Cast-as-Intended Verification in Electronic Elections Based on Oblivious Transfer

Rolf Haenni(✉), Reto E. Koenig, and Eric Dubuis

Bern University of Applied Sciences, CH-2501 Biel, Switzerland
{rolf.haenni,reto.koenig,eric.dubuis}@bfh.ch

Abstract. In this paper, we propose a new method for cast-as-intended verification in remote electronic voting. We consider a setting, in which voters receive personalized verification code sheets from the authorities over a secure channel. If the codes displayed after submitting a ballot correspond to the codes printed on the code sheet, a correct ballot must have been submitted with high probability. Our approach for generating such codes and transferring them to the voter is based on an existing oblivious transfer protocol. Compared to existing cast-as-intended verification methods, less cryptographic keys are involved and weaker trust and infrastructure assumptions are required. This reduces the complexity of the process and improves the performance of certain tasks. By looking at cast-as-intended verification from the perspective of an oblivious transfer, our approach also contributes to a better understanding of the problem and relates it to a well-studied cryptographic area of research.

1 Introduction

In remote electronic voting, voters may not always have access to a trustworthy platform for creating and casting the ballot. Malware on such a platform may take control over the vote casting process, for example by submitting a ballot containing a vote different from the voter's intention or by not casting a ballot at all. Without any counter-measures, such attacks are difficult to detect and may remain unnoticed even by a large number of affected voters. Since the correct outcome of an election is of great significance for the whole electorate, every infected computer becomes inevitably a problem for everybody. This so-called *secure platform problem* is one of the most critical and challenging obstacles in remote electronic voting [SV12].

Malware attacks against remote electronic voting may aim at violating either the secrecy or the integrity of the vote (or both). Full protection against both types of attacks is very hard to achieve. Some approaches suggest using an out-of-band channel such as regular postal mail as a trust anchor, over which additional information is transmitted securely to the voters. In this paper, we consider a setting, in which each voter receives a *verification code sheet* from the authorities over such a trusted channel. After submitting the ballot, codes for the chosen candidates are displayed by the voting application and voters are instructed to check if the displayed codes match with the codes printed on the verification

R. Krimmer et al. (Eds.): E-Vote-ID 2016, LNCS 10141, pp. 73–91, 2017.
DOI: 10.1007/978-3-319-52240-1_5

code sheet. Matching codes imply with high probability that a correct ballot has been submitted. This step—called *cast-as-intended verification*—is an effective counter-measure against integrity attacks by malware on the voting platform, but obviously not against privacy attacks. Nevertheless, countries such as Norway or Switzerland have approved this as a sufficient solution for conducting elections over the Internet [GB12, BK113c].

1.1 Related Work

The idea of printing verification code sheets and distributing them over a trusted channel to the voters has first been proposed for the Norwegian Internet voting projects *eValg2011* and *eValg2013* [GB12]. From a technical point of view, the cryptographic protocols for the offline generation of the verification code sheets and the online generation of corresponding *return codes* for the chosen candidates have changed slightly in the course of time [Gjø10, Gjø11, Lip11, PG11, PG12], but the general underlying idea remained the same. Upon receiving one or multiple encrypted votes from a voter, two non-colluding servers conduct a series of cryptographic computations to remove the encryption randomizations in such a way that the plaintext votes are not disclosed. For this mechanism to work, the two servers must hold shares of the private key, under which the votes are encrypted. The return codes are then derived from the resulting deterministic values (the same deterministic values have been computed during the election preparation phase to enable the printing of the verification code sheets) and delivered over a separate channel to the voters' mobile phones. In case of non-matching return codes, voters are instructed to submit another ballot from a different platform. The separate channel for delivering the return codes is necessary to prevent the malware-infected voting application from learning the return codes when multiple ballots are submitted by the same voter.

A similar approach has been proposed for the voting system in the canton of Neuchâtel in Switzerland [GGP15]. In the Swiss context, vote updating by submitting multiple ballots is explicitly prohibited. This has two important consequences for the voting process. First, sending the return codes to the voting application is no longer a threat, even if malware has taken full control over the voting process. Second, since voters cannot re-submit the ballot from a different platform in case of non-matching return codes, ballots can only be accepted after receiving a correct *confirmation code* from the voter. In such a case, the server responds by displaying a *finalization code* to the voter for inspection.[1] Both the confirmation and the finalization code are printed on the verification code sheet along with the return codes. In the *Neuchâtel protocol* as presented in [GGP15], a matching finalization code implies that the vote has been cast as intended

[1] This extended vote casting process is approved by the Swiss Federal Chancellery as a possible solution for the secure platform problem [BK113a, Appendix 7]. If there is a mismatch between any of the return codes, voters are instructed to abort the online voting process and to submit a paper ballot. In case of mismatched finalization codes, voters are instructed to contact the election administration for an investigation.

by the voting application and recorded as cast by the server. Compared to the Norwegian protocol, the main technical difference is that voters participate in the generation of the return codes. For this, they receive a private key during the registration phase. This key replaces one of the two server-side key shares.

A very different protocol for cast-as-intended verification has been proposed in [HLv10]. To the best of our knowledge, it is the first and only such protocol based on *oblivious transfer* (OT), but it has never been implemented in practice. The idea is to transmit the return codes to the voters via a third party (the proxy) using the 1-*out-of-n proxy oblivious transfer* (POT) protocol from [AIR01]. The choice of using this particular POT protocol has multiple reasons, but most importantly, it enables voters to prove, in zero-knowledge, that the POT query and the encrypted vote contain identical plaintexts. To prove the validity of the encrypted votes, non-interactive *range proofs* are added to the ballots. The protocol is designed for the simple case where voters choose a single candidate from a set of n candidates. Multiple instances of the protocol can be executed in parallel to support general k-out-of-n limited votes, but the protocol is very inefficient for such general cases.

1.2 Contribution and Paper Overview

This paper contains two principal contributions. First, we introduce a new method for cast-as-intended verification, in which the return codes for k candidates are transmitted by an efficient k-out-of-n oblivious transfer [CT05]. This particular protocol requires no additional cryptographic keys and imposes no restrictions with regard to the space of messages that can be transferred. As a consequence, generating the return codes during the preparation of an election and transferring them to the voters during vote casting become two completely independent processes. We provide a description of a cryptographic voting protocol in Sect. 3, which shows how the query for the oblivious transfer can be linked in a natural way to the encrypted vote. Details about the cryptographic setting and the oblivious transfer protocol are given in Sect. 2.

Second, we propose a new technique to guarantee the validity of an encrypted vote without generating expensive zero-knowledge proofs. For this, we derive the return codes from random points of a random polynomial $p(x) \in_R \mathbb{Z}_p[x]$ of degree $k-1$. This implies that receiving k correct points from the oblivious transfer is sufficient to interpolate the polynomial, whereas receiving $k-1$ or less points does not provide any information about any other point on the polynomial. As a consequence, provided that p is large enough, knowing the polynomial $p(x)$ for a given verification code sheet entails with high probability that both the original OT query and the encrypted vote contain a valid set of candidates. This allows us to avoid expensive zero-knowledge proofs for proving the validity of the encrypted votes. The details of this technique are also included in the protocol description of Sect. 3. In Sect. 4, we discuss the security properties and performance of our protocol and compare it to existing work. We conclude the paper in Sect. 5.

