sElect: A Lightweight Verifiable Remote Voting System

Ralf Küsters*, Johannes Müller*, Enrico Scapin*, Tomasz Truderung†
*University of Trier, Germany
†Polyas GmbH, Germany
e-mail: {kuesters, muellerjoh, scapin}@uni-trier.de, 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.

I. 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], [39]). 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 [36], STAR-Vote [7], and Remotegrity [41]. 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, particu-

larly 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 VIII.

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 VII). 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 VIII, 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 VIII).

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 II, 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 VIII.

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 IV, V, and VI 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.[1] In contrast, sElect demonstrates that one can achieve (a certain level of) end-to-end verifiability, as well as accountability, without universal verifiability. This is interesting from a conceptual point of view and may lead to further new applications and system designs.

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 II, we describe sElect in detail on a conceptual level. Verifiability, accountability, and privacy of sElect are then analyzed in Sections IV, V, and VI, respectively, based on the model of sElect provided in Section III. Details of our implementation of sElect are presented in Section VII, with a detailed discussion of sElect and related work provided in Section VIII. We conclude in Section IX. Full details and proofs can be found in the full version of this paper [27]; see [2] for the implementation and an online demo of sElect.

[1] As pointed out in [31], this combination does not guarantee end-to-end verifiability, though.

II. Description of sElect

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

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[2] 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 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 the full version [27] 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 VII, 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 α_{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

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

The VSD submits α_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 α_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.

[2]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.

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 IV, 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 VII. 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 V).

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 IV). 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.[3] 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 VII, 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 α_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 signature of

[3] 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.

AS on the ballot α_0^i). Otherwise, i.e., if α_0^i is in C_0, the VSD checks whether α_1^i is listed in the published result C_1 of the first mix server M_1. If C_1 contains α_1^i, the VSD vsd_i checks whether α_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 α_j^i (such a mix server has to exist), the VSD anonymously sends (j, α_j^i, r_j^i) to the bulletin board B. This triple demonstrates that M_j misbehaved: the encryption of α_j^i under pk_j with randomness r_j^i yields α_{j-1}^i, and hence, since α_{j-1}^i is in the input to M_j, α_j^i should have been in M_j's output, which, however, is not the case. The reason that an anonymous channel is necessary to submit the triple is the fact that it might reveal how the voter voted, for example, if M_j is the last mix server and thus α_j^i contains the voter's 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.

III. Modeling

In this section, we formally model the sElect voting protocol, with full details provided in the full version [27]. 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 [29], [31]. 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 π as $\pi = p_1 \parallel \cdots \parallel p_l$, where $p_1 \ldots, p_l$ are programs. If π_1 and π_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 π where all programs are given the security parameter ℓ is denoted by $\pi^{(\ell)}$. The processes we consider are such that the length of a run is always polynomially bounded in ℓ. Clearly, a run is uniquely determined by the random coins used by the programs in π.

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

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

Negligible, overwhelming, δ-bounded. As usual, a function f from the natural numbers to the interval $[0,1]$ is *negligible* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \frac{1}{\ell^c}$ for all $\ell > \ell_0$. The function f is *overwhelming* if the function $1 - f$ is negligible. A function f is λ-*bounded* if, for every $c > 0$ there exists ℓ_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 $P_{sElect} = P_{sElect}(n, m, \mu, p_{voter}^{verif}, p_{vsd}^{verif}, p_{abst}^{verif})$ 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 μ we denote a probability distribution on the set of candidates/choices, including abstention. An honest voter makes her choice according to this distribution.[4] This choice is provided to her VSD and is called the *actual choice* of the voter. By $p_{voter}^{verif} \in [0,1]$ we denote the probability that an honest voter who does not abstain from voting verifies the result, i.e., performs the voter-based verification procedure. By $p_{vsd}^{verif} \in [0,1]$ we denote the probability that an honest VSD of a voter who does not abstain from voting is triggered to verify the result. By $p_{abst}^{verif} \in [0,1]$ we denote the probability that an honest voter who abstains from voting verifies that her name is not listed in the list *LN* output by the authentication server. Note that the set of valid choices (candidates) is implicitly

[4] 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.

given by μ. 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 II, i.e., the bulletin board B, n voters $v_1,\ldots,v_n$, n VSDs $vsd_1,\ldots,vsd_n$, the authentication servers AS, m mix servers $M_1,\ldots,M_m$, 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 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 II. We assume that the scheduler and the bulletin board are honest. Technically, this means that the set of programs Π_{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 Π_{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 VIII).

IV. 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 [29]. 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 VIII).

For brevity of presentation, we state a simple domain specific instantiation of the general end-to-end verifiability definition in [29]. This definition is centered around the *goal* that a system is supposed to achieve. Informally speaking, according to [29], 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.

A. 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 III).

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 γ_k for voting. First, recall from Section III that an honest voter[5] 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.

Definition 1 (Goal γ_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 γ_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 γ_k is not satisfied in r.

The above definition says that in a run r the goal γ_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 $k = 0$, γ_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 [30] for more discussion on γ_k.

B. Definition of Verifiability

As mentioned at the end of Section II, 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 II; 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 δ.

Definition 2 (Verifiability): An e-voting protocol P provides δ-verifiability with tolerance k if, for every instance π of P, the probability that in a run r of π

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

(b) all voters accept r

is δ-bounded (i.e., bounded by δ plus some negligible function, as defined in Section III).

C. 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

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

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* [32].

