_{j-1}^i, v_i)$ is logged by $\hat{\pi}_{H_3}(c)$.

*Honest mixing (simulated):* Let $C_{j-1}$ 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 $\alpha_{j-1}^i$ is in $C_{j-1}$, $\hat{\pi}_{H_3}(c)$ picks a candidate ($c$ or according to $\mu$, respectively) and encrypts it along with a randomly chosen nonce under the public keys $pk_m, ..., pk_{j+1}$ to obtain $\alpha_j^i$. Afterwards $\hat{\pi}_{H_3}(c)$ adds $\alpha_j^i$ to the output $C_j$ (which is supposed to fake the output of the mix server $M_j$). Apart from this, $\hat{\pi}_{H_3}(c)$ follows $\hat{\pi}_{H_2}(c)$. △

The only difference between Game 3 and Game 4 is that the simulator in Game 4, i.e., $\hat{\pi}_{H_3}(c)$, uses the ideal voting protocol for $l-k$ honest voters in order to receive $l-k$ 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 F both games are perfectly indistinguishable. Additionally, and this is the central idea of the proof, the advantage of the adversary to decide what the voter under observation voted for is bounded by $\delta_{l-k,\mu}^{id}$. To see this, assume that the adversary also controlled the simulator without the ideal voting protocol for $l-k$ honest voters. In this case the advantage is obviously bounded by $\delta_{l-k,\mu}^{id}$. This is Claim F.

**Game 4.** $\hat{\pi}_{H_4}$ is identical to $\hat{\pi}_{H_3}$ except for the simulation of the honest mix server $M_j$. In the honest mixing phase the process $\hat{\pi}_{H_4}$ (which is now independent of $c$) uses the ideal voting protocol (which now depends on $c$) to generate the choices of the first $l-k-1$ honest voters and the voter under observation (as described below).

*Honest mixing (simulated):* Let $C_{j-1}$ be the input to the (simulated) honest mix server $M_j$. Note that, according to Game 2, $\hat{\pi}_{H_3}$ halts if $C_{j-1}$ contains less that $l-k$ ciphertexts associated to the honest voters. This is done in $\hat{\pi}_{H_4}$ as well. Otherwise, at the beginning of the honest mixing phase, $\hat{\pi}_{H_4}$ triggers the ideal voting protocol $\hat{\pi}_{l-k}(c) \parallel \hat{\pi}_{VAI}$ (see Definition 6); we implicitly compose the program of the observer in the ideal voting protocol with the adversary $\pi_A$. The ideal voting protocol then outputs the list of permutated candidates (as plaintexts) that have been chosen by $l-k-1$ honest voters plus the voter under observation (inside the ideal voting protocol and independently of $\hat{\pi}_{H_4}$). Now, $\hat{\pi}_{H_4}$ calculates the number $\kappa$ of ciphertexts $\alpha_{j-1}^i$ associated to the honest voters in the input $C_{j-1}$ of the (simulated) honest mix server $M_j$. If $\kappa-(l-k)>0$, then $\hat{\pi}_{H_4}$ picks $\kappa-(l-k)$ candidates according to $\mu$. Afterwards, $\hat{\pi}_{H_4}$ has a list of $\kappa$ plaintexts: $l-k$ from the ideal functionality and $\kappa-(l-k)$ generated by itself. For each of these plaintexts, $\hat{\pi}_{H_4}$ generates a nonce and encrypts the vote-nonce pair under the public keys $pk_m, ..., pk_{j+1}$ as in the previous games. Afterwards, it adds them to the output $C_j$. The rest of $\hat{\pi}_{H_4}$ is identical to $\hat{\pi}_{H_3}(c)$. So, altogether Game 4 has the form $\hat{\pi}_{l-k}(c) \parallel \hat{\pi}_{VAI} \parallel \hat{\pi}_{H_4} \parallel \pi_A$. △

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 $\pi_A$ the advantage to correctly guess the candidate the voter under observation voted for is bounded by the privacy level of the ideal voting protocol for $l-k$ honest voters. This result in combination with the indistinguishability of all games allows us to derive Theorem 2.

