Verifiable Internet Elections with Everlasting Privacy and Minimal Trust

Philipp Locher[1,2(✉)] and Rolf Haenni[1]

[1] Bern University of Applied Sciences,
2501 Biel, Switzerland
{philipp.locher,rolf.haenni}@bfh.ch
[2] University of Fribourg,
1700 Fribourg, Switzerland
philipp.locher@unifr.ch

Abstract. This paper presents a new cryptographic Internet voting protocol based on a set membership proof and a proof of knowledge of the representation of a committed value. When casting a vote, the voter provides a zero-knowledge proof of knowledge of the representation of one of the registered voter credentials. In this way, votes are anonymized without the need of trusted authorities. The absence of such authorities reduces the trust assumptions to a minimum and makes our protocol remarkably simple. Since computational intractability assumptions are only necessary to prevent the creation of invalid votes during the voting period, but not to protect the secrecy of the vote, the protocol even offers a solution to the everlasting privacy problem.

1 Introduction

Two types of trust assumptions are commonly found in cryptographic voting protocols. First, it is usually assumed that a threshold number of non-colluding trusted authorities exists, for instance for mixing the list of encrypted votes or for decrypting them in a distributed manner. In an ideal setting, each trusted authority is completely independent from all the others, both in terms of the people engaged in providing the expected service and in terms of the available computer and software infrastructure. In practice, recruiting such a group of trusted authorities and equipping them with independent hard- and software is a very difficult problem.

The second type of assumptions in cryptographic voting protocols limits the adversary's computational capabilities, for example with respect to computing discrete logarithms or factoring large numbers. Such computational intractability assumptions are very common in many cryptographic applications, but they are very problematical in the context of electronic elections. It means that the secrecy of the votes of an election today may be at risk in the future, when more powerful computers and better methods of cryptanalysis are available. Choosing very conservative security parameters may postpone the privacy breach, but does not prevent it.

DOI: 10.1007/978-3-319-22270-7_5

1.1 Contribution

The contribution of this paper is a new cryptographic voting protocol for remote electronic elections, which guarantees the secrecy of the vote without relying on trusted authorities or on computational intractability assumptions. It offers therefore *everlasting privacy* in an information-theoretical sense. Trusted authorities are only needed for fairness, and computational intractability assumptions are only necessary to prevent the creation of invalid votes during the voting period.

From a technical point of view, our protocol differs strongly from mainstream approaches such as those based on homomorphic tallying, mix-nets, or blind signatures. The core of the protocol is a combination of a set membership proof [3] and a proof of known representation of a committed value [2]. When casting a vote, the voter provides a zero-knowledge proof of knowledge of the representation of one of the registered public voter credentials. Informally, the protocol consists of the following four consecutive steps (some details about preventing double voting or providing fairness are left out):

- **Registration:** Each voter creates a pair of private and public voter credentials and sends the public voter credentials over an authentic channel to the election administration.
- **Election Preparation:** The election administration publishes the list of public voter credentials—one for every registered voter—on the public bulletin board.
- **Vote Casting:** The voter creates an electronic ballot and sends it over an anonymous channel to the public bulletin board. The ballot consists of the vote, a commitment to the voter's public credential, and the above-mentioned composition of zero-knowledge proofs.
- **Public Tallying:** At the end of the election period, anyone can derive the final election result from the data published on the public bulletin board. The correctness of the result follows from verifying the zero-knowledge proofs included in the ballots.

The proofs included in every ballot are computationally sound and perfectly zero-knowledge. This implies with very high probability that every vote with a valid proof stems from an eligible voter, but nothing more than that. Every single voter remains completely anonymous within the set of registered voters, independently of the computational capabilities of a future adversary. In this way, our protocol achieves everlasting privacy without the help of trusted authorities. As a consequence, the protocol requires almost no central infrastructure and no complicated process coordination. Except during registration, interactions are limited to writing data to and reading data from the public bulletin board. The main computational efforts are spent by the voters themselves during vote casting and by anyone computing and verifying the final election result.

1.2 Related Work

The position that only the strongest notion of privacy is sufficient for electronic voting has first been proclaimed by Chaum [10]. He argued that ballot secrecy must be *unconditionally secure*, meaning that the partial tally of a group of voters can only be determined by a coalition of all other voters. In its strict sense, this definition includes an adversary with unlimited computational power. Two protocols by Kiayias and Yung [16] and Groth [15] achieve a weaker form of so-called *perfect ballot secrecy* under the Decisional Diffie-Hellman assumption. They are intended for use in the context of boardroom voting with a small number of participants. Both protocols are *self-tallying*, meaning that the election result can be computed without the aid of a trusted authority.

In a series of papers [18–20], Moran and Naor proposed several protocols with everlasting privacy (but not unconditional privacy according to Chaum's definition). They are intended for use in the traditional setting, in which ballots are cast in a private polling booth. In all three protocols, everlasting privacy is only achieved with the aid of a single or a group of trusted authorities, which could potentially cause voter privacy to be breached. Another protocol [21] for the traditional setting achieves everlasting privacy by combining concepts from Punchscan and Prêt àVoter.

In a more recent series of papers [6, 11, 12], everlasting privacy with the aid of trusted authorities has been brought into the context of remote elections. While the information published on the public bulletin board does not reveal anything about somebody's vote, the trusted server could potentially break the encrypted votes transmitted over the private channel between voter and server.

Another important line of related work are the protocols based on blind signatures [14]. They are also based on submitting votes over an anonymous channel, but they achieve everlasting privacy under much stronger trust assumptions. Their main problem is ballot-stuffing by malicious signing authorities, which cannot be detected. More generally speaking, protocols based on blind signatures do not support the verification of the electorate. Other disadvantages are the facts that voters need to interact with the authorities during vote casting and that the authorities learn who actually voted. To overcome some of the drawbacks of blind signatures, Canard and Traoré introduced a system based on list signatures [9].

1.3 Paper Overview

In the following section, we introduce the cryptographic building blocks of our protocol. In particular, we describe possible instances of a set membership proof and a proof of known representation of a committed value. In Sect. 3, we provide a detailed description of our protocol and a discussion of the underlying adversary model and the resulting system properties. In Sect. 4, we analyse the running times of the vote casting and tallying procedures and present the results from corresponding performance tests. Finally, we summarize the findings of this paper in Sect. 5.

