How to prove the validity of a complex ballot encryption to the voter and the public[☆]

Rui Joaquim

University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
6, rue Richard Coudenhove-Kalergi, L-1359, Luxembourg

Abstract

One crucial aspect of any verifiable electronic voting system that uses encryption is the proof that the vote encryption is well-formed, i.e. the proof that the vote encryption encrypts a valid vote accordingly to the race specification. It makes no sense accepting an encrypted vote if, at the end of the election, the vote cannot be included in the tally because it is badly formed.

Proving the validity of a complex vote encryption, without revealing the vote, is a hard problem. This paper first contribution addresses exactly that problem and provides a set of new constructions to create a vote encryption and the corresponding public proof of validity for several types of complex ballots ($[k_{min}, k_{max}]$-out-of-n approval, weighted and ranked ballots. The second contribution is a technique that allows to create a single, constant size, verification code for a ballot containing one or several races of any mix of the race types considered. With this single verification code the voter can verify that her vote was casted-as-intended.

Moreover, our constructions can be tuned for either mix net or homomorphic tallying and support both types of tallying in the same multi-race ballot.

Keywords: Verifiable vote encryption, complex ballots, electronic voting, cast-as-intended verification, verification codes

1. Introduction

Elections are essential for a democratic society as they are the basis of our democracies. Therefore, in order to allow free voting to everyone, it is critical that the election system ensures the correctness of the elections' results while preserving the voter's privacy.

An election system should not only provide proofs that the votes are counted correctly but also that they capture the intention of the voters, preserving the anonymity of the votes. While the techniques to provide an electronic verifiable tally are well established,

[☆]This work was supported by the Fonds National de la Recherche, Luxembourg (INTER/SNF/11/11).
Email address: rui.joaquim@uni.lu (Rui Joaquim)

the same cannot be said about the techniques to prove that a vote encryption performed by an untrusted machine encrypts the voter's vote intention.

This paper describes a set of new constructions inspired on the MarkPledge family of voter verifiable vote encryption protocols [38, 4, 32]. Despite significant differences at the technical details, at a high level, a MarkPledge voter verification protocol is a slightly modified version of a fairly straightforward zero-knowledge proof, in which the voter chooses the challenge and performs a simple string comparison to verify that her vote was encrypted correctly. All equations necessary for soundness are publicly verifiable, thus can be verified by any interested party, including the voter, using an independent machine.

To our knowledge, our constructions are the first to offer a highly sound voter verification mechanism for complex ballots with a constant size vote verification code. The use of a single verification code requires the voter to have access to a trusted piece of software/hardware to help her compute the verification code from the public election data and hers secret selections. We also show how to create a more traditional code voting receipt (with one verification code per candidate) from our constructions. This receipt allows the voter to verify the correct vote encryption without the need of a trusted device, although, like other vote confirmation techniques that use one verification code for each vote selection, it rapidly becomes unusable with the increase of the ballot complexity.

In this work we only address the problem of creating a voter verifiable vote encryption and prove it to be honest-verifier zero-knowledge. We do not propose any full vote protocol, although the adaptation to the coercion resistant protocol in [3] or the simplified vote protocol described in [38] is straightforward.

The new constructions proposed in this paper are very flexibly and allow to support in an uniform way several types of complex ballots ($[k_{min}, k_{max}]$-out-of-n, weighted, ranked and multi-race ballots). It can be tuned for either mix net or homomorphic tallying, allowing even the use of both types of tallying on different races in a single multi-race ballot. This flexibility can be very useful in anonymous surveys where some answers must be correlated.

One crucial aspect of our vote encryption verification constructions, and of electronic voting in general, is the proof that the vote encryption is well-formed, i.e. the proof that the vote encryption encrypts a valid vote accordingly to the race specification. It makes no sense in verifying a vote encryption if, at the end of the election, the vote cannot be included in the tally because it is badly formed.

We solve the vote encryption well-formness verification problem by: i) creating the vote encryption from a verifiable shuffle of a set of encryptions of known messages; and ii) whenever necessary, using additional zero-knowledge proofs of compliance to the vote specification. To our knowledge this approach is new and completely different from the previous ones that impose a certain mathematical structure to the vote plaintext construction, e.g. [28].

The remainder of the paper is organized as follows: in the next section presents the related work. Then, section 3 gives the background necessary for our constructions. Section 4 details the constructions for mix net tallying and section 5 describe the constructions for homomorphic tallying. Section 6 shows how to extract a MarkPledge style voter verifiable receipt from our complex ballot encryption constructions and section 7 presents the conclusions.

2. Related work

In 2004, with the work of Chaum [14] and Neff [38] a new paradigm in electronic voting research has emerged: End-to-End (E2E) voting systems. The goal of E2E voting systems is to develop voting systems with both voter cast-as-intended and universal counted-as-cast verifications.

Chaum, in [14], addresses the voter cast-as-intended verification using visual cryptography [37]. His proposal uses special printers that print a human readable vote encryption on two overlapped transparent sheets.

Later the Prêt-à-Voter [20, 43, 48] and the Punchscan [15, 40] systems simplified the original Chaum's setup using pre-printed ballots. Adida and Rivest proposed the Scratch-and-Vote system [5], based on Prêt-à-Voter, which uses scratch strips to allow off-line ballot verification. In 2007, Moran and Naor [36] proposed an everlasting private[1] system based on Punchscan. The E2E ideas proposed in Punchscan served also as inspiration for the development of the optical scanner based E2E verifiable voting system Scantegrity [17], which was improved in Scantegrity II [16] with the incorporation of vote confirmation codes. All these protocols, have pre-printed ballots, which allow for a ballot auditing process before the election to minimize the risk of using bogus ballots.

The ideas of the above described poll station E2E systems where later adapted to the Internet voting scenario in the Pretty Good Democracy (PGD) [44] and the Scratch, Click and Vote (SCV) [33] voting systems. PGD achieves E2E verifiability by enhancing a code voting protocol inspired by some ideas used in the Scantegrity II and Prêt-à-Voter systems. PGD was later enhanced to support expressive voting schemes in which the voter lists the candidates in order of preference [29]. SCV uses the voter cast-as-intended verification ideas of Punchscan, Prêt-à-Voter and ThreeBallot[2].

Neff's proposal [38] (also known as MarkPledge) uses a quite different approach. It codifies a verification code for each candidate into a set of 1-out-of-2 cut and choose proofs of encryption. The verification codes are then computed from each set of encryptions and a vote receipt with a random looking verification code for each candidate is created. This technique achieves a soundness of $1/2^{\alpha}$ for a verification code with a length of α bits. Neff's proposal main disadvantages are: i) the high computational costs; and ii) the complex vote protocol, which forces the voter to perform a complex challenge-response style protocol with the voting machine, at the voting booth. The usability issues were addressed in [3, 31] and the efficiency issues in [4, 32]. Moran and Naor presented a MarkPledge like system with "everlasting privacy" [35] by replacing the vote encryptions with bit commitments.

A completely different voter verification approach was proposed by Benaloh in [8, 9]. Benaloh's proposal separates the vote encryption from the vote casting process. There is a vote preparation machine that encrypts the vote but does not cast it, instead it delivers the vote encryption to the voter in a paper support. The voter can then choose to cast the encrypted vote or to verify it by asking the vote preparation machine to decrypt it. In this solution the voter can verify as many vote encryptions as she wants until she

[1] Moran and Naor define that a system has "everlasting" privacy if a computational unbounded adversary gains no information about specific votes from observing the protocol's output.

[2] The ThreeBallot system is a paper based voter verifiable voting system proposed by Rivest [41, 42] which "does not" use any cryptography.

gains confidence in the vote preparation machine. This approach has the advantage that it is easy to run independent tests to verify that the vote preparing machine is working properly. The ideas of Benaloh's work were used by the VoteBox [45, 39] and Helios [1] voting systems (the latter is a remote Internet voting system).

Another approach to the cast-as-intended verification problem was proposed by Araújo et al. [6]. They use a different verification approach in which the voter takes home a random set of already casted ballots as a receipt, i.e. each voter verifies votes from other voters and not her own vote.

In recent years there has been several proposals for E2E verifiable voting systems by combining the code voting approach to an universal verifiable vote tally protocols [44, 34, 30, 27].

Previously to this work, the "only usable way" to support complex ballots encryptions in remote E2E voting systems was using the Benaloh approach.

Usually, to prove a valid vote encryption, the protocol designers use disjunctive proofs of plaintext equality, e.g. in [2], or more elaborated proofs of structure of the plaintext vote [28] which usually restrict its applications to specific types of complex ballots. However, none of these approaches allow a MarkPledge style direct vote encryption verification.