2 Cryptographic Preliminaries

Let $(\mathcal{G}, \cdot, ^{-1}, 1)$ be a cyclic group of prime order q, for which the decisional Diffie-Hellman (DDH) assumption is believed to hold. Since q is prime, every element $x \in \mathcal{G} \setminus \{1\}$ is a generator. At the moment, we do not restrict ourselves to a particular group, but at some point, we will assume that $\mathcal{G}$ is identical to the set $\mathbb{G}_q \subset \mathbb{Z}_p^*$ of quadratic residues modulo a safe prime $p = 2q + 1$.

2.1 Oblivious Transfer

An oblivious transfer is the execution of a protocol between two parties called *sender* and *receiver*. In a k-out-of-n oblivious transfer, denoted by OT_n^k, the sender holds a list $\mathbf{m} = (m_1, \ldots, m_n)$ of messages $m_i \in \{0,1\}^\ell$, of which $k \leq n$ can be selected by the receiver. The selected messages are transferred to the receiver such that the sender remains oblivious about the receiver's selections and that the receiver learns nothing about the $n - k$ other messages. Let $\mathbf{s} = (s_1, \ldots, s_k)$ denote the k selections $s_j \in \{1, \ldots, n\}$ of the receiver and $\mathbf{m_s} = (m_{s_1}, \ldots, m_{s_k})$ the k messages to transfer. In the simplest possible case of a two-round protocol, the receiver sends a randomized query $Q \leftarrow \mathsf{Query}(\mathbf{s}, r)$ of size $O(k)$ to the sender, the sender replies with a response $R \leftarrow \mathsf{Response}(Q, \mathbf{m})$ of size $O(n)$, and the receiver obtains $\mathbf{m_s} \leftarrow \mathsf{Open}(R, r)$ by removing the randomization r from R. For the correctness of the protocol, $\mathsf{Open}(\mathsf{Response}(\mathsf{Query}(\mathbf{s}, r), \mathbf{m}), r) = \mathbf{m_s}$ must hold for all possible values of $\mathbf{m}$, $\mathbf{s}$, and r. If a triple $(\mathsf{Query}, \mathsf{Response}, \mathsf{Open})$ of such algorithms satisfies this property, we call it a *(two-round)* OT_n^k*-scheme*.

An OT_n^k-scheme is called *secure*, if the three algorithms guarantee both *receiver privacy* and *sender privacy*. Usually, receiver privacy is defined in terms of indistinguishability of two selections $\mathbf{s}_1$ and $\mathbf{s}_2$ relative to corresponding queries Q_1 and Q_2, whereas sender privacy is defined in terms of indistinguishable transcripts obtained from executing the real and the ideal protocols in the presence of a malicious receiver (called *simulator*). In the ideal protocol, $\mathbf{s}$ and $\mathbf{m}$ are sent to an incorruptible trusted third party, which forwards $\mathbf{m_s}$ to the simulator.

There are many general ways of constructing OT_n^k-schemes, for example on the basis of less complex OT_n^1 or OT_2^1-schemes, but such general constructions are usually not very efficient. In this paper, we propose to use the second OT_n^k-scheme presented in [CT05], which satisfies our requirements almost perfectly.[2] There are several public parameters: a description of a group $\mathcal{G}$ of prime order q, a generator $g \in \mathcal{G} \setminus \{1\}$, an encoding $\Gamma : \{1, \ldots, n\} \to \mathcal{G}$ of the possible selections into $\mathcal{G}$, and a collision-resistant hash function $H_\ell : \{0,1\}^* \to \{0,1\}^\ell$ with output length ℓ. In Fig. 1, we provide a detailed formal description of the protocol. The query Q is a vector $\mathbf{a} \in \mathcal{G}^k$ of length k and the response R is a tuple $(\mathbf{b}, \mathbf{c}, d)$

[2] The modified protocol as presented in [CT08] is slightly more efficient, but it fits less into the particular context of this paper.

consisting of a vector $\mathbf{b} \in \mathcal{G}^k$ of length k, a vector $\mathbf{c} \in (\{0,1\}^\ell)^n$ of length n, and a single value $d \in \mathcal{G}$. Calls of the algorithms will therefore be denoted by

$$\begin{aligned} \mathbf{a} &\leftarrow \mathsf{Query}(\mathbf{s}, \mathbf{r}), \\ (\mathbf{b}, \mathbf{c}, d) &\leftarrow \mathsf{Response}(\mathbf{a}, \mathbf{m}, s), \\ \mathbf{m_s} &\leftarrow \mathsf{Open}(\mathbf{b}, \mathbf{c}, d, \mathbf{r}), \end{aligned}$$

where $\mathbf{r} = (r_1, \ldots, r_k) \in_R \mathbb{Z}_q^k$ is the vector of random values used in computing the query and $s \in_R \mathbb{Z}_q$ an additional random value used in computing the response. Both Query and Open require k fixed-base exponentiations in $\mathcal{G}$, whereas $\mathsf{Response}$ requires $n + k + 1$ fixed-exponent exponentiations in $\mathcal{G}$. Note that among the $2k$ exponentiations of the receiver, k can be pre-computed, and among the $n + k + 1$ exponentiations of the sender, $n + 1$ can be pre-computed. Therefore, only k online exponentiations remain for both the receiver and the sender, i.e., the protocol is very efficient in terms of computation and communication costs. In the random oracle model, the scheme is provably secure against a malicious receiver and a semi-honest sender.[3] Receiver privacy is unconditional and sender privacy is computational under the *chosen-target computational Diffie-Hellman* (CT-CDH) assumption, which is a weaker assumption than standard CDH [Bol03].

2.2 ElGamal Encryption and Extended Pedersen Commitments

In the case of the ElGamal encryption scheme, a group $\mathcal{G}$ of prime order q and a generator $g \in \mathcal{G} \setminus \{1\}$ are usually fixed as public parameters. If this is the case, the scheme consists of the following three algorithms: (1) a randomized key generation algorithm $(sk, pk) \leftarrow \mathsf{KeyGen}()$, which picks $sk \in_R \mathbb{Z}_q$ uniformly at random and computes $pk = g^{sk}$; (2) a randomized encryption algorithm $e \leftarrow \mathsf{Enc}_{pk}(m)$, which picks $r \in_R \mathbb{Z}_q$ uniformly at random and computes $e = (m \cdot pk^r, g^r)$ for a given plaintext $m \in \mathcal{G}$; (3) a deterministic decryption algorithm $m \leftarrow \mathsf{Dec}_{sk}(e)$, which computes $m = a \cdot b^{-sk}$ for a given ciphertext $e = (a, b) \in \mathcal{G} \times \mathcal{G}$. It is easy to verify that $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m$ holds for all $m \in \mathcal{G}$ and all key pairs $(sk, pk) \in \mathbb{Z}_q \times \mathcal{G}$. The ElGamal encryption scheme is provably IND-CPA secure under the decisional Diffie-Hellman assumption.