2 Cryptographic Preliminaries

Let $\mathcal{G}_p$ be a multiplicative cyclic group of prime order p, for which the discrete logarithm assumption is believed to hold. Furthermore, let $\mathbb{G}_q \subset \mathbb{Z}_p^*$, be a large prime-order subgroup of the group of integers modulo p, where $\gamma = (p-1)/q$ denotes the corresponding co-factor. Finally, suppose that independent generators $g_0, g_1 \in \mathcal{G}_p$ and $h_0, \dots, h_N \in \mathbb{G}_q$ are publicly known. Independence with respect to generators of a cyclic group means that their relative discrete logarithms are unknown.[1]

In our protocol, we use two instances of the perfectly hiding Pedersen commitment scheme, one over $\mathcal{G}_p$ and one over $\mathbb{G}_q$. We distinguish them by $\text{com}_p(u,r) = g_0^r g_1^u$ as a commitment to u with randomization r and $\text{com}_q(v,s) = h_0^s h_1^v$ as a commitment to v with randomization s, where $u, r \in \mathbb{Z}_p$ and $v, s \in \mathbb{Z}_q$. In the case of $\mathbb{G}_q$, we write $\text{com}_q(v_1, \dots, v_N, s) = h_0^s h_1^{v_1} \cdots h_N^{v_N}$ for a commitment to N values $v_1, \dots, v_N \in \mathbb{Z}_q$. Recall that Pedersen commitments are perfectly hiding, computationally binding, and additively homomorphic.

The main cryptographic tools in our protocol are non-interactive zero-knowledge proofs of knowledge. The voter uses them to demonstrate knowledge of some secret values involved in a mathematical statement, but without revealing any information about the secret values. One of the most fundamental type of zero-knowledge proofs of knowledge is a preimage proof for a one-way group homomorphism $\phi : X \rightarrow Y$, denoted by $\mathit{NIZKP}[(a) : b = \phi(a)]$, where $a \in X$ is the secret preimage of a public value $b = \phi(a) \in Y$. Examples of such preimage proofs result from the above additively homomorphic Pedersen commitment schemes, for example $\mathit{NIZKP}[(u,r) : c = \text{com}_p(u,r)]$ for proving knowledge of an opening $u, r \in \mathbb{Z}_p$ for a publicly known commitment $c \in \mathcal{G}_p$.

The most common construction of a non-interactive preimage proof is the Σ-protocol in combination with the Fiat-Shamir heuristic [13]. The transcript of such a non-interactive proof consists of one or multiple commitments and one or multiple responses to a challenge computed by a publicly known hash function. Some auxiliary information can be linked to the transcript by using it as an additional input to the hash function. In Sect. 3, we will write $\pi_i = \mathit{NIZKP}_x[\cdot]$ for the transcripts of the non-interactive proofs used in the voting protocol, where x represents some auxiliary information linked to the proof.

2.1 Set Membership Proof

Let $U = \{u_1 \dots, u_M\}$ be a finite set of values $u_i \in \mathbb{Z}_p$ and $c = \text{com}_p(u,r)$ a commitment to an element $u \in U$. Both U and c are publicly known. With a *set membership proof*, denoted by $\mathit{NIZKP}[(u,r) : c = \text{com}_p(u,r) \wedge u \in U]$, the prover demonstrates knowledge of corresponding values $u \in U$ and $r \in \mathbb{Z}_p$, but without revealing any information about them. Such a proof can be constructed by a standard OR combination of individual preimage proofs for each $u \in U$, but this proof has a size linear to M and is therefore not efficient. The first set membership proof with a sub-linear size has been given by Camenisch et al. [7].

As suggested by Brands et al. [5], a general way of constructing a set membership proof is to compute the polynomial $P(X) = \prod_{i=1}^{M}(X - u_i)$ and

[1] To ensure that generators are independent, they need to be generated in some publicly reproducible way, for example by deriving them from a common reference string.

to demonstrate that $P(u) = 0$. This proof, denoted by $\mathit{NIZKP}[(u,r) : c = \mathrm{com}_p(u,r) \wedge P(u) = 0]$, is a particular case of a *polynomial evaluation proof* $\mathit{NIZKP}[(u,r,v,s) : c = \mathrm{com}_p(u,r) \wedge d = \mathrm{com}_p(v,s) \wedge P(u) = v]$ for $v = s = 0$. In a recent publication [3], Bayer and Groth proposed a polynomial evaluation proof with a logarithmic size, which is the current state-of-the-art. We use a non-interactive version of this proof in our voting protocol, instantiated to the special case of $v = s = 0$. A summary of the proof generation and verification is given in Fig. 1.

A complete proof transcript consists of $4\lfloor \log M \rfloor + 2$ elements of $\mathcal{G}_p$ and $3\lfloor \log M \rfloor + 3$ elements of $\mathbb{Z}_p$. The proof generation requires $8\lfloor \log M \rfloor + 4$ exponentiations in $\mathcal{G}_p$ and not more than $2M\lfloor \log M \rfloor$ multiplications in $\mathbb{Z}_p$. Similarly, $6\lfloor \log M \rfloor + 6$ exponentiations in $\mathcal{G}_p$ and $3M$ multiplications in $\mathbb{Z}_p$ are needed for the verification.[2] In terms of exponentiations only, the computational costs for generating and verifying a proof are both logarithmic with M, but for very large values M, the cost of the (quasi-)linear number of multiplications becomes dominant.

2.2 Proof of Known Representation of a Committed Value

In a cyclic group such as $\mathbb{G}_q$ with generators $h_1, \ldots, h_N$, a tuple $(v_1, \ldots, v_N)$ of values $v_i \in \mathbb{Z}_q$ is called *DL-representation* (or simply *representation*) of $u \in \mathbb{G}_q$ with respect to $(h_1, \ldots, h_N)$, if $u = h_1^{v_1} \cdots h_N^{v_N}$ [4]. Note that the general definition of DL-representation does not require the values $h_1, \ldots, h_N$ to be generators, nor do they need to be independent or distinct. On the other hand, every opening of a Pedersen commitment is clearly a DL-representation of the commitment with respect to the given independent generators.

Let $c = \mathrm{com}_p(u, r)$ be a commitment to a single value $u \in \mathbb{G}_q \subset \mathbb{Z}_p$ and $d = \mathrm{com}_q(v_1, \ldots, v_N, s)$ a commitment to multiple values $v_1, \ldots, v_N \in \mathbb{Z}_q$. Both c and d are publicly known. Following Au et al. [2], a proof of known *representation of a commmited value* (or simply *representation proof*), denoted by

$$\begin{aligned} \mathit{NIZKP}[(u, r, v_1, \ldots, v_N, s) :\ & c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_q(v_1, \ldots, v_N, s) \\ & \wedge\ u = h_1^{v_1} \cdots h_N^{v_N}], \end{aligned}$$

demonstrates that the tuple of committed values in d is a DL-representation of the committed value in c. Note that this is a generalization of proof of knowledge of double discrete logarithms, $\mathit{NIZKP}\{(v) : c = g^{(h^v)}\}$, by Camenisch and Stadler [8]. A summary of the proof generation and verification is given in Fig. 2, where a security parameter K determines the soundness of the proof.