*Claim.* Game 0 and Game 1 are computationally indistinguishable, i.e., we have that

$$\left|\mathsf{Pr}[(\hat{\pi}_{H_0}(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{H_1}(c) \parallel \pi_A)^{(\ell)} \mapsto 1]\right| \tag{11}$$

is negligible as a function of $\ell$.

*Proof.* We prove that, if $\pi_A$ can distinguish between Game 0 and Game 1 for some candidate $c$, then there exists an attacker $\pi_{A'}$ who can break the IND-CCA2-security of the public-key encryption scheme by using $\pi_A$ (see Definition 10). So, let us assume that the difference (11) is *non-negligible*.

Let $\hat{\pi}_C(b) = \hat{\pi}_C(pk_j, sk_j, b)$ be a challenger as in Definition 9, 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 $\pi_{A'}$ on the public-key encryption scheme with key pair $(pk_j, sk_j)$ such that for the adversary $\pi_A$ the process $\pi_A \parallel \pi_{A'} \parallel \hat{\pi}_C(b)$ is identical to $\hat{\pi}_{H_0}(c) \parallel \pi_A$, if $b=0$, and to $\hat{\pi}_{H_1}(c) \parallel \pi_A$, if $b=1$.

The process $\pi_{A'}$ is defined to be identical to $\hat{\pi}_{H_1}(c)$ until the encryption of the ciphertexts $\alpha^i_j$ under the public key $pk_j$. This step is modified in the following way. The attacker $\pi_{A'}$ sends two vectors to the challenger $\hat{\pi}_C(b)$: the first vector contains the ciphertexts $\alpha^i_j$ of all honest voters and the second vector contains a randomly chosen bit string of length $\eta_j$ 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, $\pi_{A'}$ follows the process $\hat{\pi}_{H_1}(c)$ with one exception explained below. In particular this means that $\pi_{A'}$ encrypts the vector of ciphertexts it has received from the challenger under the remaining public keys $pk_{j-1}, ..., pk_0$. Also for each honest voter $v_i$ whose associated ciphertext $\alpha^i_{j-1}$ is in $C_{j-1}$, $\pi_{A'}$ adds $\alpha^i_j$ to its output $C_j$. The only difference between $\hat{\pi}_{H_1}(c)$ and $\pi_{A'}$ at this point is that whenever $\hat{\pi}_{H_1}(c)$ would decrypt a ciphertext $z$ (where $z$ is in the input to $M_j$ but it is not one of the logged $\alpha^i_{j-1}$), $\pi_{A'}$ obtains the decryption of $z$ by querying the decryption oracle of the challenger.

Observe that for $b=0$, the ciphertext $\alpha^i_{j-1}$ is an encryption of $\alpha^i_j$ under $pk_j$ as in Game 0, and for $b=1$, the ciphertext $\alpha^i_{j-1}$ is an encryption of a random bit string of length $\eta_j$ under $pk_j$ as in Game 1. Also note that in neither game, and hence, also not in $\pi_A \parallel \pi_{A'} \parallel \hat{\pi}_C(b)$ honest voters blame the honest mix server $M_j$ since if $\alpha^i_{j-1}$ is in the input of $M_j$, $M_j$ adds $\alpha^i_j$ to its output. In particular, honest voters do not need to be able to provide evidence for the misbehavior of $M_j$, which in $\pi_A \parallel \pi_{A'} \parallel \hat{\pi}_C(b)$ they would not be able to do as encryption is done by the challenger. Now, it is straightforward to see that from the point of view of the adversary $\pi_A$ the process $\pi_A \parallel \pi_{A'} \parallel \hat{\pi}_C(b)$ is identical to Game 0, if $b=0$, and to Game 1, if $b=1$.

By assumption, the adversary $\pi_A$ can distinguish between Game 0 and Game 1. Defining that $\pi_{A'}$ outputs 0, if $\pi_A$ outputs 1, and 1, otherwise, the attacker $\pi_{A'}$ has a non-negligible advantage in the IND-CCA2-security game with the challenger $\hat{\pi}_C(b)$ (see Definition 10). 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.* The adversary $\pi_A$ is $k$-semi-honest in Game 1 (meaning that with overwhelming probability a run of the system does not stop before $C_m$ is published and there are at least $l-k$ vote-nonce pairs in $C_m$ chosen by honest voters).