3. Preliminaries

Our work relies on the *independent generators assumption* (Assumption 1), on the ElGamal cryptosystem [25] and its homomorphic properties, on verifiable shuffles and several zero-knowledge proofs of knowledge. The setup for our constructions requires a cyclic group G_q of prime order q, e.g. a subgroup of $\mathbb{Z}_p^*$, where p and q are large primes such that $q|p-1$. We say that for any given set of generators $(g_1, ..., g_m)$ and any element $y \in \mathrm{G}_q$, $(x_1, ..., x_m) \in \mathbb{Z}_q^m$ is a representation of y with respect to the g_i's if $y = \prod_{i=1}^{m} g_i^{x_i}$.

Assumption 1. (independent generators assumption, see [18, 12]) *No probabilistic polynomial-time algorithm, on input q and a randomly chosen, polynomial-sized tuple of generators $(g_1, ..., g_m)$, can output with non-negligible probability an element $h \in G_q$ and two different representations of h with respect to some of the g_i's.*

3.1. The ElGamal cryptosystem

The ElGamal cryptosystem operates over a cyclic group, e.g. G_q.[3] Let both primes p, q and a generator g of G_q be the public parameters of the system. The ElGamal key pair consists of a private key s and the corresponding public key $h = g^s \mod p$. The private key s is a randomly chosen integer such that $0 < s < q$. The ElGamal message space is $M = \mathrm{G}_q$, the ciphertext space is $C = \mathrm{G}_q$ x G_q and the encryption and decryption algorithms are described below.

. The ElGamal encryption algorithm $\mathcal{E}$ is defined as follows:

$$r \xleftarrow{R} \mathbb{Z}_q^*,\ c = \langle \alpha, \beta \rangle = \langle g^r \mod p, h^r m \mod p \rangle = \mathcal{E}_h(m, r)$$

[3] It is also possible to setup the ElGamal cryptosystem over elliptic curves, but, for simplicity, the description uses the setup over the large prime order cyclic group of integers G_q.

. The ElGamal decryption algorithm D is defined as follows:

$$m = \frac{\beta}{\alpha^s} \mod p = \mathcal{D}_s(c)$$

In e-voting protocols it is usual to share the election private key among a set $\mathcal{T}$ of trustees such that at the end of the election a subset of $\mathcal{T}$ must cooperate to decrypt the votes. The details of distributed key generation and decryption algorithms for the ElGamal cryptosystem can be found in [22, 11].

ElGamal homomorphic properties:. ElGamal is an homomorphic cryptosystem, i.e. there are known algebraic operations performed on the ciphertexts that are equivalent to other (possibly the same) algebraic operations performed on the plaintexts (messages).

. Multiplicative homomorphism

$$\begin{aligned} \mathcal{E}_h(m_1,\ r_1) \otimes \mathcal{E}_h(m_2,\ r_2) &= \langle g^{r_1},\ h^{r_1} m_1 \rangle \otimes \langle g^{r_2},\ h^{r_2} m_2 \rangle \\ &= \langle g^{r_1} g^{r_2},\ h^{r_1} m_1 h^{r_2} m_2 \rangle \\ &= \langle g^{r_1+r_2},\ h^{r_1+r_2} m_1 m_2 \rangle \\ &= \mathcal{E}_h(m_1 m_2 \mod p,\ r_1 + r_2 \mod q) \end{aligned} \tag{1}$$

. Exponentiation homomorphism

$$\begin{aligned} \mathcal{E}_h(m,\ r)^n &= \langle g^r,\ h^r m \rangle^n = \langle (g^r)^n,\ (h^r m)^n \rangle \\ &= \langle (g^r)^n,\ (h^r)^n m^n \rangle = \langle g^{rn},\ h^{rn} m^n \rangle \\ &= \mathcal{E}_h(m^n \mod p,\ rn \mod q) \end{aligned} \tag{2}$$

The above described homomorphisms can be turned into additive and multiplicative exponent homomorphisms when using exponential ElGamal [22]. In exponential ElGamal the message space is $M' = \mathbb{Z}_q$, and the encryption operation requires the use of a generator G of G_q (can be g). The exponential ElGamal encryption of a message $m \in M'$ is $\mathcal{E}(G^m, r)$ and the additive and multiplicative exponent homomorphisms are defined as follows:

. Additive exponent homomorphism

$$\begin{aligned} \mathcal{E}_h(G^{m_1},\ r_1) \otimes \mathcal{E}_h(G^{m_2},\ r_2) &= \langle g^{r_1},\ h^{r_1} G^{m_1} \rangle \otimes \langle g^{r_2},\ h^{r_2} G^{m_2} \rangle \\ &= \langle g^{r_1} g^{r_2},\ h^{r_1} G^{m_1} h^{r_2} G^{m_2} \rangle \\ &= \langle g^{r_1+r_2},\ h^{r_1+r_2} G^{m_1+m_2} \rangle \\ &= \mathcal{E}_h(G^{m_1+m_2 \mod q} \mod p,\ r_1 + r_2 \mod q) \end{aligned} \tag{3}$$

. Multiplicative exponent homomorphism

$$\begin{aligned} \mathcal{E}_h(G^m,\ r)^n &= \langle g^r,\ h^r G^m \rangle^n = \langle (g^r)^n,\ (h^r G^m)^n \rangle \\ &= \langle (g^r)^n,\ (h^r)^n (G^m)^n \rangle = \langle g^{rn},\ h^{rn} G^{mn} \rangle \\ &= \mathcal{E}_h(G^{mn \mod q} \mod p,\ rn \mod q) \end{aligned} \tag{4}$$

3.2. Re-encryption, verifiable shuffles and mix nets

Given a ciphertext $\mathcal{E}_h(m,\ r)$ it is possible of obtain a different encryption of m simply by applying a component-wise multiplication ($\otimes$):

$$\mathcal{E}_h(m,\ r) \otimes \langle g^{r'},\ h^{r'} \rangle = \langle g^{r+r'},\ h^{r+r'} m \rangle = \mathcal{E}_h(m,\ r + r')$$

A re-encryption shuffle inputs a set of ciphertexts $C = C_1, ..., C_N$ and outputs a set of ciphertexts $C' = C'_1, ..., C'_N$ with a permutation π of the same plaintexts, i.e. a permuted re-encryption of C. A verifiable re-encryption shuffle outputs a proof that C' contains a permutation π of the plaintexts in C, without revealing it.

A re-encryption mix net consists on a set of peers (called mixers) that perform individual re-encryption shuffles in sequence. If at least one mixer is honest the final output of the mix net will not reveal the link between the input ciphertexts and the output ciphertexts. Examples of efficient shuffles/mix nets can be found in [47, 7].

3.3. Zero-knowledge proofs

To prove the correctness of a protocol it is usual to use zero-knowledge proofs. The constructions proposed in this paper use the below described interactive (honest-verifier) zero-knowledge proofs, which can be made non-interactive using the Fiat-Shamir heuristic [26].

3.3.1. Proof of knowledge of a discrete logarithm

In the proof by Schnorr [46] the prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ that, for a given message v and a generator g of G_q, she knows x such that $\alpha = g^x \ mod\ p$.

$$ZKPK\left[x : \alpha = g^x\right]$$

The proof unfolds as follows:
Public input: g, α
$\mathcal{P}$ private input: x such that $\alpha = g^x \mod p$

1. $\mathcal{P}$ chooses $w \xleftarrow{R} \mathbb{Z}_q$ and sends $a = g^w \mod p$ to $\mathcal{V}$.
2. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ sends $r = (w + cx) \mod q$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies $g^r \equiv a\alpha^c \mod p$

3.3.2. Proof of knowledge of a representation

The Schnorr proof above can be generalized to provide a proof of knowledge of a representation $(x_1, ..., x_m)$ of a value $\alpha \in \mathrm{G}_q$, with respect to a set of generators $(g_1, ..., g_m)$ [10].

$$ZKPK\left[x_1, ..., x_m : \alpha = \prod_{i=1}^{m} g_i^{x_i}\right]$$

The proof unfolds as follows:
Public input: g, α
$\mathcal{P}$ private input: $(x_1, ..., x_m)$ such that $\alpha = \prod_{i=1}^{m} g_i^{x_i} \mod p$

1. $\mathcal{P}$ chooses $w_1, ..., w_m \xleftarrow{R} \mathbb{Z}_q$ and sends $a = \prod_{i=1}^{m} g_i^{w_i} \mod p$ to $\mathcal{V}$.
2. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ sends $r_i = (w_i + cx_i) \mod q$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies $\prod_{i=1}^{m} g_i^{r_i} \equiv a\alpha^c \mod p$

3.3.3. Proof of equality of two discrete logarithms

A proof that two discrete logarithms are equal can be achieved by executing in parallel two zero-knowledge proofs of knowledge of a discrete logarithm [19].

$$ZKPK\left[x : \log_g \alpha = x \ \wedge \ \log_h \beta = x\right]$$

which we can also represent in the more compact form:

$$ZKPK\left[x : \alpha = g^x \ \wedge \ \beta = h^x\right]$$

The proof unfolds as follows: Public input: g, h, α, β
$\mathcal{P}$ private input: x such that $\alpha = g^x \mod p$ and $\beta = h^x \mod p$

1. $\mathcal{P}$ chooses $w \xleftarrow{R} \mathbb{Z}_q$ and sends $a = g^w \mod p$ and $b = h^w \mod p$ to $\mathcal{V}$.
2. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ sends $r = (w + cx) \mod q$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies $g^r \equiv a\alpha^c \mod p \ \wedge \ h^r \equiv b\beta^c \mod p$

This proof can be used to prove that an ElGamal ciphertext $c = \langle \alpha, \beta \rangle$ encrypts a message m, i.e. $ZKPK\left[x : \alpha = g^x \ \wedge \ \beta/m = h^x\right]$.