A complete proof transcript consists of $K + 1$ elements of $\mathcal{G}_p$, K elements of $\mathbb{G}_q$, $K + 2$ elements of $\mathbb{Z}_p$, and $K(N + 1)$ elements of $\mathbb{Z}_q$. Note that elements

[2] The number of exponentiations given in [3, Table 2] is incorrect for the verification. The correct result of $6\lfloor \log M \rfloor$ exponentiations is obtained by counting c_j^x in Step 2 and c_{j+1}^x in Step 3 as one exponentiation only. This remark together with the correct result can be found in [3, Page 11], i.e., only the table entry is incorrect. Furthermore, we cannot reproduce the result of $2M$ multiplications for the verification reported in [3, Table 2]. According to our analysis, at least $3M$ multiplications are needed.

Public Input: $c = \mathrm{com}_p(u, r) \in \mathcal{G}_p$, $P(X) = \sum_{i=0}^{M} a_i X^i \in \mathbb{Z}_p[X]$
Secret Input: $u, r \in \mathbb{Z}_p$
Generation:

1. For $j = 1, \ldots, m$, pick $r_j \in_R \mathbb{Z}_p$ and compute $c_j = \mathrm{com}_p(u^{2^j}, r_j)$.
2. For $j = 0, \ldots, m$, pick $\bar{a}_j, \bar{r}_j \in_R \mathbb{Z}_p$ and compute $\bar{c}_j = \mathrm{com}_p(\bar{a}_j, \bar{r}_j)$.
3. Compute new polynomial
$$\tilde{P}(X) = \sum_{j=0}^{m} \tilde{a}_j X^j = \sum_{i=0}^{M} a_i \prod_{j=0}^{m} (u^{2^j} X + \bar{a}_j)^{i[j]} X^{1-i[j]} \in \mathbb{Z}_p[X]$$
of degree m. For $j = 0, \ldots, m$, pick $\tilde{r}_j \in_R \mathbb{Z}_p$ and compute $\tilde{c}_j = \mathrm{com}_p(\tilde{a}_j, \tilde{r}_j)$.
4. For $j = 0, \ldots, m-1$, compute $\hat{a}_j = u^{2^j} \bar{a}_j$, pick $\hat{r}_j \in_R \mathbb{Z}_p$, and compute $\hat{c}_j = \mathrm{com}_p(\hat{a}_j, \hat{r}_j)$.
5. Compute $x = h(c, a_0, \ldots, a_M, c_1, \ldots, c_m, \bar{c}_0, \ldots, \bar{c}_m, \tilde{c}_0, \ldots, \tilde{c}_m, \hat{c}_0, \ldots, \hat{c}_{m-1})$.
6. For $j = 0, \ldots, m$, compute $\bar{a}'_j = \bar{a}_j + x u^{2^j}$.
7. For $j = 0, \ldots, m$, compute $\bar{r}'_j = \bar{r}_j + x r_j$.
8. For $j = 0, \ldots, m-1$, compute $\hat{r}'_j = \hat{r}_j + x r_{j+1} - b_j r_j$.
9. Compute $\tilde{r}' = \sum_{j=0}^{m} \tilde{r}_j x^j$.

Transcript:
$(c_1, \ldots, c_m, \bar{c}_0, \ldots, \bar{c}_m, \tilde{c}_0, \ldots, \tilde{c}_m, \hat{c}_0, \ldots, \hat{c}_{m-1}, \bar{a}'_0, \ldots, \bar{a}'_m, \bar{r}'_0, \ldots, \bar{r}'_m, \hat{r}'_0, \ldots, \hat{r}'_{m-1}, \tilde{r}')$
Verification:

1. Compute $x = h(c, a_0, \ldots, a_M, c_1, \ldots, c_m, \bar{c}_0, \ldots, \bar{c}_m, \tilde{c}_0, \ldots, \tilde{c}_m, \hat{c}_0, \ldots, \hat{c}_{m-1})$.
2. For $j = 0, \ldots, m$, check $c_j^x \bar{c}_j = \mathrm{com}_p(\bar{a}'_j, \bar{r}'_j)$.
3. For $j = 0, \ldots, m-1$, check $c_{j+1}^x \hat{c}_j = c_j^{\bar{a}'_j} \cdot \mathrm{com}_p(0, \hat{r}'_j)$.
4. Check
$$\prod_{j=0}^{m} \tilde{c}_j^{x^j} = \mathrm{com}_p\left(\sum_{i=0}^{M} a_i \prod_{j=0}^{m} \bar{a}'^{\,i[j]}_j x^{1-i[j]}, \tilde{r}' \right).$$

Fig. 1. Non-interactive version of the polynomial evaluation proof $NIZKP[(u, r) : c = \mathrm{com}_p(u, r) \wedge P(u) = 0]$ according to Bayer and Groth [3], using a slightly adjusted formal notation. We use $m = \lfloor \log M \rfloor = |M| - 1$ to denote the bit length of M minus 1 and a publicly known hash function $h(\cdot)$ with values in $\mathbb{Z}_p$ to compute the challenge x. The j-th bit of the binary representation of an index $i \in \{0, \ldots, M\}$ is denoted by $i[j] \in \{0, 1\}$, for $j = \{0, \ldots, m\}$. For reasons of convenience, let $c_0 = c$ and $r_0 = r$.

of $\mathbb{G}_q$ can be counted as elements of $\mathbb{Z}_p$, thus resulting in $2K + 2$ elements of $\mathbb{Z}_p$.[3] The proof generation requires $2K + 2$ exponentiations in $\mathcal{G}_p$ and $K(N+1)$ exponentiations in $\mathbb{G}_q$. Similarly, the verification requires $2K+1$ exponentiations in $\mathcal{G}_p$ and $K(N + 1)$ exponentiations in $\mathbb{G}_q$.