In an (extended) Pedersen commitment scheme, the public parameters are a group $\mathcal{G}$ of prime order q and independent generators $g, h_1, \ldots, h_s \in \mathcal{G} \setminus \{1\}$. The scheme consists of two deterministic algorithms, one for computing a commitment $c = g^r h_1^{m_1} \cdots h_s^{m_s} \in \mathcal{G}$ to s messages $m_i \in \mathbb{Z}_q$ with randomization $r \in_R \mathbb{Z}_q$, and one for checking the validity of a commitment c when $m_1, \ldots, m_s$ and r are revealed. We denote respective algorithms by $c \leftarrow \mathsf{Commit}(m_1, \ldots, m_s, r)$

[3] In the voting protocol presented in Sect. 3, which uses this OT_n^k-scheme to transfer return codes obliviously from the authorities to the voter, sender privacy is only required during vote casting. By revealing all n return codes at the end of the vote casting process, any attempt by malicious authorities to transfer incorrect return codes will be detected.

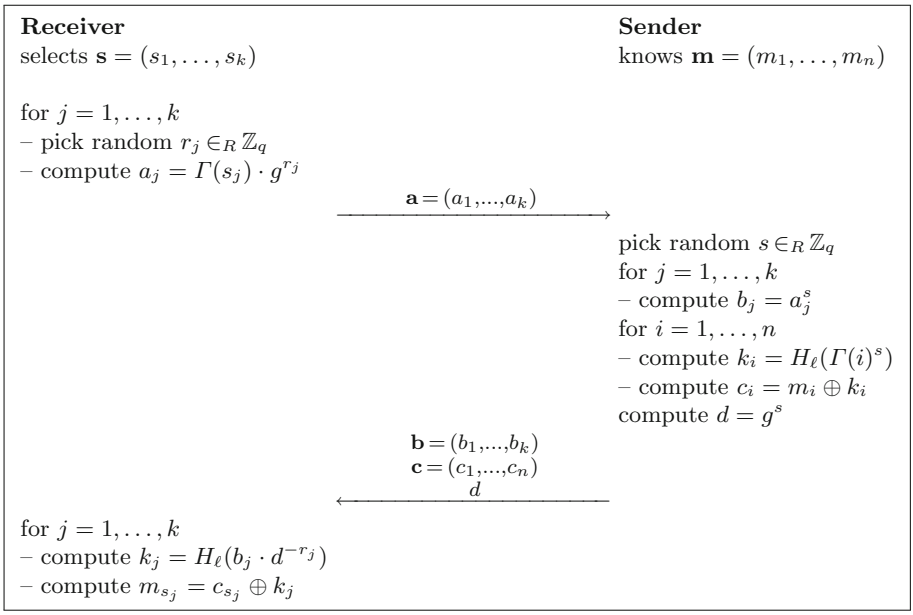

Fig. 1. Two-round OT_n^k-scheme for malicious receiver, where $\mathcal{G}$ is a group of prime order q, $g \in \mathcal{G} \setminus \{1\}$ a generator of $\mathcal{G}$, $\Gamma : \{1, \ldots, n\} \to \mathcal{G}$ an encoding of the selections into $\mathcal{G}$, and $H_\ell : \{0,1\}^* \to \{0,1\}^\ell$ a collision-resistant hash function with output length ℓ.

and $d \leftarrow \mathsf{Decommit}(c, m_1, \ldots, m_s, r)$ for $d \in \{0,1\}$. The Pedersen commitment scheme is perfectly hiding and computationally binding under the DL assumption.

2.3 Non-interactive Zero-Knowledge Proofs

Non-interactive zero-knowledge proofs of knowledge are important building blocks in cryptographic protocol design. In a non-interactive *preimage proof* $\mathsf{NIZKP}[(x) : y = \phi(x)]$ for a one-way group homomorphism $\phi : X \to Y$, the prover proves knowledge of a secret preimage $x = \phi^{-1}(y) \in X$ for a public value $y \in Y$ [Mau09]. The most common construction of a non-interactive preimage proof results from combining the Σ-protocol with the Fiat-Shamir heuristic. Proofs constructed in this way are perfect zero-knowledge in the random oracle model. In practice, the random oracle is implemented with a collision-resistant hash function H.

Generating a preimage proof $(t, c, s) \leftarrow \mathsf{GenNIZKP}_\phi(x, y)$ consists of picking a random value $w \in_R X$ and computing a commitment $t = \phi(w) \in Y$, a challenge $c = H(t, y) \in [0, c_{\max}]$, and a response $s = w + c \cdot x \in X$. Verifying a proof includes checking $c = H(t, y)$ and $\phi(s) = t \times y^c$. Sometimes, the hash function is called with an additional public input z. We denote the inclusion of such an additional

input by $(t, c, s) \leftarrow \mathsf{GenNIZKP}_{\phi}(x, y, z)$ for commitments $c = H(t, y, z)$. This technique, which ties z and (t, c, s) together, is a common practice to prevent copying proofs from one context to another. The verification of a given proof $\pi = (t, c, s)$ is denoted by $v \leftarrow \mathsf{VerifyNIZKP}_{\phi}(\pi, y, z)$ for $v \in \{0, 1\}$.

An example of a preimage proof results from the ElGamal encryption scheme. The goal of $(t, c, s) \leftarrow \mathsf{GenNIZKP}_{\mathsf{Enc}_{pk}}((m, r), (a, b), z)$ is to prove knowledge of the plaintext m and the randomization r for a given ElGamal ciphertext (a, b) and an additional public input z. Here we understand $\mathsf{Enc}_{pk}(m, r)$ as a deterministic algorithm with two arguments rather than a randomized algorithm $\mathsf{Enc}_{pk}(m)$ with one argument. Since Enc_{pk} is a homomorphism from $\mathcal{G} \times \mathbb{Z}_q$ to $\mathcal{G} \times \mathcal{G}$, both the commitment $t = (t_1, t_2)$ and the response $s = (s_1, s_2)$ are pairs of values. Generating the proof requires two and verifying the proof four exponentiations in $\mathcal{G}$. We will use this proof in the next section.

3 Cryptographic Voting Protocol

The protocol as presented in this section is designed for elections in which submitting multiple ballots is prohibited. Therefore, we assume that someone's right to vote electronically extinguishes with the first submitted ballot. If the vote casting process fails at some point, we assume that voters have an alternative vote casting channel such as postal mail or a local polling station. Note that this scenario corresponds exactly to the particular situation in Switzerland, where postal mail is the most common voting channel and where vote buying and coercion is only a minor security concern. To strengthen the compatibility with the political and legal context in Switzerland, we try to follow the existing technical recommendations as precisely as possible [BK113a, BK113b, BK113c].