3.3.4. Proof of knowledge of an ElGamal plaintext representation

Here we present a proof of knowledge of an ElGamal plaintext representation with respect to a set of generators $(g_1, ..., g_m)$ and a ciphertext $c = \langle \alpha, \beta \rangle$.

$$ZKPK\left[x_0, x_1, ..., x_m : \alpha = g^{x_0} \ \wedge \ \beta / \prod_{i=1}^{m} g_i^{x_i} = h^{x_0}\right]$$

In our constructions description we will use the following shorter notation:

$$ZKPK\left[x_0, x_1, ..., x_m : \mathcal{E}_h\left(\prod_{i=1}^{m} g_i^{x_i}, x_0\right)\right]$$

To our knowledge, such proof is not described in the literature, thus we proof it to be special honest-verifier zero-knowledge [21], which suffices for our application. The proof unfolds as follows:
Public input: g, h, α, β
$\mathcal{P}$ private input: $x_0, (x_1, ..., x_m)$ such that $\alpha = g^{x_0} \mod p$ and $\beta = h^{x_0} \prod_{i=1}^{m} g_i^{x_i} \mod p$.

1. $\mathcal{P}$ chooses $w_0, ..., w_m \xleftarrow{R} \mathbb{Z}_q$ and sends $a = g^{w_0} \mod p$, $b = h^{w_0} \prod_{i=1}^{m} g_i^{w_i} \mod p$ to $\mathcal{V}$.
2. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.

3. $\mathcal{P}$ sends $r_0, ..., r_m : r_i = (w_i + cx_i) \mod q$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies $g^{r_0} \equiv a\alpha^c \mod p \ \wedge \ h^{r_0} \prod_{i=1}^{m} g_i^{r_i} \equiv b\beta^c \mod p$

Theorem 1. *The above protocol is special honest-verifier zero-knowledge for proving knowledge of an ElGamal plaintext representation $(x_1^*, ..., x_m^*)$ of a value $v \in G_q$ with respect to a set of generators $(g_1, ..., g_m)$.*

Proof. The protocol is complete because if both $\mathcal{P}$ and $\mathcal{V}$ are honest it will always hold that $g^{r_0} \equiv g^{w_0+cx_0} \equiv g^{w_0} g^{cx_0} \equiv a\alpha^c \mod p$ and that $h^{r_0} \prod_{i=1}^{m} g_i^{r_i} \equiv h^{w_0+cx_0} \prod_{i=1}^{m} g_i^{w_i+cx_i} \equiv h^{w_0} \prod_{i=1}^{m} g_i^{w_i} h^{cx_0} \prod_{i=1}^{m} g_i^{cx_i} \equiv b\beta^c \mod p$.

To prove soundness we show that if $\mathcal{P}$ can answer to two different challenges (c, c') it is able to compute values x_i^*, such that $v = \prod_{i=1}^{m} g_i^{x_i^*}$ and $\mathcal{E}_h(v, x_0) = \langle \alpha, \beta \rangle$, thus it knows a representation of v with respect to the set of generators $(g_1, ..., g_m)$.

If $\mathcal{P}$ can correctly answer to two different challenges, c and c', then it knows that $x_0 = (r_0 - r_0')/(c - c') \mod q$ because $g^{r_0} = g^w + g^{cx_0} \mod q$ and $g^{r_0'} = g^w + g^{c'x_0} \mod q$. Additionally, the following also holds $b = \beta^{-c} h^{r_0} \prod_{i=1}^{m} g_i^{r_i} \equiv \beta^{-c'} h^{r_0'} \prod_{i=1}^{m} g_i^{r_i'} \mod p$, thus $h^{(r_0 - r_0')/(c-c')} \prod_{i=1}^{m} g_i^{(r_i - r_i')/(c-c')} \equiv \beta \equiv h^{x_0} v \mod p$, which means that $(x_1^* = (r_1 - r_1')/(c - c'), ..., x_m^* = (r_m - r_m')/(c - c'))$ is a representation of v with respect to the set of generators $(g_1, ..., g_m)$.

Note that if $\mathcal{P}$ knows the values $w_1, ..., w_m$ and the correct answer to single a challenge c it can directly compute suitable values $x_0 = (r_0 - w_0)/c$ and $x_i^* = (r_i - w_i)/c$.

Finally, we show how to build a simulator $\mathcal{S}$. To simulate a conversation $\mathcal{S}$ chooses c and $w_0, ..., w_m$ at random. Then it sets $a = g^{w_0} \alpha^{-c} \mod p$, $b = h^{w_0} \prod_{i=1}^{m} g_i^{w_i} \beta^{-c} \mod p$ and $r_i = w_i$ which completes the simulated conversation. Since c and $w_0, ..., w_m$ are chosen freely we obtain special honest-verifier zero-knowledge.

□

3.3.5. Disjunctive proof of ElGamal plaintext equality

A disjunctive proof of ElGamal plaintext equality can be obtained from [21, 22]. The proof described below allows to prove that a given ElGamal ciphertext $\mathcal{E}_h(m_i, x) = \langle \alpha, \beta \rangle$ encrypts a message m_i in $\{m_1, ..., m_N\} : \forall_{j \in [1,N]}, m_j \in \mathrm{G}_q$, without revealing which one it is.

$$ZKPK\left[x, m_i : \mathcal{E}_h(m_i, x) = \mathcal{E}_h(m_0, x) \ \vee ... \vee \ \mathcal{E}_h(m_i, x) = \mathcal{E}_h(m_N, x)\right]$$

The proof unfolds as follows:
Public input: $g, h, \alpha, \beta, \{m_1, ..., m_N\}$
$\mathcal{P}$ private input: $m_i \in \{m_1, ..., m_N\}$, x such that $\alpha = g^x \mod p$ and $\beta = h^x m_i \mod p$

1. $\mathcal{P}$
   (a) chooses $\forall_{j \in [1,N] \wedge j \neq i} c_j, r_j \xleftarrow{R} \mathbb{Z}_q$ and computes $a_j = g^{r_j}/\alpha^{c_j} \mod p$, $b_j = h^{r_j}/(\beta/m_j)^{c_i} \mod p$.
   (b) chooses $w \xleftarrow{R} \mathbb{Z}_q$ and computes $a_i = g^w \mod p$ and $b_i = h^w \mod p$.
   (c) sends $(a_1, b_1, ..., a_N, b_N)$ to $\mathcal{V}$.

2. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ computes $c_i = c - \sum_{j \neq i} c_j \mod q$ and $r_i = w + x c_i \mod q$ and sends $(c_1, r_1, ..., c_N, r_N)$ to $\mathcal{V}$.
4. $\mathcal{V}$ verifies $c = \sum_{j=1}^{N} c_j \ \wedge \ \forall_{j \in [1,N]} \ g^{r_j} \equiv a_j \alpha^{c_j} \mod p \ \wedge \ h^{r_j} \equiv b_j (\beta / m_j)^{c_j} \mod p$

3.3.6. Proof of inequality of two discrete logarithms

The inequality of two discrete logarithms ($\log_g \alpha \neq \log_h \beta$) can be proven using the following protocol by Camenisch and Shoup [13]:

$$ZKPK\left[x : \alpha = g^x \ \wedge \ \log_g \alpha \neq \log_h \beta\right]$$

The proof unfolds as follows: Public input: g, h, α, β
$\mathcal{P}$ private input: x such that $\alpha = g^x \mod p$

1. $\mathcal{P}$ chooses $r \xleftarrow{R} \mathbb{Z}_q$ and sends $C = (h^x/\beta)^r = h^{xr}(1/\beta)^r = h^{\omega}(1/\beta)^{\sigma} \mod p =$ to $\mathcal{V}$.
2. $\mathcal{P}$ performs a $ZKPK\left[\omega, \sigma : C = h^{\omega}(1/\beta)^{\sigma} \ \wedge \ 1 = g^{\omega}(1/\alpha)^{\sigma}\right]$
   (a) $\mathcal{P}$ chooses $s_{\omega}, s_{\sigma} \xleftarrow{R} \mathbb{Z}_q$ and sends $t_c = h^{s_{\omega}}(1/\beta)^{s_{\sigma}} \mod p$, $t_1 = g^{s_{\omega}}(1/\alpha)^{s_{\sigma}} \mod p$ to $\mathcal{V}$.
   (b) $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.
   (c) $\mathcal{P}$ sends $r_{\omega} = \omega c + s_{\omega} \mod q$ and $r_{\sigma} = \sigma c + s_{\sigma} \mod q$ to $\mathcal{V}$.
3. $\mathcal{V}$ accepts the proof if $C \neq 1 \ \wedge \ h^{r_{\omega}}(1/\beta)^{r_{\sigma}} \equiv C^c t_c \mod p \ \wedge \ g^{r_{\omega}}(1/\alpha)^{r_{\sigma}} \equiv t_1 \mod p$.