[3] The bandwidth requirements given in [2, Table 4] are clearly incorrect. It seems that the K elements of $\mathbb{G}_q$ have been counted falsely as elements of $\mathcal{G}_p$.

```
Public Input: $c = \mathrm{com}_p(u, r) \in \mathcal{G}_p$, $d = \mathrm{com}_q(v_1, \dots, v_N, s) \in \mathbb{G}_q$
Secret Input: $u, r \in \mathbb{Z}_p$, $v_1, \dots, v_N, s \in \mathbb{Z}_q$
Generation:
    1. Pick $\bar{u}, \bar{r} \in_R \mathbb{Z}_p$ and compute $\bar{c} = \mathrm{com}_p(\bar{u}, \bar{r})$.
    2. For $j = 1, \dots, K$,
        (a) pick $\bar{v}_{1,j}, \dots, \bar{v}_{N,j} \in_R \mathbb{Z}_q$ and compute $\bar{u}_j = h_1^{\bar{v}_{1,j}} \cdots h_N^{\bar{v}_{N,j}}$,
        (b) pick $\bar{r}_j \in_R \mathbb{Z}_p$ and compute $\bar{c}_j = \mathrm{com}_p(\bar{u}_j, \bar{r}_j)$,
        (c) pick $\bar{s}_j \in_R \mathbb{Z}_q$ and compute $\bar{d}_j = \mathrm{com}_q(\bar{v}_{1,j}, \dots, \bar{v}_{N,j}, \bar{s}_j)$.
    3. Compute $x = h(c, d, \bar{c}, \bar{c}_1, \dots, \bar{c}_k, \bar{d}_1, \dots, \bar{d}_k)$.
    4. Compute $\bar{u}' = \bar{u} - xu$ and $\bar{r}' = \bar{r} - xr$.
    5. For $j = 1, \dots, K$,
        (a) for $i = 1, \dots, N$, compute $\bar{v}'_{i,j} = \bar{v}_{i,j} - x[j]v_i$,
        (b) compute $\bar{r}'_j = \bar{r}_j - x[j] \cdot \mathrm{com}_q(\bar{v}'_{1,j}, \dots, \bar{v}'_{N,j}, r)$,
        (c) compute $\bar{s}'_j = \bar{s}_j - x[j]s$.
Transcript:
    $(\bar{c}, \bar{c}_1, \dots, \bar{c}_k, \bar{d}_1, \dots, \bar{d}_k, \bar{u}', \bar{r}', \bar{v}'_{1,1}, \dots, \bar{v}'_{N,K}, \bar{r}'_1, \dots, \bar{r}'_k, \bar{s}'_1, \dots, \bar{s}'_k)$
Verification:
    1. Compute $x = h(c, d, \bar{c}, \bar{c}_1, \dots, \bar{c}_k, \bar{d}_1, \dots, \bar{d}_k)$.
    2. Check $\bar{c} = c^x \cdot \mathrm{com}_p(\bar{u}', \bar{r}')$.
    3. For $j = 1, \dots, K$,
        (a) check $\bar{d}_j = d^{x[j]} \cdot \mathrm{com}_q(\bar{v}'_{1,j}, \dots, \bar{v}'_{N,j}, \bar{s}'_j)$,
        (b) compute $\bar{u}'_j = h_1^{\bar{v}'_{1,j}} \cdots h_N^{\bar{v}'_{N,j}}$, and check
            $\bar{c}_j = \begin{cases} \mathrm{com}_p(\bar{u}'_j, \bar{r}'_j), & \text{if } x[j] = 0, \\ c^{\bar{u}'_j} \cdot \mathrm{com}_p(0, \bar{r}'_j), & \text{if } x[j] = 1. \end{cases}$
```

Fig. 2. Non-interactive version of the representation proof $NIZKP[(u, r, v_1 \dots, v_N, s) : c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_q(v_1, \dots, v_N, s) \wedge u = h_1^{v_1} \cdots h_N^{v_N}]$ according to Au et al. [2], using a slightly adjusted formal notation. We use a publicly known hash function $h(\cdot)$ with values in $\mathbb{Z}_p$ to compute the challenge x. The j-th bit of the binary representation of x is denoted by $x[j] \in \{0, 1\}$ and $K < \log p$ is the security parameter.

3 Internet Elections with Everlasting Privacy

In this section, we present our new protocol for internet elections with everlasting privacy. We start with a discussion of the adversary model and the underlying trust assumptions. In Sects. 3.2 and 3.3, which constitutes the main contribution of this paper, we provide a detailed formal description of the protocol and analyse its security properties. A compact summary of the protocol is given in Fig. 3. We round off this section with a discussion of two important side aspects and corresponding protocol extensions.

3.1 Adversary Model and Trust Assumptions

We consider two types of adversaries with different capabilities and goals. An adversary of the first type acts at the present time, before or while an election takes place, whereas an adversary of the second type acts at any point in the future. Accordingly, we call them *present adversaries* and *future adversaries*.

The goal of present adversaries is to break the integrity or secrecy of the votes during an election, for example by submitting votes in the name of someone else or by linking votes to voters. We assume present adversaries to be polynomially bounded and thus incapable of solving mathematical problems such as computing discrete logarithms in large prime order groups or breaking cryptographic primitives such as contemporary hash functions. This implies that present adversaries cannot efficiently find valid openings of Pedersen commitments or valid proof transcripts for zero-knowledge proofs of knowledge without knowing the secret inputs. We also assume that the present adversary cannot control the machines used for vote casting.[4]

For a future adversary, the only goal is breaking the secrecy of the votes of an election that took place at the present time. To avoid the problem of estimating the available computational resources far in the future, we simply assume the strongest possible adversary, one with unlimited resources in terms of computational power and time. Although contemporary cryptography will be completely useless in the presence of such an adversary, the secrets hidden in perfectly hiding commitments or in perfect zero-knowledge proofs of knowledge will never be revealed, even if they were generated today.

From the point of view of the necessary communication infrastructure, the protocol requires an authentic channel between voter and election administration during the registration process. In the basic protocol version of Sect. 3.2, voters need to re-register in every new election, but we will show later how to circumvent this limitation. Furthermore, the protocol requires a broadcast channel with memory, for example in the form of a robust append-only public bulletin board collecting the entire election data. Finally, for sending their votes to the bulletin board, voters need access to an anonymous channel. We assume that no adversary is capable of intercepting and recording the whole traffic over this channel during an election and storing the data for future vote privacy attacks [1].

3.2 Protocol Description

The first step of the protocol is the voter registration before an election. To register, voter V picks a *private credential* $(\alpha, \beta) \in_R \mathbb{Z}_q^2$ at random and computes the *public credential* $u = h_1^\alpha h_2^\beta \in \mathbb{G}_q$. Note that the private credential is a representation of the public credential with respect to (h_1, h_2). Finally, the voter sends u over an authentic channel to the election administration.[5]

[4] We are aware that requiring a secure platform is a strong assumption. We do not explicitly address this problem in this paper, but our protocol allows voters at least to detect a compromised platform as long as they can read the bulletin board in a secure way.

[5] To ensure that u has been computed from fresh values (α, β), the voter could be asked to prove knowledge of (α, β) by computing $NIZKP[(\alpha, \beta) : u = h_1^\alpha h_2^\beta]$. As this is not an essential step for our protocol, we omit it in our presentation.