3.1 General Setting

The set of *voters* and a small number of *authorities* are the principal parties involved in our protocol. They communicate over different communication channels. To set up an election, the protocol requires a secure channel from the authorities to the voters for the distribution of the verification code sheets. In a real-world setting, like the one described in [BK113a], this channel is implemented by a trusted printing office and a trusted postal service, They print the verification code sheets and deliver them to the voters. Furthermore, a broadcast channel with memory—in the form of a robust append-only *bulletin board*—is needed for collecting the submitted ballots and other election data. We assume that the authorities have their own designated areas on the bulletin board, which they can access for example by signing their messages with a private key. Finally, to emphasize our focus on cast-as-intended verification, we make a distinction between voters and the machines they use for vote casting. We call such a machine *voting platform* and assume that voters can communicate with their voting platform in a secure way (but obviously with limited bandwidth).

Candidate List. We consider elections in which voters can vote for exactly k different candidates from a set $\mathcal{C} = \{c_1, \ldots, c_n\}$ of $n \geq 2$ candidates, i.e., no candidate can be selected more than once. Note that this setting is less restrictive than it appears, because $\mathcal{C}$ may contain up to k "blank candidates" to allow votes for less than k real candidates. Similarly, $\mathcal{C}$ may contain multiple values for each real candidate to allow more than one vote per candidate. We will always refer to the elements of $\mathcal{C}$ as *candidates*, but they could as well be parties or any other type of election options. In the simplest case of a yes/no-referendum, we have either $\mathcal{C} = \{\texttt{yes}, \texttt{no}\}$ or $\mathcal{C} = \{\texttt{yes}, \texttt{no}, \texttt{blank}\}$, depending on whether blank votes are allowed or not. We assume that $\mathcal{C}$ is defined and published by the election administration prior to an election, so that it is known to everyone.

Verification Code Sheets. If the electorate consists of N eligible voters, we suppose that exactly N verification code sheets are printed, one for each eligible voter. Without loss of generality, we identify both voters and verification code sheets by corresponding indices $i \in \{1, \ldots, N\}$ and assume that code sheet i is sent to voter i prior to an election. Code sheet i contains the list $\mathcal{C}$ of candidates along with corresponding *return codes* $R_{ij} \in \{0,1\}^r$ for each candidate $c_j \in \mathcal{C}$. It also contains a unique *code sheet identifier* ID_i, a *voting code* $V_i \in \{0,1\}^v$, a *confirmation code* $C_i \in \{0,1\}^c$, and a *finalization code* $F_i \in \{0,1\}^f$. The information printed on code sheet i is therefore a tuple

$$(ID_i, V_i, C_i, F_i, \{(c_j, R_{ij})\}_{j=1}^{n}).$$

For improved usability, we assume that return codes are printed using $r' = \lceil \frac{r}{\log |A|} \rceil$ characters from an alphabet A, for example $A = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}\}$. The same holds for the voting, confirmation, and finalization codes. To detect mistyped voting or confirmation codes, we propose the inclusion of checksums.

Voter Authentication. In the remaining of this paper, we assume that someone's right to vote is identical to possessing a valid verification code sheet. With this assumption, we do not disregard the necessity of using additional voter authentication mechanisms based on passwords, biometrics, digital certificates, or physical presence in person, but we do not explicitly include this aspect in our discussion. In other words, we assume that the voter authentication problem is solved, but that eligible voters still require a valid verification code sheet for casting a vote. This implies that the codes printed on a given code sheet must remain secret, especially the voting code V_i and the confirmation code C_i, which the voter enters during vote casting to prove possession of a valid code sheet. These codes should therefore be protected by physical means such as a scratchcard or invisible ink. Note that we do not specify whether code sheets are personal or impersonal, i.e., whether they are tied to a particular voter or not. This aspect is not relevant in this paper.

3.2 Adversary Model and Trust Assumptions

We assume that the general adversarial goal is to break the integrity or secrecy of the votes, but not to influence the election outcome via bribery or coercion. We consider *covert adversaries*, which may arbitrarily interfere with the voting process or deviate from the protocol specification to reach their goals, but only if such attempts are likely to remain undetected [AL10]. Voters and authorities are potential covert adversaries, as well as any external party. This includes adversaries trying to spread dedicated malware to gain control over the voting platforms. For preparing and conducting an election, we assume that a threshold number of non-colluding authorities is available.

All parties are polynomially bounded and thus incapable of solving supposedly hard problems such as the DDH problem or breaking cryptographic primitives such as contemporary hash functions. This implies that adversaries cannot efficiently decrypt ElGamal ciphertexts or generate valid non-interactive zero-knowledge proofs without knowing the secret inputs.

3.3 Detailed Protocol Description

The subsequent description of the cryptographic voting protocol is focused on our new mechanism for cast-as-intended verification, which affects mainly the election preparation and the vote casting phase of the protocol, but not the tallying phase. We are therefore not discussing all the necessary details of the operations executed by the authorities to determine the election result from the list of submitted ballots. This part of an electronic election system is well-documented in the literature. However, we stress that defining an appropriate cryptographic protocol for the tallying phase is crucial for protecting the system against corrupt authorities.

To further simplify the presentation of the protocol, we will look at the group of authorities as a single party called *authorities*. Let $(sk, pk) \leftarrow \mathsf{KeyGen}()$ be their ElGamal key pair, which in reality will be generated in a distributed manner and such that sk is threshold shared among the authorities, for example using the protocol of [Ped91]. We assume that pk is publicly known. In Sect. 3.4, the case of multiple authorities will be discussed in further detail.

Another simplification is to fix the group $\mathbb{G}_q \subseteq \mathbb{Z}_p^*$ of quadratic residues modulo a safe prime $p = 2q + 1$ as the common group for all the cryptographic operations used in this paper. We assume that p (which determines $\mathbb{G}_q$) and independent generators $g, h_1, h_2, h_3, h_4 \in \mathbb{G}_q \setminus \{1\}$ are publicly known. Other public parameters are a second prime number $p' \leq q$, the bit lengths v, c, f, r of the voting, confirmation, finalization, and return codes, respectively, collision-resistant hash functions $H_r : \{0,1\}^* \to \{0,1\}^r$, $H_f : \{0,1\}^* \to \{0,1\}^f$, and $H_\ell : \{0,1\}^* \to \{0,1\}^\ell$ for $\ell = 2 \cdot \lceil \log p' \rceil$, and the list $\mathcal{C} = \{c_1, \ldots, c_n\}$ of candidates.

Election Preparation. As shown by the diagram depicted in Fig. 2, the election preparation consists of two tasks executed by the authorities. They first

generate the N verification code sheets and transmit them to the voters. In the second step, they publish commitments to the values contained in the code sheets on the public bulletin board. Under the assumption that possessing a verification code sheet implies eligibility, this list of commitments can be seen as the electoral roll.