This proof is used in our constructions to prove that an ElGamal ciphertext $c = \langle \alpha, \beta \rangle$ is not an encryption of the value 1.

4. Verifiable vote encryption constructions for mix net tallying

In this section we describe our new constructions for a verifiable vote encryption that supports mix net tallying. Our constructions have three steps: i) first, we perform an initial private and verifiable shuffle of the candidates/options identifiers encryptions; then ii) we use the anonymized encryptions to create the final vote encryption; and finally, iii) we provide ZKPK for any additional "structural" vote constrains. The initial shuffle allow us to prove that a vote is well formed based on the structure of the vote encryption (i.e. the organization of the shuffled ciphertexts) instead proving the message structure inside a ciphertext.

We start by describing the construction for the simpler 1-out-of-n approval voting. Then, we show how to generalize it to k-out-of-n approval, weighted and ranked voting. To keep the description as generic as possible, we assume that the vote is encrypted by a vote machine $\mathcal{M}$, which can be a dedicated vote machine at the polling station or the voter's computer in case of Internet Voting.

At the end of each construction we give an extra output definition (V') that will be used in the cast-as-intended verification protocol described in section 6. V' will be a single encryption that represents the voter's selection(s). For every ZKPK used in this section the reader should consider the non-interactive version.

4.1. Approval voting

The simpler vote structure is selecting 1-out-of-n options (candidates). In this scenario we have the following public election setup: p, q, g are the ElGamal parameters, h is the public key of the election, and $\mathcal{I}_G = \{g_1, ..., g_n\}$ is a set of n independent generators of G_q.[4] Each generator $g_i \in \mathcal{I}_G$ is assigned to the corresponding option i. Consider also $C = C_1, ..., C_n = \mathcal{E}_h(g_1, 0), ..., \mathcal{E}_h(g_n, 0)$, a set of trivial encryptions of all the generators in $\mathcal{I}_G$.

To create a vote encryption the vote machine $\mathcal{M}$ engages in the following protocol with voter $\mathcal{V}$.

1. $\mathcal{V}$ selects one option by sending the corresponding index $1 \leq s \leq n$ to $\mathcal{M}$.
2. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_n$ of C and the corresponding proof of correctness P_π.[5]
   The element of C' that corresponds to the element $C_i \in C$ is denoted by $C'_{\pi(i)} = C_i \otimes \mathcal{E}_h(1, \rho_i) = \mathcal{E}_h(g_i, \rho_i)$, where π is the permutation used in the shuffle and ρ_i is the corresponding randomizer.
3. $\mathcal{M}$ sets $V = C'_{\pi(s)}$, i.e. the vote encryption V is set to the encryption $C'_{\pi(s)} \in C'$ that encrypts g_s.[6]
4. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P_\pi, V\}$

Assuming a private and sound shuffle, it is easy to verify the well formedness of the vote encryption V because it is one of the shuffled ciphertexts, thus it must encrypt a valid option identifier. Given that the shuffle is private no information about the chosen option/candidate is revealed. The mix net based tally takes as input V.

For the cast-as-intended verification protocol, refer to section 6, in the case of 1-out-of-n votes we define $V' = V = C'_{\pi(s)} = \mathcal{E}_h(g_s, \rho_s)$ as the single encryption that represents the voter's selection.

4.1.1. Approving k-out-of-n options

The k-out-of-n approval vote protocol can be easily extended from the 1-out-of-n protocol as follows:

1. $\mathcal{V}$ selects k options by sending the corresponding set of indexes $S = s_1, ..., s_k$ to $\mathcal{M}$.
2. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_n$ of C and the corresponding proof of correctness P_π.
3. $\mathcal{M}$ sets $V = V_1, ..., V_k = C'_{\pi(s_1)}, ..., C'_{\pi(s_k)}$, i.e. the vote V is defined as the set of encryptions which encrypt the corresponding g_{s_l}'s.
4. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P_\pi, V\}$

Like in the 1-out-of-n protocol, the private verifiable shuffle ensures that no information about the values encrypted in V are leaked. Additionally, it is easy to verify that exactly k options were selected.

[4] The generators in $\mathcal{I}_G$ are also independent of the ElGamal parameter g.
[5] Note that steps 1 and 2 can be swapped.
[6] V can also be identified by the index $\pi(s)$.

For the case of k-out-of-n votes we define $V' = V_1 \otimes V_2 \otimes ... \otimes V_k = \mathcal{E}_h(\prod_{s_l \in S} g_{s_l}, \sum_{s_l \in S} \rho_{s_l})$ as the single encryption that represents the voter's selection.

The mix net based tally takes as input V. If the decryption of V' is "easy" ("low" number of possible plaintexts in V') and there is no privacy issue in decrypting the vote "aggregation", V' can be used directly in the mix net tally.

4.1.2. Approving $[k_{min}, k_{max}]$-out-of-n options

When the voter can choose between k_{min} to k_{max} options and there is no privacy issues in knowing how many options the voter has selected we can use the k-out-of-n protocol because everyone can easily verify that $k_{min} \leq k \leq k_{max}$. However, if revealing the exact number of selected options is a problem we need a different protocol, one that hides the value of k. To address this issue we propose the following vote construction:

1. $\mathcal{V}$ selects k options by sending the corresponding set of indexes $S = s_1, ..., s_k$ to $\mathcal{M}$.
2. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_n, C'_{\delta_1}, ..., C'_{\delta_{k_{max}-k_{min}}}$ of $C \parallel C_\delta$ and the corresponding proof of correctness P_π. The δ_i indexes are defined as $\delta_i = n+i$. Let $C_\delta = C_{\delta_1}, ..., C_{\delta_{k_{max}-k_{min}}} : \forall_{i \in [1, k_{max}-k_{min}]}\ C_{\delta_i} = \mathcal{E}_h(1, 0)$. Thus, C' contains $k_{max} - k_{min}$ encryptions of the value 1.
3. $\mathcal{M}$ sets $V = V_1, ..., V_{k_{max}} = C'_{\pi(s_1)}, ..., C'_{\pi(s_k)} \bigcup_{m=1}^{k_{max}-k} C'_{\pi(\delta_m)}$, i.e. the vote V is defined as a set of k_{max} encryptions containing the k selections of the voter and $k_{max} - k$ encryptions of the value 1.
4. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P_\pi, V\}$

The above protocol hides the number of selections because V is of constant size, namely k_{max} encryptions, thus there is nothing that reveals the number of selections made by the voter, i.e. the size of S.

As in the previous protocols, the private verifiable shuffle ensures that no information about the values encrypted in V are leaked. Additionally, it is easy to verify that $k_{min} \leq k \leq k_{max}$ options were selected. In this protocol V' is defined as $V' = V_1 \otimes V_2 \otimes ... \otimes V_{k_{max}} = \mathcal{E}_h(\prod_{s_l \in S} g_{s_l}, \sum_{s_l \in S} \rho_{s_l} + \sum_{m=1}^{k_{max}-k} \rho_{\delta_m})$.

The mix net based tally takes as input V or V', as in the k-out-of-n protocol.

4.2. Ranked voting

There are many scenarios in which a voter is requested to order a set of options. Next we describe our construction for a verifiable $[k_{min}, k_{max}]$-out-of-n ranked vote and then we explain how to transform it for the simpler k-out-of-n ranked vote.

1. $\mathcal{V}$ selects $k_{min} \leq k \leq k_{max}$ options by sending the corresponding ordered set of indexes $S = s_1, ..., s_k$ to $\mathcal{M}$.
2. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_n, C'_{\delta_1}, ..., C'_{\delta_{k_{max}-k_{min}}}$ of $C \parallel C_\delta$ and the corresponding proof of correctness P_π. The δ_i indexes are defined as $\delta_i = n+i$. Let $C_\delta = C_{\delta_1}, ..., C_{\delta_{k_{max}-k_{min}}} : \forall_{i \in [1, k_{max}-k_{min}]}\ C_{\delta_i} = \mathcal{E}_h(1, 0)$. Thus, C' contains $k_{max} - k_{min}$ encryptions of the value 1.
3. $\mathcal{M}$ sets $V = V_1, ..., V_{k_{max}} = C'_{\pi(s_1)}, ..., C'_{\pi(s_k)} \bigcup_{m=1}^{k_{max}-k} C'_{\pi(\delta_m)}$), i.e. the vote V is defined as the ordered set of k_{max} encryptions containing the k selections of the voter and $k_{max} - k$ encryptions of the value 1. Note that the order of the indexes in S is preserved in V.