After the registration phase, the election administration defines the list $U = ((V_1, u_1), \ldots, (V_M, u_M))$ based on the electoral roll. Each pair $(V_i, u_i) \in U$ links a public credential to the corresponding voter. Next, the coefficients $A = (a_0, \ldots, a_M)$ of the polynomial $P(X) = \prod_{i=1}^{M}(X - u_i) \in \mathbb{Z}_p[X]$ are computed to allow voters creating the set membership proof during the vote casting phase. As the computation of those coefficients is quite expensive ($\frac{1}{2}M^2$ multiplications in $\mathbb{Z}_p$), it is performed by the election administration, possibly already during the registration phase in an incremental way. Note that the coefficients can be re-computed and verified by anyone, and voters can efficiently verify the inclusion of their public credential u by checking $P(u) = 0$. Finally, an independent *election generator* $\hat{h} \in \mathbb{G}_q$ is defined in some publicly reproducible way and $(U, A, \hat{h})$ is posted to the public bulletin board.

During the election, voters create their vote e by selecting their preferred election options. We do not further specify these options and their encoding, since our protocol does not impose any restrictions. Similarly, we do not discuss vote encryption, as this is a side aspect of the protocol and only affects fairness (see Sect. 3.4). To cast the vote, the voter computes the *election credential* $\hat{u} = \hat{h}^{\beta} \in \mathbb{G}_q$, a commitment $c = \text{com}_p(u, r)$ to the public credential, and a commitment $d = \text{com}_q(\alpha, \beta, s)$ to the private credential, where $r \in_R \mathbb{Z}_p$ and $s \in_R \mathbb{Z}_q$. Next, the voter generates three non-interactive zero-knowledge proofs. The first proof is a set membership proof $\pi_1 = \mathit{NIZKP}_e[(u, r) : c = \text{com}_p(u, r) \wedge P(u) = 0]$ proving that c is indeed a commitment to one of the public credentials in U. To prevent that a voter can take just any credential from U, the voter generates $\pi_2 = \mathit{NIZKP}_e[(u, r, \alpha, \beta, s) : c = \text{com}_p(u, r) \wedge d = \text{com}_q(\alpha, \beta, s) \wedge u = h_1^{\alpha} h_2^{\beta}]$ to prove knowledge of the representation of the committed value in c. Finally, the voter shows by a third proof $\pi_3 = \mathit{NIZKP}_e[(\alpha, \beta, s) : d = \text{com}_q(\alpha, \beta, s) \wedge \hat{u} = \hat{h}^{\beta}]$ that β used to build d and $\hat{u}$ is the same. All three proofs are linked to e. The ballot $B = (c, d, e, \hat{u}, \pi_1, \pi_2, \pi_3)$ consisting of the two commitments, the vote, the election credential, and the three proofs is posted over an anonymous channel to the bulletin board.

The final result of the election can be derived by anyone. For this, the list $\mathcal{B}$ of submitted ballots is retrieved from the bulletin board and the proofs included in each ballot $B \in \mathcal{B}$ are verified. Then duplicate votes are determined based on identical values $\hat{u}$ and conflicts are resolved according to some policy. As we will see in Sect. 4.3, verifying the proofs can be an expensive task for a large electorate. To accelerate the publication of the final result, the election administration may verify the proofs and mark invalid ballots as soon as they appear on the bulletin board. The correctness of the result can then still be checked by anyone.

3.3 Protocol Discussion

The correctness of the protocol is based on the fact that the public credential u can be seen as a perfectly hiding commitment to β and the election credential $\hat{u}$ as a perfectly binding commitment to β. For a present adversary not in possession

Registration (Voter):

1. Pick private credential $\alpha, \beta \in_R \mathbb{Z}_q$.
2. Compute public credential $u = h_1^\alpha h_2^\beta \in \mathbb{G}_q$.
3. Send u over an authentic channel to the election administration.

Election Preparation (Election Administration):

1. Define $U = ((V_1, u_1), \dots, (V_M, u_M))$ based on the electoral roll.
2. Compute coefficients $A = (a_0, \dots, a_M)$ of $P(X) = \prod_{i=1}^{M}(X - u_i) \in \mathbb{Z}_p[X]$.
3. Define election generator $\hat{h} \in \mathbb{G}_q$.
4. Post $(U, A, \hat{h})$ to the bulletin board.

Vote Casting (Voter):

1. Select vote e.
2. Compute election credential $\hat{u} = \hat{h}^\beta$.
3. Pick $r \in_R \mathbb{Z}_p$ and $s \in_R \mathbb{Z}_q$ and compute commitments $c = \mathrm{com}_p(u, r)$ and $d = \mathrm{com}_q(\alpha, \beta, s)$.
4. Compute the following non-interactive proofs:

$$\begin{aligned}
\pi_1 &= NIZKP_e[(u, r) : c = \mathrm{com}_p(u, r) \wedge P(u) = 0],\\
\pi_2 &= NIZKP_e[(u, r, \alpha, \beta, s) : c = \mathrm{com}_p(u, r) \wedge d = \mathrm{com}_q(\alpha, \beta, s)\\
&\qquad\qquad\qquad \wedge u = h_1^\alpha h_2^\beta],\\
\pi_3 &= NIZKP_e[(\alpha, \beta, s) : d = \mathrm{com}_q(\alpha, \beta, s) \wedge \hat{u} = \hat{h}^\beta].
\end{aligned}$$

5. Post ballot $B = (c, d, e, \hat{u}, \pi_1, \pi_2, \pi_3)$ to the bulletin board over an anonymous channel.

Public Tallying:

1. Retrieve the set $\mathcal{B}$ of all ballots from the bulletin board.
2. For each $B \in \mathcal{B}$, verify π_1, π_2, π_3.
3. Detect duplicate votes based on identical values $\hat{u}$ and resolve conflicts.
4. Compute final election result.

Fig. 3. Detailed protocol description.

of a private credential, there are two principle ways of creating a ballot that will be accepted in the final tally. First, the adversary may try to find (α', β') such that $u = h_1^{\alpha'} h_2^{\beta'}$ for some $u \in U$, which is equivalent to solve the discrete logarithm problem. Second, the adversary may try to fake a proof transcript without knowing such a pair (α', β'), but this is prevented by the computational soundness of the proofs. If the present adversary is an eligible voter in possession of a valid private credential, then trying to submit more than one ballot based on the same private credential will result in identical election credentials $\hat{u} = \hat{h}^\beta$. Without using the private credential, the voter is not more powerful than any other present adversary.

Everlasting Privacy. A ballot posted over an anonymous channel to the bulletin board contains no information for identifying the voter. Clearly, the future adversary will be able to determine β from $\hat{u}$ contained in the ballot, but knowing β, a suitable value α' can be found for every credential $u' \in U$ such that

$u' = h_1^{\alpha'} h_2^{\beta}$. Therefore, the adversary is unable to link $\hat{u}$ to u from knowing β. Additionally, the proofs π_1, π_2, and π_3 are perfectly zero-knowledge and therefore of no help. This implies that even the future adversary is unable to break the privacy of the vote. In other words, our protocol offers everlasting privacy.