To generate verification code sheet i, the authorities pick a random polynomial $p_i(x) = \sum_{j=0}^{k-1} a_{ij}x^j$ of degree $k-1$ (i.e., $a_{i,k-1} \neq 0$) from the set $\mathbb{Z}_{p'}[x]$ of all such polynomials over the field $\mathbb{Z}_{p'}$ of integers modulo p'. Then they pick n distinct random integers $x_{ij} \in_R \mathbb{Z}_{p'}$, $1 \leq j \leq n$, and compute corresponding points $P_{ij} = (x_{ij}, p_i(x_{ij}))$ on the polynomial. The hash values $R_{ij} = H_r(P_{ij})$ of these points are the return codes for the candidates. The reason for selecting the return codes in this way is to allow the reconstruction of the polynomial when at least k of these points are known. We will use this property to prove the validity of an encrypted vote. Finally, the authorities define an identifier ID_i (e.g., $ID_i = i$), pick random values $V_i \in_R \{0,1\}^v$ and $C_i \in_R \{0,1\}^c$, and compute $F_i = H_f(R_{i,1} \| \cdots \| R_{i,n}) \in \{0,1\}^f$. The resulting tuple $(ID_i, V_i, C_i, F_i, \{(c_j, R_{ij})\}_{j=1}^n)$ is sent to voter i over a secure channel.

After generating verification code sheet i, the authorities select the value $P_i = p_i(0) = a_{i,0} \in \mathbb{Z}_{p'}$. Note that the points P_{ij} can be seen as the n shares obtained from applying Shamir's (k,n)-threshold secret sharing scheme to a secret P_i. Commitments $CV_i \leftarrow \mathsf{Commit}(V_i, \alpha_i)$ and $CC_i \leftarrow \mathsf{Commit}(C_i, P_i, F_i, \beta_i)$ are posted to the public bulletin board for randomizations $\alpha_i, \beta_i \in_R \mathbb{Z}_q$, respectively. The purpose of publishing the set $\{(ID_i, CV_i, CC_i)\}_{i=1}^N$ is to enable the verification that each ballot has been submitted by someone in possession of a valid verification code sheet. This set can therefore be regarded as the electoral roll in a context where possessing a verification code sheet implies eligibility.

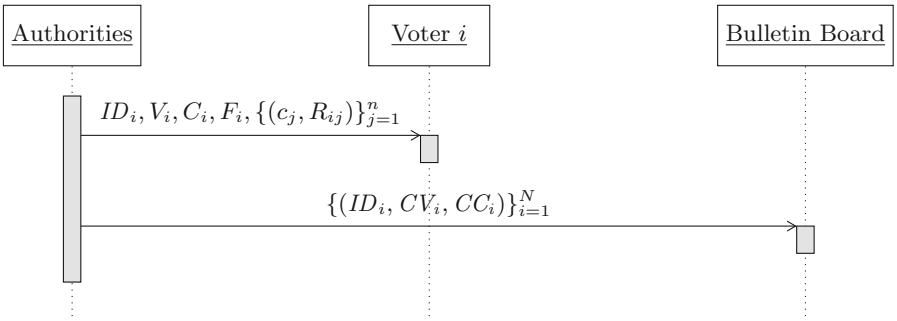

Fig. 2. Sequence diagram of the election preparation phase.

Vote Casting. The vote casting and confirmation phase is the core of the protocol. An overview of the exchanged messages is given in Fig. 3. To initiate the process, the voter enters the code sheet identifier ID_i, the voting code V_i, and the selected candidates $\mathbf{s} = (s_1, \ldots, s_k)$ into the voting platform. The voting platform

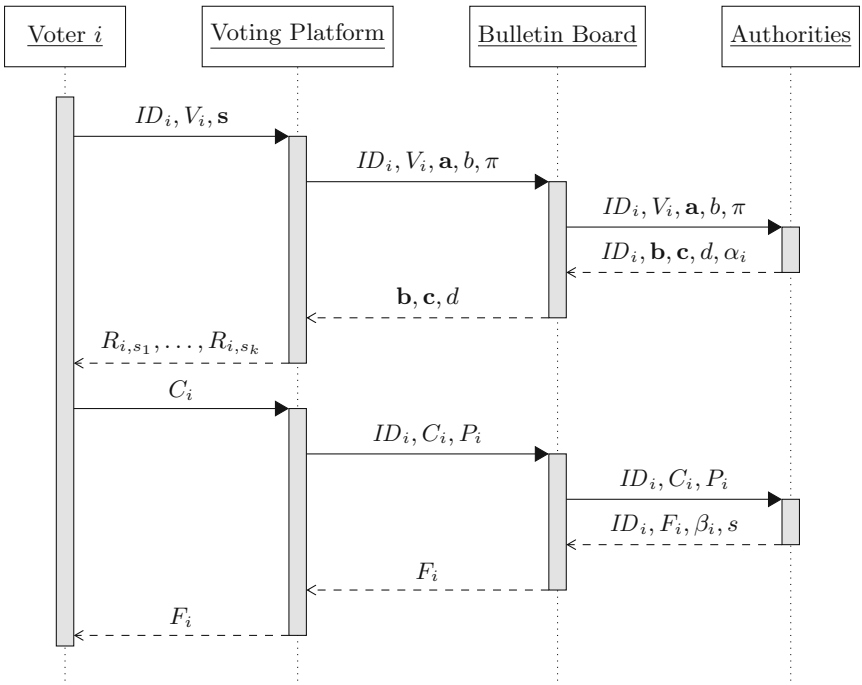

Fig. 3. Sequence diagram of the vote casting and confirmation phase.

then computes a ballot containing an OT_n^k query for the k points $P_{i,s_1}, \ldots, P_{i,s_k}$ (from which the return codes $R_{i,s_1}, \ldots, R_{i,s_k}$ of the k chosen candidates and the value P_i can be derived). For this, the voting platform picks random values $\mathbf{r} \in_R \mathbb{Z}_q^k$ and computes $\mathbf{a} \leftarrow \mathsf{Query}(\mathbf{s}, \mathbf{r})$. There are some important technical details in this step:

- Since we use the OT_n^k protocol to transfer points $P_{ij} \in \mathbb{Z}_{p'} \times \mathbb{Z}_{p'}$, we instantiate the protocol with a message length $\ell = 2 \cdot \lceil \log p' \rceil$. This allows us to encode each of the two coordinates of P_{ij} by $\frac{\ell}{2}$ bits and to concatenate them together.
- The OT_n^k protocol as presented in Sect. 2.1 requires a generator g of $\mathbb{G}_q$. Since $\mathbb{G}_q$ is of prime order, any value in $\mathbb{G}_q \setminus \{1\}$ is admissible. To establish a natural link to the encrypted vote, we require the authorities' public key $pk \in \mathbb{G}_q$ to be used as generator for the oblivious transfer.
- For the encoding $\Gamma : \{1, \ldots, n\} \to \mathbb{G}_q$ used in the OT_n^k protocol, we use the set $\mathbb{P}_n = \{p_1, \ldots, p_n\}$ of the n smallest prime numbers $p_i \in \mathbb{G}_q$, $p_i < p_{i+1}$, and define $\Gamma(i) = p_i$. The purpose of this particular choice is to encode $\mathbf{s}$ as a product $\Gamma(\mathbf{s}) = \prod_{j=1}^{k} p_{s_j}$, which can then be encrypted using ElGamal. Note that inverting $\Gamma(\mathbf{s})$ by factorization is unique if the product of the largest k primes in $\mathbb{P}_n$ is smaller than q and efficient when n is small [Gjø11].