4. $\mathcal{M}$ sets $P_{k_{min}} = \bigcup_{i=1}^{k_{min}} ZKPK\left[x_i : \alpha_i = g^{x_i} \ \wedge \ \log_g \alpha_i \neq \log_h \beta_i\right] : V_i = \langle \alpha_i, \beta_i \rangle$. $P_{k_{min}}$ is the set of ZKPKs which prove that no V_i, $1 \leq i \leq k_{min}$, is an encryption of the value 1, i.e. it proves that all ranks form 1 to k_{min} where assigned to some options.
5. $\mathcal{M}$ sets
$P_{continuous} = \bigcup_{i=k_{min}+1}^{k_{max}-1} ZKPK[x_i, x^* : (\alpha_i = g^{x_i} \ \wedge \ \log_g \alpha_i \neq \log_h \beta_i) \ \vee \ (\alpha_i^* = g^{x^*} \ \wedge \ \beta_i^* = h^{x^*})] : V_i = \langle \alpha_i, \beta_i \rangle \wedge V_i^* = V_{i+1} \otimes ... \otimes V_{k_{max}} = \langle \alpha_i^*, \beta_i^* \rangle$.
The $P_{continuous}$ is a set of ZKPK proofs that can be easily constructed from the proofs of inequality and equality of discrete logarithms using the techniques from [21]. The proofs prove for each $i \in [k_{min}+1, k_{max}-1]$ that V_i does not encrypt the value 1 or that all $V_j : j > i$ encrypt the value 1, i.e. it guarantees that a rank j cannot be given to an option if there is a rank $i : i < j$ that was not assigned.
6. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P = P_\pi \cup P_{k_{min}} \cup P_{continuous}, V\}$

Once again the voter selections are hidden by the private shuffle and the ZKPKs, which also guarantee the construction of a valid vote. If it's not desirable or possible to break the vote in the tally process, e.g. when tallying using the STV method, then a mix net that can shuffle sets of ordered ciphertexts must be used, otherwise k_{max} mix nets can be used (one for each rank) or a single one with inputs $(V_i)^i$ instead of V_i.

For this type of vote we define $V' = (V_1)^1 \otimes (V_2)^2 \otimes ... \otimes (V_{k_{max}})^{k_{max}} = \mathcal{E}_h(\prod_{s_l \in S} g_{s_l}^l, \sum_{s_l \in S} l\rho_{s_l} + \sum_{m=1}^{k_{max}-k}(m+k)\rho_{\delta_m})$.

In the case of k-out-of-n ranked votes the protocol is simplified as follows:

(a) C_δ is removed from the shuffle in step 2.
(b) There are exactly k encryptions in V, accordingly to the voter's order of preference.
(c) The $P_{k_{min}}$ (step 4) and $P_{continuous}$ (step 5) proofs are removed from the protocol.

4.3. Weighted voting

There are scenarios, e.g. the elections in Luxembourg and in some parts of Germany [23, 24], which require the ability to weight votes, i.e. the voter has the right to cast k votes and can give up to t votes to each candidate. The ranked voting protocol can be easily extended to support weight voting by including a variable weight factor instead of fixed rank ("fixed weight"). However, that solution would leak a vote pattern just by looking at the vote ciphertexts in V, which could compromise the voter's privacy.

To prevent the vote pattern leakage from the ciphertexts in V we propose a different approach based on the previous k-out-of-n and $[k_{min}, k_{max}]$-out-of-n protocols. Next we describe the protocol for the case in which the voter casts a bounded number of votes $k_{min} \leq k \leq k_{max}$ and after the simplification that can be performed for the case in which the voter must use all k votes.

1. $\mathcal{V}$ selects k options by sending the corresponding set of indexes $S = s_1^{i_1,z_1}, ..., s_k^{i_k,z_k}$ to $\mathcal{M}$. An index $s_l^{i_l,z_l}$ represents the choice of candidate i_l and $0 \leq z_l < t$ allows to choose up to t different indexes for the same candidate.
2. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_{tn}, C'_{\delta_1}, ..., C'_{\delta_{k_{max}-k_{min}}})$ of $C^t \parallel C_\delta$ and the corresponding proof of correctness P_π. The δ_i indexes are defined as $\delta_i = tn+i$.

C^t represents t times the encryptions in C, thus $\mathcal{V}$ can choose the same option up to t times. For a given candidate i, the corresponding generator g_i is encrypted in the indexes $i, i+n, ..., i+n(t-1)$, thus we define $C'_{\pi(s_l^{i_l,z_l})} = \mathcal{E}_h(g_{i_l}, \rho_{i_l+nz_l})$.
C_δ is defined as $C_\delta = C_{\delta_1}, ..., C_{\delta_{k_{max}-k_{min}}} : \forall_{i\in[1,k_{max}-k_{min}]}\ C_{\delta_i} = \mathcal{E}_h(1,0)$. Thus, C' contains $k_{max} - k_{min}$ encryptions of the value 1 and t times the encryptions of each candidate identifier.

3. $\mathcal{M}$ sets $V = V_1, ..., V_{k_{max}} = C'_{\pi(s_1^{i_1,z_1})}, ..., C'_{\pi(s_k^{i_k,z_k})} \bigcup_{m=1}^{k_{max}-k} C'_{\pi(\delta_m)}$, i.e. the vote V is defined as a set of k_{max} encryptions containing the k selections of the voter and $k_{max} - k$ encryptions of the value 1.
4. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P_\pi, V\}$

The above protocol hides the number of selections because V is of constant size, namely k_{max} encryptions, thus there is nothing that reveals the number of selections made by the voter, i.e. the size of S. Additionally, the shuffle input limits to t the number of selections that a voter can make in any candidate.

As in the previous protocols, the private verifiable shuffle ensures that no information about the values encrypted in V are leaked. Additionally, it is easy to verify that $k_{min} \leq k \leq k_{max}$ options were selected.

In this protocol V' is defined as $V' = V_1 \otimes V_2 \otimes ... \otimes V_{k_{max}} = \mathcal{E}_h(\prod_{s_l^{i_l,z_l}\in S} g_{i_l}, \sum_{s_l^{i_l,z_l}\in S} \rho_{i_l+nz_l} + \sum_{m=1}^{k_{max}-k} \rho_{\delta_m})$.

The protocol for the case in which k is a constant is very similar to the above described protocol. The only differences are: i) C_δ is removed form C'_π in step 2 and, consequently, ii) there are exactly k encryptions in V, all of them encrypting a candidate identifier, step 3.

The mix net based tally is similar to the previously described protocols.

5. Verifiable vote constructions for homomorphic vote tallying

Enabling homomorphic vote tallying has the advantages of hiding the individual vote structure, by revealing only the election totals, and enables a faster vote count process. However, enabling homomorphic vote tallying has a cost, the cost of creating one homomorphic counter for each option in the vote encryption and assuring that the votes are summed in the correct homomorphic counters.

We start by describing the construction for approval voting and then we describe the constructions for weighted and ranked voting. Like in the previous section, the reader should consider the non-interactive version of the ZKPKs.

5.1. Approving $[k_{min}, k_{max}]$-out-of-n options

We start by describing the protocol for the simpler case, i.e. $k_{min} = 0$. Then we describe the additions necessary to support $k_{min} = 1$, $k_{min} = k = k_{max}$ and the general case $k_{min} \leq k \leq k_{max}$. Recall that the election setup defines a set of independent generators $\mathcal{I}_G = \{g_1, ..., g_n\}$, one for each option i, and $C = C_1, ..., C_n = \mathcal{E}_h(g_1, 0), ..., \mathcal{E}_h(g_n, 0)$, a set of trivial encryptions of all the generators in $\mathcal{I}_G$. The protocol for approving $[0, k_{max}]$-out-of-n options unfolds as follows:

1. $\mathcal{V}$ selects $0 \leq k \leq k_{max}$ options by sending the corresponding set of indexes $S = s_1, ..., s_k$ to $\mathcal{M}$.
2. Let C_H be a list of k_{max} trivial encryptions of the value 1, i.e. $\mathcal{E}_h(1, 0)$. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_n, C'_{H_1}, ..., C'_{H_{k_{max}}}$ of $C \parallel C_H$ and the corresponding proof of correctness P_π, where the index H_i is defined as $H_i = n + i$.
3. $\mathcal{M}$ sets $V = V_1, ..., V_{k_{max}} = C'_{\pi(s_1)}, ..., C'_{\pi(s_k)} \bigcup_{m=1}^{k_{max}-k} C'_{\pi(H_m)}$, i.e. the vote V is defined as the set of k_{max} encryptions containing the k selections of the voter and $k_{max} - k$ encryptions of the value 1.
4. $\mathcal{M}$ sets one "homomorphic" vote V_{H_i} to each option i. $\forall_{i \notin S} : V_{H_i} = C'_{\pi(i)} = \mathcal{E}_h(g_i, \rho_i)$ and $\forall_{i=s_l \in S} : V_{H_i} = C'_{\pi(H_{k_{max}-k+l})} = \mathcal{E}_h(1, \rho_{n+k_{max}-k+l})$. We denote the list of assignments by V_H.
   In other words, V_{Hi} for the not selected candidates is the encryption of the corresponding generator g_i and for the selected candidates is the encryption of the value 1.
5. $\mathcal{M}$ computes a set of n non interactive zero-knowledge proofs $P_H = P_{H_1}, ..., P_{H_n}$ such that:
   $$P_{H_i} = ZKPK\left[z_i, x_i : V_{H_i} = \mathcal{E}_h(g_i^{x_i}, z_i)\right]$$
   The proof P_{H_i} is a proof of knowledge of an ElGamal plaintext representation to the generator g_i. Under the independent generators assumption, and the set of available encryptions (C'), the proof P_{H_i} can only be accepted if V_{H_i} encrypts g_i or the value $1 = g_i^0$. Here the encryption of 1 represents the selection of the option.
6. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P = P_\pi \cup P_H, V, V_H\}$

At the end of the election, the homomorphic tallying can be performed using the $\otimes$ operator to accumulate the individual homomorphic votes into n homomorphic counters. The decryption of the homomorphic counter of option i will reveal the value $g_i^{N-v_i}$, where N is the total number of votes accumulated and v_i is the number of votes received by option i.