Trust and Infrastructure Assumptions. Our protocol is based on two fundamental assumptions with regard to the available communication infrastructure. For silently casting a vote, voters require an anonymous channel, and for storing all their ballots, a robust public bulletin board must be available. Corresponding trust assumptions towards the developers and administrators of these systems are inevitable. However, no further trust assumptions are necessary in the basic version of our protocol. The election administration is the only authority involved, but the task of registering voters and publishing their public credentials can be verified by the voters themselves. The absence of further trusted authorities makes the overall election process extremely simple and allows an implementation of our protocol with almost no central infrastructure.

3.4 Extensions

In the basic version of the protocol as presented in Sect. 3.2, we have ignored some important aspects of real election systems. The following discussion of two of these aspects rounds off the description of our protocol.

Achieving Fairness. The protocol as presented is not fair. Fairness means that the published election data does not allow anyone to derive partial results during the election period. If fairness is a requirement, which is not always the case (especially in smaller elections with a very short election period), the protocol can be extended as follows. Instead of submitting the vote e in plaintext, the voter computes an encryption $E = enc_{pk}(e,t)$ using a randomized encryption scheme such as ElGamal or Paillier and generates a non-interative proof $\pi_4 = NIZKP[(e,t) : E = enc_{pk}(e,t)]$ of knowing the plaintext vote. The public encryption key pk is generated beforehand by a group of trusted authorities in a distributed manner. When the election period is over, the authorities post their shares of the corresponding private key to the bulletin board. The encrypted votes can then be decrypted by anyone.

Multiple Elections. If the protocol as presented so far is used for multiple elections, but without requiring voters to renew their credentials, then a future adversary will be able to link the votes from the same voter by uncovering the same value β from different election credentials. This does not create a direct link to the voter's identity, but it allows creating a kind of voter profile which will eventually leak information. To overcome this problem, the protocol must be modified to ensure that a single β is used for only one election. This can be achieved by extending the private and public credentials to $(\alpha, \beta_1, \ldots, \beta_L)$

and $u = h_1^{\alpha} h_2^{\beta_1} \dots h_{L+1}^{\beta_L}$, respectively, where L is the maximal number of elections the credentials can be used for. The corresponding commitment to the extended private credential, $d = \mathrm{com}_q(\alpha, \beta_1, \dots, \beta_L, s)$, implies that π_2 needs to be extended to a representation proof of size $N = L+1$. Finally, the modified election credential $\hat{u} = \hat{h}^{\beta_l}$ and an extended proof $\pi_3 = \mathit{NIZKP}_e[(\alpha, \beta_1, \dots, \beta_L, s) : d = \mathrm{com}_q(\alpha, \beta_1 \dots, \beta_L, s) \wedge \hat{u} = \hat{h}^{\beta_l}]$ are computed for $l = (\varepsilon \bmod L) + 1$, where $\varepsilon = 1, 2, \dots$ is the *election number* published beforehand by the elections administration.

4 Performance and Implementation

Given the complexity of both the set membership proof and the representation proof, we need to look closely at the computational resources required by our voting protocol. As we will see in this section, the performance is the most critical aspect of our protocol compared to others. We will first analyse the ballot size and estimate the total amount of election data that results from different electorate sizes. Then we discuss the cost of computation for creating a ballot and for verifying the entire election at the end of the election period.

4.1 Ballot Size

The size of a ballot in our protocol is mainly determined by the sizes of π_1 and π_2. In Sect. 2, we have given respective numbers. Recall that π_1 depends on M only, whereas π_2 depends on K and L. In Table 1 we recapitulate the number of group elements for $\mathcal{G}_p$, $\mathbb{Z}_p$, $\mathbb{G}_q$, and $\mathbb{Z}_q$ and sum them up. Since $\mathbb{Z}_p$ and $\mathbb{G}_q$ share the same modulo p, their elements are counted together. The table does not include corresponding numbers for the vote e and the proof of known plaintext in case of an encrypted vote.

To calculate the actual size of a ballot and estimate the total size of the election data, some of the system parameters need to be fixed. We consider the basic protocol version for a single election by setting $L = 1$. For a security parameter $K = 80$, we choose corresponding bit lengths $|q| = 160$ and $|p| = 1024$. In the light of today's recommendations for cryptographic parameters, these numbers may seem too small for offering appropriate security, but in the case of

Table 1. Ballot size as a function of M, K, and L (without encrypted vote and proof of known plaintext of the encrypted vote). Elements of $\mathbb{Z}_p$ and $\mathbb{G}_q$ are counted together.

Ballot Component	Elements of $\mathcal{G}_p$	Elements of $\mathbb{Z}_p, \mathbb{G}_q$	Elements of $\mathbb{Z}_q$
$c, d, \hat{u}$	1	2	–
π_1	$4\lfloor \log M \rfloor + 2$	$3\lfloor \log M \rfloor + 3$	–
π_2	$K + 1$	$2K + 2$	$K(L + 2)$
π_3	–	2	4
Entire Ballot	$4\lfloor \log M \rfloor + K + 4$	$3\lfloor \log M \rfloor + 2K + 9$	$KL + 2K + 3$

Table 2. Ballot size for different numbers of voters and parameters $K = 80$, $L = 1$, $|p| = 1024$, and $|q| = 160$.

$M = \|U\|$	Elements of $\mathcal{G}_p$	Elements of $\mathbb{Z}_p, \mathbb{G}_q$	Elements of $\mathbb{Z}_q$	Single Ballot	M Ballots
10	96	178	244	39.0 KB	0.4 MB
100	108	187	244	41.6 KB	4.1 MB
1'000	120	196	244	44.3 KB	43.2 MB
10'000	136	208	244	47.8 KB	466.5 MB
100'000	148	217	244	50.4 KB	4.8 GB
1'000'000	164	229	244	53.9 KB	51.4 GB

our protocol, the cryptography only needs to withstand vote integrity attacks by present adversaries during the election period. In other words, the cryptographic parameters can be chosen for an exceptionally short cryptoperiod.

Table 2 lists the results obtained for different electorates. The table shows that the size of a single ballot is certainly not a problem for voters to create and submit a ballot, even if M gets very large. On the other hand, if each voter submits a ballot, then the total size of the elections data sums up to more than 50 GB of data for one million voters. Given today's storage and network capacities, this amount of data should still be manageable by an ordinary server and communication infrastructure.

4.2 Cost of Computation: Ballot Generation

Let us now have a look at the cost of computation for generating a ballot. Corresponding computational resources need to be available to the voter for casting a vote. Again, generating the proofs π_1 and π_2 are the two critical tasks in this process. Recall from Sect. 2 that generating π_1 requires a logarithmic number of exponentiations in $\mathcal{G}_p$, but also a linearithmic number of multiplications in $\mathbb{Z}_p$. Since multiplications will become more expensive than exponentiations when M gets very large, they can not be neglected. Table 3 contains the number of critical operations in $\mathcal{G}_p$, $\mathbb{G}_q$, and $\mathbb{Z}_p$, and sums them up for the whole ballot. Again, we exclude the cost for encrypting the vote and generating a proof of known plaintext.