Since the query $\mathbf{a} = (a_1, \ldots, a_k)$ generated in this way contains values $a_j = \Gamma(s_j) \cdot pk^{r_j}$, we can compute a single value

$$a = \prod_{j=1}^{k} a_j = \prod_{j=1}^{k} \Gamma(s_j) \cdot pk^{r_j} = \Gamma(\mathbf{s}) \cdot pk^r,$$

where $r = \sum_{j=1}^{k} r_j$. Therefore, by computing a second value $b = g^r$, we obtain an ElGamal encryption $(a, b) = \mathsf{Enc}_{pk}(\Gamma(\mathbf{s}), r)$ of the encoded voter's selections $\Gamma(\mathbf{s})$. This simple connection between the OT_n^k query and the encrypted vote is crucial for making the protocol efficient.

The remaining component for forming the ballot is a non-interactive zero-knowledge proof $\pi \leftarrow \mathsf{GenNIZKP}_{\mathsf{Enc}_{pk}}((\Gamma(\mathbf{s}), r), (a, b), V_i)$ for proving knowledge of $\Gamma(\mathbf{s})$ and r. Note that we use V_i as an additional input to the proof generation to disallow copying of encrypted votes. The resulting *ballot* $B = (ID_i, V_i, \mathbf{a}, b, \pi)$ is posted to the bulletin board, from where it can be retrieved by the authorities. If V_i is the correct voting code for code sheet ID_i and if π is a valid proof, they pick a random $s \in_R \mathbb{Z}_q$, compute the response $(\mathbf{b}, \mathbf{c}, d) \leftarrow \mathsf{Response}(\mathbf{a}, (P_{i,1}, \ldots, P_{i,n}), s)$, and return $(\mathbf{b}, \mathbf{c}, d)$ to the voting platform (only if no valid ballot for ID_i has been posted earlier). Since no private channel is needed for this, we propose to send it via the bulletin board. We include ID_i and α_i in this message, which means that the commitment CV_i is opened. The full message is a tuple $(ID_i, \mathbf{b}, \mathbf{c}, d, \alpha_i)$.

Vote Confirmation. Upon receiving the response from the authorities, the voting platform computes the result $(P_{i,s_1}, \ldots, P_{i,s_k}) \leftarrow \mathsf{Open}(\mathbf{b}, \mathbf{c}, d, \mathbf{r})$ of the oblivious transfer. Corresponding return codes $R_{i,s_j} = H_r(P_{i,s_j})$ are displayed to the voter for inspection. If they match with the codes printed on the verification code sheet, the vote must have been cast and recorded as intended with high probability, which the voter confirms by entering the confirmation code C_i into the voting platform. This code is forwarded to the bulletin board together with $P_i = p_i(0)$, which can be computed by interpolating the polynomial $p_i(x)$ from the received points $(P_{i,s_1}, \ldots, P_{i,s_k})$ using Lagrange's method.

If both C_i and P_i are correct, the authorities respond by sending the finalization code F_i to the voter for inspection. If F_i as displayed by the voting platform matches with the finalization code on the code sheet, the vote confirmation must have been successful with high probability. Again, since keeping F_i private is no longer necessary at this point, we propose to send it via the bulletin board to the voter. By including the randomizations β_i, commitment CC_i of code sheet i is opened and can be publicly verified. Similarly, by including the randomization s, the commitment d of the OT_n^k response $(\mathbf{b}, \mathbf{c}, d)$ is opened and all n points P_{ij} are revealed, together with corresponding return codes $R_{ij} = H_r(P_{ij})$ of code sheet i. Public verifiers can then check if $F_i = H_f(R_{i,1} \| \cdots \| R_{i,n})$ holds, which implies that the authorities have responded properly to the OT_n^k query. Public verifiers can also interpolate the polynomial $p_i(x)$ over the points $\{P_{ij}\}_{j=1}^{n}$, check if its degree is $k-1$, and verify that $p_i(0) = P_i$. This guarantees that the

random points P_{ij} and the value P_i have been generated properly during the election preparation.[4]

Tallying. After the election period, the bulletin board contains one or multiple entries for every ID_i. There are several types of entries, depending on whether someone has participated in the election and on whether vote casting and confirmation has been successful:

- (ID_i, CV_i, CC_i): The voter has not participated in the election.
- $(ID_i, CV_i, CC_i, V_i, \mathbf{a}, b, \pi)$: The voter has initiated the vote casting process, but the process stopped after submitting the ballot. Possible causes are an incorrect voting code V_i, an invalid zero-knowledge proof π, or the existence of an earlier valid ballot for ID_i.
- $(ID_i, CV_i, CC_i, V_i, \mathbf{a}, b, \pi, \mathbf{b}, \mathbf{c}, d, \alpha_i)$: The authorities have responded to the OT_n^k query, but either the voter has not entered the confirmation code or the voting platform has not forwarded it to the bulletin board.
- $(ID_i, CV_i, CC_i, V_i, \mathbf{a}, b, \pi, \mathbf{b}, \mathbf{c}, d, \alpha_i, C_i, P_i)$: The voting platform has sent values C_i and P_i to the bulletin board, but then the process has stopped. Possible causes are incorrect values C_i or P_i.
- $(ID_i, CV_i, CC_i, V_i, \mathbf{a}, b, \pi, \mathbf{b}, \mathbf{c}, d, \alpha_i, C_i, P_i, F_i, \beta_i, s)$: This is the success case, in which the authorities have responded to correct values C_i and P_i with the finalization code F_i and randomization s.

It is evident that only ballots from the success case can be considered in the tally. A list of corresponding ElGamal encryptions $(a, b) = (\prod_{j=1}^{k} a_j, b)$ is extracted for further processing. As mentioned earlier, we do not further discuss the tallying part of the protocol, because this is well-studied in the literature of electronic voting protocols. We simply assume that this process reveals—in a publicly verifiable manner—a list of plaintext votes $\Gamma(\mathbf{s})$, which can be decoded into the voter's selections $\mathbf{s} = (s_1, \ldots, s_k)$. Accumulating these selections over all valid votes generates the final election result.

Verification. At the end of an election, a number of verifications can be performed by the public. In Table 1, we list all computations and checks that can be performed for every submitted ballot in the success case. In our setting, in which possessing a verification code sheet implies eligibility, these checks prove

[4] Without such checks, malicious authorities could actively attack the vote secrecy of some voters by responding to the OT_n^k query with some incorrect return codes. If the voter then confirms the ballot as cast, the authorities learn that no candidate corresponding to an incorrect return code has been selected. A similar attack could be launched during the election preparation. If some of the random points P_{ij} are not selected from the polynomial, then responding with the correct value P_i tells the authorities that no candidate corresponding to such an incorrect point has been selected. In the covert adversary model, publishing s prevents both variants of this attack (see paragraph on vote secrecy in Sect. 4.1).

that every valid vote has been submitted by an eligible voter and that every eligible voter has voted at most once. To achieve a complete chain of universal verifiability, we assume that the authorities publish cryptographic proofs for the correctness of the election result (corresponding checks are not listed in Table 1).