The private verifiable shuffle prevents the leakage of the voter's preferences and guarantees that: i) no more than k votes can be cast, and ii) no candidate gets more than one vote.

For this protocol we define $V' = V_1 \otimes V_2 \otimes ... \otimes V_{k_{max}} = \mathcal{E}_h(\prod_{s_l \in S} g_{s_l}, \sum_{s_l \in S} \rho_{s_l} + \sum_{m=1}^{k_{max}-k} \rho_{H_m}))$.

Case $k_{min} = 1$. . When $k_{min} = 1$ it is necessary to prove that at least one option was selected, i.e. proof that one $V_i = \langle \alpha, \beta \rangle$ encrypts a value different than 1, which can be performed with the proof of inequality of two discrete logarithms:

$$P_{\neq 1} = ZKPK\left[x : \alpha = g^x \ \wedge \ \log_g \alpha \neq \log_h \beta\right]$$

The proof $P_{\neq 1}$ must be added to P in step 6 and can be performed on any V_i, e.g. V_1.

Case $k_{min} = k = k_{max}$.. When $k_{min} = k_{max}$ it is necessary to prove that all elements of V encrypt a generator, i.e. every element of V selects one option. To prove the above in an efficient way we redefine C_H to be a list of k trivial encryptions of the value g_v,

i.e. $\mathcal{E}(g_v, 0)$. g_v is an additional independent generator, thus $g_v \notin \mathcal{I}_G$. The remainder of the protocol is unaltered. With this change it is possible to prove that all k votes, the k encryptions of g_v, were assigned to an option by verifying that no element of V encrypts the value g_v. The efficient way to do this is to publish a proof of knowledge of the ElGamal plaintext representation of V' in order to the set of generators $\mathcal{I}_G$.

$$P_K = ZKPK\left[x_0, x_1, ..., x_n : V' = \mathcal{E}_h\left(\prod_{i=1}^{n} g_i^{x_i}, x_0\right)\right]$$

Given that $g_v \notin \mathcal{I}_G$, P_K proves that all k votes were assigned to a valid option. The proof P_K must be added to P in step 6. Note that now the homomorphic counters decryption outputs $g_i^{N-v_i} g_v^{v_i}$.

The general case $k_{min} \leq k \leq k_{max}$.. If we are not in any of the special cases described above, the proof of vote well-formed is more complex. Here we require the use of g_v as in the previous case. In the general case, V can contain up to $k_{max} - k_{min}$ encryptions of g_v. To prove that the vote is well-formed, $\mathcal{M}$ publishes the additional data:

$$V_{g_v} = \mathcal{E}_h(g_v^{k_{max}-k}, r)$$

$$P_{g_v} = ZKPK\left[r, k : V_{g_v} = \mathcal{E}_h\left(g_v^0, r\right) \ \vee ... \vee \ V_{g_v} = \mathcal{E}_h\left(g_v^{k_{max}-k_{min}}, r\right)\right]$$

$$V' = V_1 \otimes V_2 \otimes ... \otimes V_{k_{max}} \otimes V_{g_v}^{-1} = \mathcal{E}_h\left(\prod_{s_l \in S} g_{s_l}, -r + \sum_{s_l \in S} \rho_{s_l} + \sum_{m=1}^{k_{max}-k} \rho_{H_m}\right)$$

$$P_K = ZKPK\left[x_0, x_1, ..., x_n : V' = \mathcal{E}_h\left(\prod_{i=1}^{n} g_i^{x_i}, x_0\right)\right]$$

The proofs P_{g_v} and P_K prove that there is a valid number of encryption of g_v in V and must be added to P in step 6.

5.2. Weighted voting

In this section we describe the protocol to distribute $k \in [0, k_{max}]$ votes by n options, where it is possible to weight each option up to t. For the other different cases of bounds on k we can apply directly the techniques described in section 5.1. Let t_i represent the weight given to option i.

1. $\mathcal{V}$ selects k options by sending the corresponding set of indexes $S = s_1^{i_1,z_1}, ..., s_k^{i_k,z_k}$ to $\mathcal{M}$. An index $s_l^{i_l,z_l}$ represents the choice of candidate i_l and $0 \leq z_l < t$ allows to choose up to t different indexes for the same candidate.
2. Let $C_H = C_{H_1}, ..., C_{H_{max}}$ be a list of k_{max} trivial encryptions of the value 1, i.e. $\mathcal{E}_h(1, 0)$. $\mathcal{M}$ creates a verifiable shuffle $C' = C'_1, ..., C'_{tn}, C'_{H_1}, ..., C'_{H_{max}}$ of $C^t \parallel C_H$ and the corresponding proof of correctness P_π. The H_i indexes are defined as $H_i = tn + i$.
   C^t represents t times the encryptions in C, thus $\mathcal{V}$ can choose the same option up to t times. For a given candidate i, the corresponding generator g_i is encrypted in the indexes $i, i+n, ..., i+n(t-1)$, thus we define $C'_{\pi(s_l^{i_l,z_l})} = \mathcal{E}_h(g_{i_l}, \rho_{i_l+nz_l})$.

3. $\mathcal{M}$ sets $V = V_1, ..., V_{k_{max}} = C'_{\pi(s_1^{i_1,z_1})}, ..., C'_{\pi(s_k^{i_k,z_k})} \bigcup_{m=1}^{k_{max}-k} C'_{\pi(H_m)}$, i.e. the vote V is defined as a set of k_{max} encryptions containing the k selections of the voter and $k_{max} - k$ encryptions of the value 1.
4. $\mathcal{M}$ creates a vote table T of size n by t. The T_i row in T is the row of option i. Then, $\mathcal{M}$ assigns to each position T_{ij} an element of $C' \backslash V$ such that row T_i contains t_i encryptions of the value 1 and $t - t_i$ encryptions of g_i. The "homomorphic" vote definition is now $V_{H_i} = T_{i1} \otimes ... \otimes T_{it}$, i.e. V_{H_i} encrypts the value $g_i^{t-t_i}$. V_H is now defined as $V_H = T$.
5. $\mathcal{M}$ computes a set of n non interactive zero-knowledge proofs $P_H = P_{H_1}, ..., P_{H_n}$ such that:
$$P_{H_i} = ZKPK\left[z_i, x_i : V_{H_i} = \mathcal{E}_h(g_i^{x_i}, z_i)\right]$$
The proof P_{Hi} is a proof of knowledge of an ElGamal plaintext representation to the generator g_i. Under the independent generators assumption, and the set of available encryptions (C'), the proof P_{H_i} can only be accepted if V_{H_i} encrypts a valid power of g_i or the value $1 = g_i^0$.
6. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C', P = P_\pi \cup P_H, V, V_H\}$

The homomorphic tally is performed as in the approval vote case, see section 5.1. Once again it is easy to check, without revealing the voter's choices, that no candidate can have more than t votes and that the overall sum of votes cannot exceed k_{max}. For this protocol, V' is defined as $V' = V_1 \otimes V_2 \otimes ... \otimes V_{k_{max}} = \mathcal{E}_h(\prod_{s_l^{i_l,z_l} \in S} g_{i_l}, \sum_{s_l^{i_l,z_l} \in S} \rho_{i_l + n z_l} + \sum_{m=1}^{k_{max}-k} \rho_{H_m})$.

5.3. Ranked voting

Unfortunately, due to the ordering of the candidates the internal structure of a ranked vote is more complex than the structure of the approval or weighted votes. To deal with the additional complexity we have to use a more elaborated proof of vote well-formness. Note that the feasibility of an homomorphic tally of ranked votes highly depends on the tally method. Our vote construction is specially suited for Borda like tally methods but it is not enough to deal with more elaborated tally methods such as IRV/STV or Condorced.