Table 3. Number of exponentiations and multiplications required to generate a single ballot (without encrypted vote and proof of known plaintext of the encrypted vote).

Ballot Component	Exponentiations in $\mathcal{G}_p$	Exponentiations in $\mathbb{G}_q$	Multiplications in $\mathbb{Z}_p$
$c, d, \hat{u}$	2	4	–
π_1	$8\lfloor\log M\rfloor + 4$	–	$2M\lfloor\log M\rfloor$
π_2	$2K + 2$	$K(L + 2)$	–
π_3	–	4	–
Entire Ballot	$8\lfloor\log M\rfloor + 2K + 8$	$KL + 2K + 8$	$2M\lfloor\log M\rfloor$

To estimate actual computation times for generating a ballot, we select the same parameters as in the previous subsection. Furthermore, we assume that the voter's computer is capable of calculating 350 exponentiations per second in $\mathcal{G}_p$, 2'000 exponentiations per second in $\mathbb{G}_q$, and 200'000 multiplications per second in $\mathbb{Z}_p$. We derive these numbers from performance tests in Java on a MacBook Pro with a 2.7 GHz Intel Core i7 processor (16GB RAM, OS X Yosemite 10.10.2, JRE 8, standard `BigInteger` class, single-threaded). The results of our analysis are shown in Table 4. The estimated cost of computation for generating a single ballot turns out to be perfectly acceptable for a medium-sized or even a large electorate. Only when M gets very large (e.g. more than 100'000 voters), the ballot generation gets delayed inappropriately. This is roughly the threshold when the multiplications start to dominate the exponentiations.

Table 4. Cost of ballot generation for different numbers of voters and parameters $K = 80$, $L = 1$, $|p| = 1024$, and $|q| = 160$. The time estimates are based on 350 exponentiations per second in $\mathcal{G}_p$, 2'000 exponentiations per second in $\mathbb{G}_q$, and 200'000 multiplications per second in $\mathbb{Z}_p$.

$M = \|U\|$	Exponentiations in $\mathcal{G}_p$	Exponentiations in $\mathbb{G}_q$	Multiplications in $\mathbb{Z}_p$	Estimated Time (Single Ballot)
10	192	248	60	0.7 s
100	216	248	1'200	0.7 s
1'000	240	248	18'000	0.9 s
10'000	272	248	260'000	2.2 s
100'000	296	248	3'200'000	17.0 s
1'000'000	328	248	40'000'000	3.4 min

4.3 Cost of Computation: Verification

The most expensive computational task of our protocol is clearly the public tallying, which involves the verification of all proofs included in the ballots. The values shown in Table 5 summarize the number of critical operations in $\mathcal{G}_p$, $\mathbb{G}_q$, and $\mathbb{Z}_p$ for verifying a single ballot. For very large values of M, the most expensive operations are again the $3M$ multiplications in $\mathbb{Z}_p$, which is why they cannot be neglected. As before, the results shown in the table do not contain additional operations for verifying the proof of known plaintext in case of an encrypted vote. Note that proper verification requires checking that the values included in the proof transcripts are elements of corresponding sets. In case of $\mathcal{G}_p$ and $\mathbb{G}_q$ this may require additional exponentiations. We omit them here to be consistent with the results given in [2,3].

To conclude our performance analysis, we adopt the system parameters and the assumptions with regard to the available computation power from the previous subsection. The resulting values for different electorate sizes are shown in

Table 5. Number of exponentiations and multiplications required to verify a single ballot (without proof of known plaintext of the encrypted vote).

Ballot Component	Exponentiations in $\mathcal{G}_p$	Exponentiations in $\mathbb{G}_q$	Multiplications in $\mathbb{Z}_p$
π_1	$6\lfloor\log M\rfloor + 6$	–	$2M$
π_2	$2K + 1$	$K(L + 2)$	–
π_3	–	6	–
Total	$6\lfloor\log M\rfloor + 2K + 7$	$KL + 2K + 6$	$2M$

Table 6. Cost of ballot verification for different numbers of voters and parameters $K = 80$, $L = 1$, $|p| = 1024$, and $|q| = 160$. The time estimates are based on 350 exponentiations per second in $\mathcal{G}_p$, 2'000 exponentiations per second in $\mathbb{G}_q$, and 200'000 multiplications per second in $\mathbb{Z}_p$.

$M = \lvert U\rvert$	Exponentiations in $\mathcal{G}_p$	Exponentiations in $\mathbb{G}_q$	Multiplications in $\mathbb{Z}_p$	Estimated Time (Single Ballot)	Estimated Time (M Ballots)
10	185	166	30	0.6 s	6.1 s
100	203	166	300	0.7 s	1.1 min
1'000	221	166	3'000	0.7 s	12.2 min
10'000	245	166	30'000	0.9 s	2.6 h
100'000	263	166	300'000	2.3 s	64.8 h
1'000'000	287	166	3'000'000	15.9 s	4417.5 h

Table 6. By multiplying the time estimates for verifying a single ballot by the total number of votes, we obtain time estimates for the full verification process.

From the given results, we conclude again that our protocol works reasonably well for a medium-sized or even a large electorate. Note that the verification of the ballots can already start during the vote casting phase, and since it can be executed in parallel, there is a huge potential for distributing the total amount of work to arbitrarily many and possibly more powerful machines. While this is in principle a solution for reducing the 4'400 hours of computation for an election with one million ballots to a more reasonable value, it restricts somewhat the idea of a *public* tallying process.

4.4 Implementation and Optimizations

In course of developing the protocol presented in this paper, we implemented both the set membership and the representation proof in UniCrypt [17]. This is an open-source Java library developed for the purpose of simplifying the implementation of cryptographic voting protocols.[6] The library consist of a mathematical and a cryptographic layer. The two implemented proofs extend the proofsystem package, which is a central component of the cryptographic layer. The same package also contains classes for generating all sorts of preimage or equality proofs, which we need for computing π_3. Other packages in the cryptographic layer provide implementations of Pedersen commitments and various encryption schemes. The library provides therefore the full functionality for a straightforward implementation of our protocol.