By performing the computations of Table 1 on their own ballot, participating voters can verify the ballot consistency and the inclusion of their vote in the tally. By checking the validity of the involved commitments, they can verify the consistency of their verification code sheet. It is also possible to check that the return codes have been generated properly and that the authorities responded faithfully to the OT query. Abstaining voters can check that their verification code sheet has not been used by an attacker.

Table 1. List of computations and checks to verify the validity of a ballot in the success case, which corresponds to an entry $(ID_i, CV_i, CC_i, V_i, \mathbf{a}, b, \pi, \mathbf{b}, \mathbf{c}, d, \alpha_i, C_i, P_i, F_i, \beta_i, s)$ on the bulletin board.

Computations	Range	Checks
$d_1 \leftarrow \mathsf{Decommit}(CV_i, V_i, \alpha_i)$		$d_1 = 1$
$d_2 \leftarrow \mathsf{Decommit}(CC_i, C_i, P_i, F_i, \beta_i)$		$d_2 = 1$
$a' = \prod_{j=1}^{k} a_j$		
$v \leftarrow \mathsf{VerifyNIZKP}_{\mathsf{Enc}_{pk}}(\pi, (a', b), V_i)$		$v = 1$
$d' = pk^s$		$d' = d$
$b'_j = a_j^s$	$j = 1, \ldots, k$	$b'_j = b_j$
$P'_{ij} = H_r(c_j \oplus H_\ell(\Gamma(j)^s) = (x'_{ij}, y'_{ij})$	$j = 1, \ldots, n$	
$R'_{ij} = H_r(P'_{ij})$	$j = 1, \ldots, n$	
$F'_i = H_f(R'_{i,1} \| \cdots \| R'_{i,n})$		$F'_i = F_i$
interpolate $p'_i(x) = \sum_{j=0}^{n-1} a'_{ij} x^j$ over $\{P'_{ij}\}_{j=1}^{n}$	$j = k, \ldots, n-1$	$a'_{ij} = 0$
		$a'_{i,k-1} \neq 0$
		$a'_{i,0} = P_i$

3.4 Multiple Authorities

The protocol as presented above generalizes naturally to $t \geq 1$ authorities such that no single authority knows the codes of code sheet i. Each authority generates its own verification code sheet exactly as described in Sect. 3.3 and transmits it to voter i over the secure channel. During vote casting, voters send a single OT_n^k query to all authorities, which can respond individually and simultaneously. The actual return codes are $R_{ij} = \oplus_{k=1}^{t} H_r(P_{ijk})$, where P_{ijk} denotes the j-th point on the random polynomial picked by authority k for code sheet i. In a similar way, multiple finalization codes F_{ik} can be merged into a single finalization code

$F_i = \oplus_{k=1}^{t} F_{ik}$. Finally, voting and confirmation codes are concatenated into $V_i = V_{i,1} \| \cdots \| V_{i,t}$ and $C_i = C_{i,1} \| \cdots \| C_{i,t}$, respectively.[5]

4 Discussion

In this section, we will briefly discuss the security properties and the performance of the proposed cryptographic voting protocol and compare it to the existing work in the literature.

4.1 Security

The principal goal of the proposed cast-as-intended verification mechanism is to enable the detection of an attack by malware on the voting platform without compromising vote secrecy on the server side. If an attack—or a defective system—is detected by some voters, it is assumed that they have access to an alternative voting channel such as postal mail.

Correctness. Submitting a ballot that makes it into the final tally requires knowledge of the codes V_i, C_i, and P_i of a valid verification code sheet $i \in \{1, \ldots, N\}$. Any attempt to submit a ballot with incorrect codes will be detected and prohibited by the authorities. Guessing correct codes or an exhaustive search for correct codes can be prevented with high probability by choosing large enough length parameters v and c and a large enough prime p'. Any attempt to submit multiple ballots with the same codes V_i, C_i, and P_i will also be detected and prohibited by the authorities. The authorities themselves can only compute correct codes and use them to submit a ballot if they all collude. A single honest authority is therefore sufficient to prevent ballot stuffing.

If a malicious voting platform tries to submit votes for candidates different from the voter's intention, then the return codes will not match and the voters will abort the voting process. Submitting less than k of the voter's actual selections will be detected as well, because $p_i(x)$ can not be interpolated and P_i can not be computed in this case. Submitting a vote for more than k candidates will be detected and prohibited by the authorities. Submitting an invalid value b along with the OT_n^k query $\mathbf{a}$ is prevented by the non-interactive zero-knowledge proof π, i.e., such attempts will be detected by the authorities. Waiting for the voter to enter the confirmation code and then changing the submitted ballot is prevented by the append-only property of the bulletin board. Not submitting the ballot or the values C_i and P_i can not be prevented, but this will be detected by the voter with high probability when a wrong response or no response at all is displayed.

[5] Concatenation of voting and confirmation codes is the simplest possible solution to generalize the protocol to multiple authorities. As a consequence, the lengths of F_i and C_i are multiplied by t, which may cause problems from a usability point of view. A discussion of such usability problems and proposals for more sophisticated solutions are beyond the scope of this paper.

Vote Secrecy. Guaranteeing vote secrecy on a malware-infected voting platform is impossible in a system in which voters enter their selections in plaintext. As a consequence, our protocol does not solve this problem. On the server side, provided that a proper privacy-preserving tallying procedure is in place, vote secrecy is guaranteed under the assumptions that the DDH problem is hard (which implies IND-CPA security for ElGamal encryptions) and that a threshold number of authorities holding a share of the private key sk is honest. If this is the case, no information about the voter's selections $\mathbf{s}$ is leaked by publishing the ballot $B = (ID_i, V_i, \mathbf{a}, b, \pi)$ on the bulletin board.

Submitting the values C_i and P_i to confirm matching return codes does not reveal anything about the voter's selections to the public, but malicious authorities could break vote secrecy by responding with some incorrect return codes to the OT_n^k query or by sending some incorrect return codes over the secure channel during election preparation. In both cases, confirming the vote reveals to the authorities that no candidate corresponding to an incorrect return code has been selected. In the covert adversary model, our protocol prevents an attack of the first type by requesting the authorities to reveal the randomization s of the OT_n^k response. This permits public verifiers to compute the return codes of all candidates of a given code sheet and to check if these codes match with the finalization code. Any attempt to respond with incorrect return codes would be detected in this way. To detect attacks of the second type and thus to prevent covert adversaries from conducting them, voters could be asked to check if all return codes match with the code sheet and to report to the election administration if this is not the case. Clearly, this is not very practical from usability point of view, especially if n is large, but our protocol does not offer a better solution for this problem.