We now state the verifiability level provided by sElect. Recall that p^{verif}_{voter} denotes the probability that an honest voter who does not abstain from voting verifies the final result, and that p^{verif}_{abst} 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 $P_{sElect}(n, m, \mu, p^{verif}_{voter}, p^{verif}_{vsd}, p^{verif}_{abst})$ provides $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$-verifiability w.r.t. the goal γ_k, where

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

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

The formal proof of Theorem 1 is provided in the full version [27]. The intuition behind the definition of $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$ 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 γ_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, $\delta^k(p^{verif}_{voter}, p^{verif}_{abst})$ 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.

V. 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 [29], which we instantiate for sElect. The detailed formal accountability result and full proofs are given in the full version [27]. Here, due to space limitations, 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.

VI. 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 II, 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 Π_{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 Π_{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.

A. 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 IV). 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 VI-C). 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 VI-C 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 γ'_k of runs of P^j_{sElect} which is defined similarly to γ_k (see Section IV-A) 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 γ'_k.

Formally, γ'_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.[6] We say that γ'_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 γ'_k is not satisfied in r.

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 γ'_k is satisfied in this run.[7] An adversary (of an instance π of P^j_{sElect}) is *k-semi-honest* if it is k-semi-honest with overwhelming probability (over the set of runs of π).

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

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

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

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

[7] 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.

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 the full version [27], 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 $1-(1-p_{voter}^{verif})^{k+1}$. This risk is enormous even for quite modest k and realistic probabilities p_{voter}^{verif} (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.

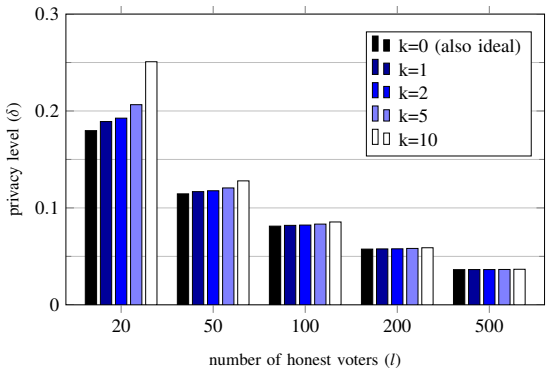

Fig. 1: Privacy level δ_{l-k} for sElect with k-semi-honest adversary, for different number of honest voters l and different k. The honest voters vote for two candidates, with probabilities 0.4 and 0.6. Note that the case $k=0$ also equals the ideal case.

B. Definition of Privacy

In our analysis of the sElect system, we use the definition of privacy for e-voting protocols proposed in [31], 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 [31] 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 [31], 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 III 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 π^* is the composition of programs of the remaining parties. In the case of sElect, π^* 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 π^*, writes the output 1 on some dedicated channel in a run of $\hat{\pi}_v(c) \parallel \pi^*$ with security parameter ℓ 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 δ-privacy w.r.t. k-semi-honest adversaries*, if

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

is δ-bounded as a function of the security parameter ℓ, for all candidates c, c' ($c, c' \neq \mathsf{abstain}$) and all programs π^* of the remaining parties such that at least l voters are honest in π^* (excluding the voter under observation v) and such that the adversary (the dishonest parties in π^*) 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 δ typically depends on the number l of honest voters, privacy is formulated w.r.t. this number. Note that a smaller δ 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, δ 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 $\delta_{l,\mu}^{id}$, where l is the number of honest voters and μ the probability distribution used by the honest voters to determine their choices (see the full version [27] for an explanation of how $\delta_{l,\mu}^{id}$ is calculated).

C. Analysis

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 δ-privacy is provided for a δ 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 μ the probability of abstention is zero.

We have the following formal privacy result for sElect. The proof is provided in the full version [27], 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 VI-B.

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 VII), 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.

VII. Implementation of sElect

In this section, we shortly describe our prototypical implementation of sElect. A more detailed overview is given in the full version [27]. 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 VIII). 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 II within the browser and without further interaction with the voting booth server.[8]

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 the full version [27] 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 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 II (see also Section V). 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 IV, V, and VI. 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

[8]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.

tend to check the result and trigger the automated verification procedure.

VIII. Related Work and Discussion

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

A. 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 [38].

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.

B. 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 V).

Obviously, for fully automated verification we need to assume that (most of) the VSDs can be trusted. Recall from Section VII 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.[9] Also, as mentioned in Section VII, 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 IV).

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, as is the case for all other remote e-voting systems. So, other systems should profit from this approach as well.[10]

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 IV.

On the positive side, voter-based verification provides a quite good level of verifiability, with the main problem being clashes (as discussed in Section IV-C). 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

[9]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]).

[10]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.

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.[11]

On the negative side, verification codes could be easily misused for coercion. A voter could (be forced to) provide a coercer with her verification code *before* the election result is published, and hence, once the result is published, a coercer can see how the voter voted.[12]

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 II). 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.[13] 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 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

[11] 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], [34], [35], [40]). 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.

[12] In very recent work, a mitigation for this problem has been considered [37], but this approach assumes, among others, a public-key infrastructure for all voters.

[13] 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.

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.

IX. 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, Johannes Müller, Enrico Scapin and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. Technical Report 2016/438, Cryptology ePrint Archive, 2016. http://eprint.iacr.org/2016/438.

[28] 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.

[29] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *CCS 2010*, pages 526–535. ACM, 2010.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] P. Y. A. Ryan, P. B. Roenne, and V. Iovino. Selene: Voting with Transparent Verifiability and Coercion-Mitigation. Cryptology ePrint Archive, Report 2015/1105.

[38] B. Schneier Applied Cryptography. John Wiley& sons, New York, 1996.

[39] 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.

[40] J. Weber, U. Hengartner Usability Study of the Open Audit Voting System Helios. http://www.jannaweber.com/wp-content/uploads/2009/09/858Helios.pdf.

[41] 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.