Below we give the details of our protocol for ranking k-out-of-n candidates. For other different cases of bounds on k we can apply the techniques described in sections 4.2 and 5.1.

1. $\mathcal{V}$ selects k options by sending the corresponding ordered set of indexes $S = s_1, ..., s_k$ to $\mathcal{M}$.
2. $\mathcal{M}$ creates k verifiable shuffles $C'^* = C'^1, ..., C'^k$, where $C'^i = C'^i_1, ..., C'^i_{n+1}$ is the shuffle of $C \parallel \mathcal{E}_h(g_v, 0)$, where g_v is an additional independent generator such that $g_v \notin \mathcal{I}_G$. $P_\pi = P_{\pi_1}, ..., P_{\pi_k}$ represents the corresponding proofs of correctness. In the protocol, only the encryptions of the shuffle C'^i can be used to give the i^{th} rank.
3. $\mathcal{M}$ sets $V = V_1, ..., V_k = C'^1_{\pi(s_1)}, ..., C'^k_{\pi(s_k)}$. V is defined as the ordered set of k encryptions containing the k selections of the voter such that $C'^i_{\pi(s_i)} = \mathcal{E}_h(g_{s_i}, \rho'^i_{s_i})$.

Additionally, $\mathcal{M}$ creates:

$$P_K = ZKPK\left[x_0, x_1, ..., x_n : V' = \mathcal{E}_h\left(\prod_{i=1}^{n} g_i^{x_i}, x_0\right)\right]$$

4. $\mathcal{M}$ creates the shuffle C'^{k+1} of $V \bigcup_{i=1}^{n-k} \mathcal{E}_h(g_v, 0)$, i.e. the shuffle of the union of the k encryptions in V with $n-k$ encryptions of the value g_v. It also creates the corresponding proof of correctness $P_{\pi_{k+1}}$.
   Then $\mathcal{M}$ creates a vote table T of size n by $k+1$. The T_i row in T is the row of option i. Then, $\mathcal{M}$ assigns to each position $T_{i,j}$ an element of $C'^j \backslash V_j$,[7] such that each row T_i contains one encryption of the value g_v and k encryptions of g_i. The encryption of the value g_v on the column $j \neq k+1$ represents the ranking of candidate i with rank j. If the value g_v appears on cell $T_{i,k+1}$ it means that no rank was given to candidate i.
   Let $V_{H_i} = T_{i,1} \otimes ... \otimes T_{i,k+1}$ and $V_H = T$.
5. Let $V_{Hi} = \langle \alpha_i, \beta_i \rangle$. $\mathcal{M}$ computes a set of n non interactive zero-knowledge proofs $P_H = P_{H_1}, ..., P_{H_n}$ such that:

$$P_{H_i} = ZKPK\left[x_i : \alpha_i = g^{x_i} \ \wedge \ \beta_i/(g_i^k g_v) = h^{x_i}\right]$$

6. $\mathcal{M}$ outputs the complete vote encryption $V^* = \{C'^*, C'^{k+1}, P = P_\pi \cup P_{\pi k+1} \cup P_K \cup P_H, V, V_H\}$

In this protocol the proof that the vote is well formed is constructed from the proofs P_π, $P_{\pi k+1}$, P_K and P_H. Given that $g_v \notin \mathcal{I}_G$, P_K proves that all k votes were assigned to a valid option. P_π guarantees that a rank j cannot be assigned to more than one candidate (there is only one encryption of the value g_v). The proofs $P_{\pi k+1}$ and P_H prove that no more than one rank can be assigned to any candidate.

The homomorphic tally for Borda like tally methods is similar to the approval vote case, see section 5.1, but now the homomorphic counters are defined on each $T_{i,j}$ instead of each V_{H_i}. For the homomorphic ranked vote construction V' is defined as:

$$V' = (V_1)^1 \otimes (V_2)^2 \otimes ... \otimes (V_k)^k = \mathcal{E}_h(\prod_{s_l \in S} g_{s_l}^l, \sum_{s_l \in S} l\rho'^l_{s_l})$$

6. Cast-as-intended verification of complex ballots

In this section we show how to extract a voter verifiable receipt from a complex ballot encryption. The receipt verification requires the computation of some modular operations computed on public data and on some private voter's input. With this technique it is also possible to have a single verification code for an entire multi race ballot. In section 6.2 we explain how a "more traditional" receipt with one verification code for each option can be obtained from our voter verifiable vote encryption constructions. This later receipt can be quite long in the more complex scenarios, but it can be verified by the voter with simple string matches.

[7] Note that $C'^{k+1} \backslash V_{k+1} = C'^{k+1}$ because there is no element V_{k+1}.

6.1. A single verification code for complex ballots

The protocol between $\mathcal{M}$ and the voter $\mathcal{V}$ to create a cast-as-intended voter verifiable vote verification code runs after the creation of the vote encryption V^*, thus the voter knows V', the set of selected candidates $S^* = s_1^*, ..., s_k^*$ and the ranks or weights given to each candidate $t_1^*, ..., t_n^*$. $\forall_{k \notin S^*}\ t_k^* = 0$ and in the case of approval voting each selected candidate i has $t_i^* = 1$).

Accordingly to the protocols in sections 4 and 5, it is expected that $V' = \mathcal{E}_h(\prod_{i=1}^{k} g_{s_i^*}^{t_i^*}, r_v)$ for some randomness r_v known to $\mathcal{M}$. $\mathcal{M}$ now proves to $\mathcal{V}$ that it has encrypted the voter's selections by running a slightly modified version of the protocol described in section 3.3.4, in which it reveals some additional information to $\mathcal{V}$. The proof unfolds as follows:

Public input: $g, h, V' = \langle \alpha, \beta \rangle$
$\mathcal{M}$ private input: $r_v = x_0, (x_1, ..., x_n)$ such that $\alpha = g^{x_0} \mod p$ and $\beta = h^{x_0} \prod_{i=1}^{n} g_i^{x_i} \mod p$.

1. $\mathcal{M}$ chooses $w_0, ..., w_n \xleftarrow{R} \mathbb{Z}_q$, computes $\sigma = \prod_{i=1}^{n} g_i^{w_i} \mod p$ and sends $a = g^{w_0} \mod p$, $b = h^{w_0}\sigma \mod p$ to $\mathcal{V}$.
2. $\mathcal{M}$ commits to the vote by sending the verification code σ to $\mathcal{V}$.
3. $\mathcal{V}$ chooses $c \xleftarrow{R} \mathbb{Z}_q$ and sends it to $\mathcal{M}$.
4. $\mathcal{M}$ sends $r_0, ..., r_n : r_i = (w_i + cx_i) \mod q$ to $\mathcal{V}$.
5. $\mathcal{M}$ sends the vote V^* and the vote receipt $a, b, c, r_0, ..., r_n$ to a $\mathcal{PBB}$ (public bulletin board).
6. $\mathcal{PBB}$ publishes $V^*, a, b, c, r_0, ..., r_n$ iif

$$g^{r_0} \equiv a\alpha^c \mod p \ \wedge\ h^{r_0} \prod_{i=1}^{m} g_i^{r_i} \equiv b\beta^c \mod p$$

   Note that $V' = \langle \alpha, \beta \rangle$ is easily computable from V^*.
7. $\mathcal{V}$ verifies that her vote/receipt information was correctly published in the PBB and that $\sigma = \prod_{i=1}^{n} g_i^{r_i - t_i^* c} \mod p$.

Given the construction of $V*$, which is based on the verifiable shuffle of known plaintext encryptions we already have a proof that $\mathcal{M}$ knows the contents of V', thus the above protocol is just to enable the construction of a voter verifiable receipt, which should not reveal the voter's selections. To prove that the vote receipt does not reveal the voter's vote we prove that the above protocol is a special honest-verifier zero-knowledge proof of a representation of the plaintext in V', thus it does not reveal the voter's vote. Then we will argue about the soundness of the voter's receipt verification.

Theorem 2. *The above protocol is special honest-verifier zero-knowledge for proving knowledge of a representation $(x_1^*, ..., x_n^*)$ of a value $v \in G_q$, with respect to a set of generators $(g_1, ..., g_n)$, such that $V' = \mathcal{E}_h(v, x_0)$.*

Proof. The protocol is complete because if both $\mathcal{M}$ and $\mathcal{V}$ are honest the equations is steps 6 and 7 will always hold. The correctness (proof of knowledge) of this protocol follows immediately from the underlying protocol described in section 3.3.4, which output

is a subset of the output of the present protocol. Thus, the same process can be used to obtain a representation of v with respect to the set of generators $(g_1, ..., g_n)$.

Finally, we show how to build a simulator $\mathcal{S}$. To simulate a conversation $\mathcal{S}$ chooses c and $w_0, ..., w_n$ at random. Then, it sets $a = g^{w_0}\alpha^{-c} \mod p$, $b = h^{w_0}\prod_{i=1}^{n} g_i^{w_i}\beta^{-c} \mod p$ and $r_i = w_i$. Finally, the simulator creates $\sigma = \prod_{i=1}^{n} g_i^{r_i - t'_i c} \mod p$ for the simulated candidate rank/weight set (i.e. vote) $t'_1, ..., t'_n$. Since c and $w_0, ..., w_m$ are chosen freely we obtain special honest-verifier zero-knowledge.