*Proof.* By assumption, $\pi_A$ is $k$-semi-honest in Game 0. Now, the claim follows immediately from the proof of Claim F: one could modify $\pi_{A'}$ in such a way that it checks whether $\gamma'_k$ is satisfied or not (which $\pi_{A'}$ can do efficiently). □

*Claim.* The probability that in a run of the process $\hat{\pi}_{H_1}(c) \parallel \pi_A$, there are at least $l-k$ ciphertexts associated to the honest voters in the input $C_{j-1}$ of the (simulated) honest mix server $M_j$ is overwhelming.

*Proof.* From Claim F we know that with overwhelming probability in a run of $\hat{\pi}_{H_1}(c) \parallel \pi_A$ there are at least $l-k$ vote-nonce pairs in $C_m$ that have been chosen by different honest voters. We will now show that the probability (over all possible runs of the process $\hat{\pi}_{H_1}(c) \parallel \pi_A$) that there are *less* than $l-k$ ciphertexts associated to the honest voters in the input $C_{j-1}$ of the (simulated) honest mix server $M_j$ is negligible as a function of the security parameter $\ell$.

Let $r$ be an arbitrary run of the system $\hat{\pi}_{H_1}(c) \parallel \pi_A$ in which there are at least $l-k$ vote-nonce pairs of honest voters in $C_m$ while there are less than $l-k$ ciphertexts associated to the honest voters in the input of the honest mix server $C_{j-1}$. Then there exists an honest voter $\hat{\pi}_{v_i}$ (or $\hat{\pi}_v(c)$) whose associated ciphertext $\alpha^i_{j-1}$ is not in $C_{j-1}$ while her vote-nonce pair $(m_i, n_i)$ is in the final output $C_m$. Since $\alpha^i_{j-1}$ is not in $C_{j-1}$, the process $\hat{\pi}_{H_1}(c)$ does not add $\alpha^i_j$ to its output $C_j$. By the definition of Game 1, it is straightforward to see that therefore, the adversary $\pi_A$ does not receive any information about the vote-nonce pair $(m_i, n_i)$ throughout the whole run. That is, the run is independent of $n_i$.

Let $\hat{\pi}_{v_i}$ be an arbitrary honest voter. We split up the set $\Omega$ of random bit strings used to determine the runs of $\hat{\pi}_{H_1}(c) \parallel \pi_A$ into $\Omega_i$ which consists of all random coins used to determine the nonce of $\hat{\pi}_{v_i}$ and $\Omega'_i$ which consists of the remaining random coins. This means that $\Omega$ can be represented as $\Omega_i \times \Omega'_i$. Now, let $E_i$ be the event that $\alpha^i_{j-1}$ is not in $C_{j-1}$ while the vote-nonce pair of $\hat{\pi}_{v_i}$ is in the final output $C_m$ (recall that $\pi_A$ is $k$-semi-honest and thus $C_m$ is published).

Let $r$ be a run in $E_i$ and let $\omega \in \Omega$ be the random coins of this run. Let $\omega_{nonce} \in \Omega_i$ be the random coins used to determine the nonce of $\hat{\pi}_{v_i}$ in $r$ and $\omega' \in \Omega'_i$ be the remaining random coins. This means that $\langle \omega_{nonce}, \omega' \rangle$ represents $\omega$ as described above. Note that by $\omega'$ up to and including the publication of the result, the view of the adversary is completely determined and independent of $\omega_{nonce}$. Thus we have that

$$\Pr\left[E_i \mid \omega'\right] \leq \frac{n}{2^\ell}. \tag{12}$$

(Recall that $n$ is the number of voters and $\ell$ is the size of the nonces/verification codes of honest voters). Therefore, we know that

$$\Pr\left[E_i\right] = \sum_{\omega' \in \Omega'_i} \Pr\left[E_i \mid \omega'\right] \cdot \Pr\left[\omega'\right] = \sum_{\omega' \in \Omega'_i} \frac{n}{2^\ell} \cdot \Pr\left[\omega'\right] = \frac{n}{2^\ell} \tag{13}$$

holds true.