In order to check the accuracy of the calculated time estimates of the previous subsections, we used UniCrypt to generate and verify ballots for different electorate sizes and measured the times of computation. The results of these measurements are shown in Table 7. We used the same machine for the tests as in the previous subsection, a MacBook Pro with a 2.7 GHz Intel Core i7 processor, and the current UniCrypt version from the project's development branch on April 1, 2015. In general, the measured running times are quite consistent with the time estimates from the previous section, for example 18.2 instead of 17.0 seconds for generating a ballot with 100'000 voters. This difference can be explained by the overhead for other less expensive operations and for Java's memory and object management. Note that for 1'000'000 voters, the actual running times are even slightly better than the estimates (3.3 instead of 3.4 min). An explanation for this is the fact, that $2M\lfloor\log M\rfloor$ is an upper approximation for the number of multiplications in $\mathbb{Z}_p$.

To conclude the discussion about our implementation and the results of the performance analysis, we need to stress that the prototype implementation has not been optimized in any way. To speed up the ballot generation, we may pre-compute the proofs in a background process of the vote preparation software, and we may distribute the computations to all available cores of the given machine, or to the machine's graphics processing unit. In the final verification of all ballots, the potential of executing tasks in parallel—possibly on many different machines—is even higher. Furthermore, techniques like multi-exponentiation

Table 7. Actual running times for generating and verifying a single ballot using the UniCrypt library.

$M = \lvert U\rvert$	Ballot generation	Ballot verification
10	1.3 s	0.9 s
100	1.4 s	1.0 s
1'000	1.6 s	1.1 s
10'000	3.0 s	1.3 s
100'000	18.2 s	2.9 s
1'000'000	3.3 min	18.8 s

[6] UniCrypt is publicly available on GitHub under a dual AGPLv3/commercial licence, see https://github.com/bfh-evg/unicrypt.

and fixed-base exponentiation may bring considerable performance improvements, especially for small elections, where the exponentiations predominate the multiplications. For very large elections, we should consider replacing the set membership proof as described in this paper by an approach by Brands et al. [5], which requires $8\sqrt{M}$ exponentiations but only $2M + 8\sqrt{M}$ multiplications for generating a proof.

5 Conclusion

In this paper, we have introduced a new approach for a cryptographic voting protocol. Its underlying mechanism is very different compared to mainstream approaches based on mixing and homomorphic tallying. In our protocol, the distinction between valid and invalid ballots is strictly based on perfectly hiding commitments and perfect zero-knowledge proofs of knowledge. This prevents computationally bounded adversaries from submitting illegitimate votes during the election. At the same time, even a computationally unbounded adversary in the future will never be able to link votes to voters. Our protocol offers therefore a solution to the everlasting privacy problem. Compared to other protocols offering everlasting privacy, we do not require any trusted authorities. This makes our protocol particularly attractive for straightforward implementation in a practical system. The relatively high computational costs for generating and verifying the ballots is a clear disadvantage of our approach, but we have demonstrated that with today's technology, this is only a drawback for very large electorates.

Acknowledgments. We thank the anonymous reviewers for their thorough reviews and appreciate their comments and suggestions. This research has been supported by the Swiss National Science Foundation (project No. 200021L_140650).

References

1. Arapinis, M., Cortier, V., Kremer, S., Ryan, M.: Practical everlasting privacy. In: Basin, D., Mitchell, J.C. (eds.) POST 2013 (ETAPS 2013). LNCS, vol. 7796, pp. 21–40. Springer, Heidelberg (2013)
2. Au, M.H., Susilo, W., Mu, Y.: Proof-of-knowledge of representation of committed value and its applications. In: Steinfeld, R., Hawkes, P. (eds.) ACISP 2010. LNCS, vol. 6168, pp. 352–369. Springer, Heidelberg (2010)
3. Bayer, S., Groth, J.: Zero-knowledge argument for polynomial evaluation with application to blacklists. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 646–663. Springer, Heidelberg (2013)
4. Brands, S.: Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. MIT Press, Cambridge (2000)
5. Brands, S., Demuynck, L., De Decker, B.: A practical system for globally revoking the unlinkable pseudonyms of unknown users. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 400–415. Springer, Heidelberg (2007)

6. Buchmann, J., Demirel, D., van de Graaf, J.: Towards a publicly-verifiable mix-net providing everlasting privacy. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 197–204. Springer, Heidelberg (2013)
7. Camenisch, J.L., Chaabouni, R., Shelat, A.: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
8. Camenisch, J.L., Stadler, M.A.: Efficient group signature schemes for large groups. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997)
9. Canard, S., Traoré, J.: List signature schemes and application to electronic voting. In: Augot, D., Charpin, P., Kabatianski, G. (eds.) WCC'03, 3rd International Workshop on Coding and Cryptography, Versailles, France, pp. 81–90 (2003)
10. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol. **1**(1), 65–75 (1988)
11. Demirel, D., Henning, M., van de Graaf, J., Ryan, P.Y.A., Buchmann, J.: Prêt à voter providing everlasting privacy. In: Heather, J., Schneider, S., Teague, V. (eds.) Vote-ID 2013. LNCS, vol. 7985, pp. 156–175. Springer, Heidelberg (2013)
12. Demirel, D., van de Graaf, J., Araújo, R.: Improving Helios with everlasting privacy towards the public. In: Halderman, J.A., Pereira, O. (eds.) Electronic Voting Technology Workshop/Workshop on Trustworthy Elections, EVT/WOTE 2012, Bellevue, USA (2012)
13. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
14. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Seberry, J., Zheng, Y. (eds.) ASIACRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1992)
15. Groth, J.: Efficient maximal privacy in boardroom voting and anonymous broadcast. In: Juels, A. (ed.) FC 2004. LNCS, vol. 3110, pp. 90–104. Springer, Heidelberg (2004)
16. Kiayias, A., Yung, M.: Self-tallying Elections and Perfect Ballot Secrecy. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 141–158. Springer, Heidelberg (2002)
17. Locher, P., Haenni, R.: A lightweight implementation of a shuffle proof for electronic voting systems. In: Plödereder, E., Grunske, L., Schneider, E., Ull, D. (eds.) INFORMATIK 2014. Lecture Notes in Informatics, Stuttgart, Germany, pp. 1391–1400. Gesellschaft für Informatik, Bonn (2014)
18. Moran, T., Naor, M.: Receipt-free universally-verifiable voting with everlasting privacy. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 373–392. Springer, Heidelberg (2006)
19. Moran, T., Naor, M.: Split-ballot voting: everlasting privacy with distributed trust. In: Ning, P., de Capitani di Vimercati, S., Syverson, P. (eds.) CC 2007, 14th ACM Conference on Computer and Communications Security, Alexandria, USA, pp. 246–255 (2007)
20. Moran, T., Naor, M.: Split-ballot voting: everlasting privacy with distributed trust. ACM Trans. Inf. Syst. Secur. **13**(2), 16:1–16:43 (2010)
21. van de Graaf, J.: Voting with unconditional privacy by merging Prêt à Voter and PunchScan. IEEE Trans. Info. Forensics Secur. **4**(4), 674–684 (2009)