4.2 Comparison to Existing Work

In Table 2, we present a performance comparison between our approach and the two most relevant approaches from the literature. Since the approach presented in [HLv10] turned out to be much less efficient, we do not further discuss its properties and exclude it from the subsequent comparison.

Compared to the Neuchâtel protocol [GGP15], our approach offers a number of conceptual advantages. First, while the Neuchâtel protocol requires three different types of server-side parties (registrars, code generator, voting server), which are pairwise assumed not to collude, we only require a threshold number of non-colluding authorities performing identical operations. This implies that our protocol offers better flexibility in terms of robustness. Second, while the Neuchâtel protocol requires a private channel to transmit the return codes from the code generator to the voters (otherwise vote secrecy could be violated by the registrars), we can send the OT_n^k response over a public channel. Third, there are two types of private keys in the Neuchâtel protocol, which are used by multiple parties. This creates unnecessary and uncommon trust assumptions, which we do not have in our protocol. Finally, while nN so-called *reference values* need to

Table 2. Performance comparison between the protocol of this paper and existing work in terms of exponentiations in the underlying group. The values given in parentheses indicate the number of exponentiations that can be pre-computed. In the case of [HLv10], which is restricted to 1-out-of-n votes, we assume that k votes are submitted in parallel.

		This paper	[GGP15]	[HLv10]
Election preparation	Authorities	$6N$	$(n+2)N$	nN
Vote casting	Voting platform	$2k+3$	$k+10$	$k(7\log n+8)$
		$(k+3)$	(7)	$(k(6\log n+8))$
	Authorities	$n+k+5$	11	$k(5n+6\log n+8)$
		$(n+1)$	(0)	$(k(2n+2\log n))$

be generated and stored in the Neuchâtel protocol for proving vote correctness, we achieve the same in a more elegant way using only N values $P_1, \ldots, P_N$.

In the light of the numbers shown in Table 2, the overall performance of the two protocols is similar. While the election preparation is considerably more efficient in our protocol when n is large, our approach requires more expensive online computations during vote casting. However, if we assume that the voting platform performs pre-computations in the background while the voter is interacting with the voting platform, our approach is slightly more efficient: k versus $k+3$ online exponentiations. If we assume that pre-computations are also performed on the server side, our approach is more efficient for $k < 7$ and less efficient for $k > 7$. Note that server-side pre-computations can be performed well in advance, for example as part of the election preparation. In that case, the overall performance of the election preparation is very similar: $(n+1)N'+6N$ versus $(n+2)N$ exponentiations, where $N' \leq N$ denotes the maximal expected number of participating voters. Nevertheless, by allowing server-side pre-computations at any moment before an election, not necessarily as part of the election preparation, our approach is slightly more flexible.

5 Conclusion

The cryptographic voting protocol presented in this paper introduces a new mechanism for cast-as-intended verification based on oblivious transfer. We believe that the problem of transferring return codes as a response to submitting an encrypted vote *is* an oblivious transfer problem and therefore should be solved as such. The approach presented in this paper is the first efficient solution. Compared to existing cast-as-intended verification methods, our approach is conceptually more elegant and requires less trust assumptions and cryptographic keys. We think that it offers an appropriate solution for countries such as Switzerland, where providing a solution to the secure platform problem is a prerequisite for introducing the next-generation systems. We have been invited

by the State of Geneva to participate in implementing this approach for their future system. Formal security proofs will be developed in a separate project.

Acknowledgments. We thank the anonymous reviewers for their reviews and appreciate their comments and suggestions. We are also grateful to Stephan Fischli, Severin Hauser, Thomas Hofer, and Philipp Locher for helpful discussions and proofreading. This research has been supported by the State of Geneva.

References

[AIR01] Aiello, B., Ishai, Y., Reingold, O.: Priced oblivious transfer: how to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001). doi:10.1007/3-540-44987-6_8

[AL10] Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. J. Cryptol. **23**(2), 281–343 (2010)

[BK113a] Ergänzende Dokumentation zum dritten Bericht des Bundesrates zu Vote électronique. Die Schweizerische Bundeskanzlei (BK) (2013)

[BK113b] Technische und administrative Anforderungen an die elektronischen Stimmabgabe. Die Schweizerische Bundeskanzlei (BK) (2013)

[BK113c] Verordnung der Bundeskanzlei über die elektronische Stimmabgabe (VEleS). Die Schweizerische Bundeskanzlei (BK) (2013)

[Bol03] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). doi:10.1007/3-540-36288-6_3

[CT05] Chu, C.-K., Tzeng, W.-G.: Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 172–183. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30580-4_12

[CT08] Chu, C.K., Tzeng, W.G.: Efficient k-out-of-n oblivious transfer schemes. J. Univ. Comput. Sci. **14**(3), 397–415 (2008)

[GB12] Gebhardt Stenerud, I.S., Bull, C.: When reality comes knocking - Norwegian experiences with verifiable electronic voting. In: Kripp, M.J., Volkamer, M., Grimm, R. (eds.) EVOTE 2012, 5th International Workshop on Electronic Voting, Bregenz, Austria. Lecture Notes in Informatics, vol. P-205, pp. 21–33 (2012)

[GGP15] Galindo, D., Guasch, S., Puiggalí, J.: 2015 Neuchâtel's cast-as-intended verification mechanism. In: Haenni, R., Koenig, R.E., Wikström, D. (eds.) VOTELID 2015. LNCS, vol. 9269, pp. 3–18. Springer, Heidelberg (2015). doi:10.1007/978-3-319-22270-7_1

[Gjø10] Gjøsteen, K.: Analysis of an internet voting protocol. IACR Cryptology ePrint Archive, 2010/380 (2010)

[Gjø11] Gjøsteen, K.: The Norwegian internet voting protocol. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 1–18. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32747-6_1

[HLv10] Heiberg, S., Lipmaa, H., Laenen, F.: On E-vote integrity in the case of malicious voter computers. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 373–388. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15497-3_23

[Lip11] Lipmaa, H.: Two simple code-verification voting protocols. IACR Cryptology ePrint Archive, 2011/317 (2011)

[Mau09] Maurer, U.: Unifying zero-knowledge proofs of knowledge. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 272–286. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02384-2_17

[Ped91] Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (1991). doi:10.1007/3-540-46416-6_47

[PG11] Allepuz, J.P., Castelló, S.G.: Internet voting system with cast as intended verification. In: Kiayias, A., Lipmaa, H. (eds.) Vote-ID 2011. LNCS, vol. 7187, pp. 36–52. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32747-6_3

[PG12] Puiggalí, J., Guasch, S.: Cast-as-intended verification in Norway. In: Kripp, M., Volkamer, M., Grimm, R. (eds.) EVOTE 2012, 5th International Workshop on Electronic Voting, Bregenz, Austria. Lecture Notes in Informatics, vol. P-205, pp. 49–63 (2012)

[SV12] Schläpfer, M., Volkamer, M.: The secure platform problem: taxonomy and analysis of existing proposals to address this problem. In: ICEGOV 2012, 6th International Conference on Theory and Practice of Electronic Governance, Albany, USA (2012)