Now, let $E$ be the event that there are less than $l-k$ ciphertexts associated to the honest voters in the input $C_{j-1}$ of the (simulated) honest mix server $M_j$ in the process $\hat{\pi}_{H_1}(c) \parallel \pi_A$. Let $E'$ be the event that an adversary in a run of $\hat{\pi}_{H_1}(c) \parallel \pi_A$ is $k$-semi-honest. 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

$$\mathsf{Pr}\left[E \cap E'\right] = \mathsf{Pr}\left[\bigcup_{i=1}^{l} E_i\right] \leq \sum_{i=1}^{l} \mathsf{Pr}\left[E_i\right] = \frac{n \cdot l}{2^{\ell}}. \tag{14}$$

Hence, $\mathsf{Pr}\left[E \cap E'\right]$ is negligible. □

*Claim.* Game 1 and Game 2 are computationally indistinguishable, i.e., for each candidate $c$ we have that

$$\left|\mathsf{Pr}[(\hat{\pi}_{H_1}(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{H_2}(c) \parallel \pi_A)^{(\ell)} \mapsto 1]\right| \tag{15}$$

is negligible as a function of $\ell$.

*Proof.* This follows immediately from Claim F. □

*Claim.* Game 2 and Game 3 are perfectly indistinguishable, i.e., for each candidate $c$ we have that

$$\left|\mathsf{Pr}[(\hat{\pi}_{H_2}(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{H_3}(c) \parallel \pi_A)^{(\ell)} \mapsto 1]\right| = 0 \tag{16}$$

holds true.

*Proof.* The postponed creation of all $\alpha_j^i$ in Game 3 has no impact on the information the adversary can derive throughout the game because the ciphertexts $\alpha_j^i$ in Game 2 are not output before the honest mixing phase. Therefore, the claim holds true. □

*Claim.* Game 3 and Game 4 are perfectly indistinguishable, i.e., for each candidate $c$ we have that

$$\left|\mathsf{Pr}[(\hat{\pi}_{H_3}(c) \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{l-k}(c) \parallel \hat{\pi}_{VAI} \parallel \hat{\pi}_{H_4} \parallel \pi_A)^{(\ell)} \mapsto 1]\right| = 0 \tag{17}$$

holds true.

*Proof.* The only difference between Game 3 and Game 4 is the fact that in Game 4 $l-k$ honest choices are not generated by $\hat{\pi}_{H_4}$ but by the ideal voting protocol. But this is done in the same way. So the two games are essentially identical. □

*Claim.* For Game 4, we have that

$$\left|\mathsf{Pr}[(\hat{\pi}_{l-k}(c) \parallel \hat{\pi}_{VAI} \parallel \hat{\pi}_{H_4} \parallel \pi_A)^{(\ell)} \mapsto 1] - \mathsf{Pr}[(\hat{\pi}_{l-k}(c') \parallel \hat{\pi}_{VAI} \parallel \hat{\pi}_{H_4} \parallel \pi_A)^{(\ell)} \mapsto 1]\right|$$

is bounded by $\delta_{l-k,\mu}^{id}$ for all candidates $c$ and $c'$ ($c, c' \neq \mathsf{abstain}$).

*Proof.* This follows immediately from (6) for $\pi_O = \hat{\pi}_{H_4} \parallel \pi_A$. □

From these claims, Theorem 2 follows immediately.

# G IND-CCA2-secure public key encryption and EUF-CMA-secure signatures

## G.1 IND-CCA2 Encryption

**Definition 7 (Public-key encryption schemes).** *A public-key encryption scheme consists of a triple of algorithms $(Gen, Enc, Dec)$, where*

1. *Gen, the* key generation algorithm*, is a probabilistic algorithm that takes a security parameter $\ell$ and returns a pair $(pk, sk)$ of matching public and secret keys.*
2. *Enc, the* encryption algorithm*, is a probabilistic algorithm that takes a public key pk and a message $x \in \{0,1\}^*$ to produce a ciphertext y.*
3. *Dec, the* decryption algorithm*, is a deterministic algorithm which takes a secret key sk and a ciphertext y to produce either a message $x \in \{0,1\}^*$ or a special symbol $\perp$ to indicate that the ciphertext was invalid.*

*We require that for all $(pk, sk)$ which can be output by $Gen(1^\ell)$, for all $x \in \{0,1\}^*$, and for all y that can be output by $Enc_{pk}(x)$, we have that $Dec_{sk}(y) = x$. We also require that Gen, Enc and Dec can be computed in polynomial time.*

**Definition 8 (Encryption of vectors).** *Let $(Gen, Enc, Dec)$ be a public-key encryption scheme. Let $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$ be vectors of entries in $\{0,1\}^*$. We write*

$$Enc_{pk}(x) = (Enc_{pk}(x_1), ..., Enc_{pk}(x_n))$$
$$Dec_{sk}(y) = (Dec_{sk}(y_1), ..., Dec_{sk}(y_n))$$

*for every public key pk and every secret key sk.*

**Definition 9 (Challenger).** *Let $(Gen, Enc, Dec)$ be a public-key encryption scheme. The* challenger *C is a probabilistic polynomial-time algorithm that takes a bit b as well as a key pair $(pk, sk)$ and that serves two types of queries:*

1. *For a vector of messages y, the challenger returns the decryption of y, that is $Dec_{sk}(y)$.*
2. *For a pair of vectors of messages $(x_0, x_1)$ where both vectors have the same size and all messages at the same position in the vectors have the same length, the challenger encrypts $x_b$ under pk and returns the vector of ciphertexts, that is $Enc_{pk}(x_b)$.*

**Definition 10 (IND-CCA2-security).** *Let $(Gen, Enc, Dec)$ be a public-key encryption scheme with security parameter $\ell$ and let C be the challenger. Then the encryption scheme $(Gen, Enc, Dec)$ is* IND-CCA2-secure*, if for every polynomially bounded adversary A who never submits decryption queries for (parts of) a vector of messages y previously returned by a challenge query, we have that*

$$\mathsf{Pr}\left[(pk, sk) \leftarrow Gen(1^\ell); b' \leftarrow A^{C(1,pk,sk)}(1^\ell, pk); b' = 1\right]$$
$$-\mathsf{Pr}\left[(pk, sk) \leftarrow Gen(1^\ell); b' \leftarrow A^{C(0,pk,sk)}(1^\ell, pk); b' = 0\right]$$

*is a negligible function in $\ell$.*

### G.2 EUF-CMA Signatures

**Definition 11 (Signature schemes).** *A digital signature scheme consists of a triple of algorithms* $(Gen, Sig, Ver)$*, where*

1. *Gen, the* key generation algorithm*, is a probabilistic algorithm that takes a security parameter* $\ell$ *and returns a pair* $(sk, pk)$ *of matching secret and public keys.*
2. *Sig, the* signing algorithm*, is a (possibly) probabilistic algorithm that takes a private key sk and a message* $x \in \{0,1\}^*$ *to produce a signature* $\sigma$*.*
3. *Ver, the* verification algorithm*, is a deterministic algorithm which takes a public key pk, a message* $x \in \{0,1\}^*$ *and a signature* $\sigma$ *to produce a boolean value.*

*We require that for all* $(sk, pk)$ *which can be output by* $Gen(1^\ell)$*, for all* $x \in \{0,1\}^*$*, and for all* $\sigma$ *that can be output by* $Sig_{sk}(x)$*, we have that* $Ver_{sk}(x,\sigma) = true$*. We also require that Gen, Sig and Ver can be computed in polynomial time.*

**Definition 12 (EUF-CMA-security).** *Let* $(Gen, Sig, Ver)$ *be a signature scheme with security parameter* $\ell$*. 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* $(x,\sigma)$ *for which x has previously been signed by the oracle, we have that*

$$\mathsf{Pr}\left[(sk, pk) \leftarrow Gen(1^\ell); (x,\sigma) \leftarrow A^{Sig_{sk}(\cdot)}(1^\ell, pk); Ver_{pk}(x,\sigma) = true\right]$$

*is negligible as a function in* $\ell$*.*