□

The soundness of the voter verification relies on the assumption that the data published in the $\mathcal{PBB}$ is correct. Given that anyone can verify the correctness of the published data, we assume that the voter is verifying the correct $\mathcal{PBB}$ entry. We also assume that under the independent generators assumption, Assumption 1, $\mathcal{M}$ only knows one representation with respect to the set of generators $\mathcal{I}_G = \{g_1, ..., g_n\}$ of σ and of the plaintexts encrypted in W, V' and of the value $\prod_{i=1}^{m} g_i^{r_i} \mod p$.

Under our assumptions, the above protocol guarantees that: i) removing one of the voter's selections implies that there will be one $r_j = w_j : j \in S^*$; ii) adding one more selection i with rank/weight t'_i implies that there will be one $r_i = w_i + t'_i c : i \notin S^*$; and iii) replacing one of the voter's selections implies that there will be one $r_j = w_j : j \in S^*$ and one $r_i = w_i + t'_i c : i \notin S^*$. Consequently, i) removing one selection implies that, in the computation of σ, $\mathcal{M}$ uses $w'_j = w_j + t^*_j c$; ii) adding on selection implies the use of $w'_i = w_i - t'_i c$; and ii) replacing one selection would imply both. Thus, assuming that $\mathcal{V}$ selects a random challenge c, $\mathcal{M}$ has only a negligible probability of successfully changing the voter's vote without being detected, i.e. the probability of $\mathcal{M}$ guessing the random challenge c issued by $\mathcal{V}$.

6.1.1. Support for multi-race ballots

The extension of the single verification code to a multi-race / multi-questions ballot is very simple and requires only a few tweaks. The protocol follows the steps of the previously described protocol with the following tweaks:

1. The multi-race ballot setup has a global set of independent generators $\mathcal{I}^*_G = \mathcal{I}^1_G \cup ... \cup \mathcal{I}^y_G$ for all the y races in the ballot. The set of generators $\mathcal{I}^i_G$ is the set of generators needed for race i.
2. The proof is made for a "global" vote $V'^* = V'_1 \otimes ... \otimes V'_y$, where V'_i is the V' encryption computed for race i.

6.1.2. Usability considerations

In this paper we do not attempt to give a full voting protocol but we have identified some usability issues that should be addressed in future work that uses the described techniques.

The use of a single verification code to verify that a vote is cast-as-intended implies that the voter is somehow able to perform modular arithmetic. Of course the voter cannot do it by itself and will need the help of a modular arithmetic calculator. The good thing is that the needed computation is all on public data, i.e. the data published in the $\mathcal{PBB}$ and on one set of valid vote selections.

Note that only the voter knows which really was her set of selections S^*, the ranks/weights given to each candidate $t_1^*, ..., t_n^*$ and the σ value given by $\mathcal{M}$. Thus, using a calculator (e.g. a dedicated hardware or an application running on a mobile phone/tablet or computer) does not provide any proof about the real contents of the vote to the calculator. In fact, anyone can try any possible vote combination on any vote published in the $\mathcal{PBB}$, but only the voter can verify that the vote was cast-as-intended because only she knows her set of selections S^*, the ranks/weights given to each candidate $t_1^*, ..., t_n^*$ and the σ value given by $\mathcal{M}$.

It is also important to consider the fact that the voter must input her selections S into the calculator, thus for complex ballots it is important to evaluate if the voter can effectively replicate her choices in the both $\mathcal{M}$ and in the calculator.

In the cases where the number of vote possibilities is small, the receipt can contain all possible σ_i values, one for each of the i vote possibilities. In this scenario the voter just has to perform a simple string match, i.e. there is no need for an auxiliary calculator because anyone can verify that the data published in the $\mathcal{PBB}$, and therefore the σ_i values on the receipt, is correct.

Another important aspect of usability are the sizes of the challenge c and of σ, which as described are respectively of size $\log_2 q$ and $\log_2 p$ bits. Reducing the size of the challenge reduces the voter's verification soundness because as explained in section 6.1, $\mathcal{M}$ can create a fake proof if it is able to guess the challenge value c. The honest-verifier zero-knowledge property of the protocol remains unchanged because the values $w_0, ..., w_n$ are still chosen randomly from $\mathbb{Z}_q$, which makes the output of the proofs random in the corresponding domains. Assuming now that the distribution of the elements of G_q is uniform in $\mathbb{Z}_p^*$, $\mathcal{V}$ could verify only m bits of the σ value while keeping a verification soundness linear to $1 - 2^m$.

6.2. Creating a receipt with one verification code per option/candidate

There are some scenarios, e.g. poll station voting, in which for anti-coercion reasons the voter should not end the voting process knowing only one σ value or in which, for dispute resolution, the receipt must be immediately verified by the voter without the help of electronic equipment. In these scenarios, a more "traditional" vote receipt with one verification code for each option/candidate is more appropriate for a human verification. The protocol described in section 6.1 allows to create such receipt easily because the receipt already contains one verification code for each option/candidate i, the r_i value.

For the cases of k-out-of-n approval votes the verification procedure is trivial, i.e. the voter must check that k individual verification codes match her selections. In this scenario, $\mathcal{M}$ must reveal the k individual verification codes $w_{s_l^*} : \forall s_l^* \in S^*$, instead of σ in step 2 of the protocol. $\mathcal{M}$ adds to the receipt the values $r'_i = r_i - c$. Then, the voter can check that every individual verification code revealed to her match the corresponding r'_i value on the receipt by doing simple k string matches.

For the $[k_{min}, k_{max}]$-out-of-n approval votes it is necessary to adapt the vote construction to avoid having to verify n codes. The idea here is to add $k_{max} - k_{min}$ dummy options/candidates. In this way the not used votes will go to the dummy candidates and, therefore, the voter can easily verify that her not used votes did not went to some valid candidate.

The verification of ranked or weighted votes is more tricky because it is necessary to guarantee that the candidates get the correct rank/weight. One possible solution for this

problem is to have, per candidate, one verification code for each possible rank/weight. In our vote constructions, this can be achieved by simply creating an extended receipt from the r_i values. The procedure for the voter verification is similar to the procedure for k-out-of-n approval votes. First, $\mathcal{M}$ must reveal the k individual verification codes $w_{s_l^*} : \forall s_l^* \in S^*$, instead of σ. Second, $\mathcal{M}$ adds to the receipt table T of size n by t, where each entry $T_{i,j}$ contains the confirmation code for candidate i with respect to rank/weight j. $T_{i,j}$ is defined as $T_{i,j} = r_i - jc = w_i + t_i c - jc$, where t_i is the rank/weight given to option/candidate i. Thus, for any candidate i, in column $j = t_i$ we have $T_{i,j} = w_i$. Finally, the voter checks if $T_{s_l^*, t_{s_l^*}^*} = w_{s_l^*} : \forall s_l^* \in S^*$. For the $[k_{min}, k_{max}]$-out-of-n cases we can, as in approval voting, use dummy candidates.

7. Conclusions

In this paper we have described a very flexible technique to encrypt a diversity of complex ballot types: $[k_{min}, k_{max}]$-out-of-n, weighted, ranked and multi-race ballots. Additionally, our technique can be tuned for either mix net or homomorphic tallying

We've also shown how to extract a single, constant size, verification code from the vote encryption, which we have proved to be honest-verifier zero-knowledge. When compared with other confirmation code techniques, our approach is the only that supports a variety of complex ballots formats with a single, constant size, verification code.

To use a single verification code we require the voter to have access to a trusted piece of software/hardware to help him compute the verification code from the public election data and hers secret selections. On the other hand, other vote confirmation techniques, that use one verification code for each vote selection, rapidly become unusable with the increase of the ballot complexity. Nevertheless, we have also shown how to create a more traditional code voting receipt from our constructions, which allows the voter to verify the correct vote encryption without the need of a trusted device.

Taking into account the reduced complexity of the recent shuffle by Bayer and Groth [7], our constructions have a computational cost similar to the MarkPledge3 [32] but present shorter proofs. Like the others protocols from the MarkPledge family, the constructions described in this paper offer a much higher soundness than the one provided by simple cut-and-choose techniques, e.g. Benaloh challenges. On the other hand, the use of Benaloh challenges in conjunction with Groth's techniques [28] to prove a valid ballot encryption will in most cases be more efficient (at the cost of the voter's vote verification soundness).

Our technique is safe to use in a coercion free environment, although in non-coercion free environments the vote protocol designers must pay special attention to the challenge generation to prevent coercion.

As future work we aim to design and implement a prototype of a voting system using the verification methods proposed in this paper.

Acknowledgments

The author would like to thank Peter Y. A. Ryan, Rolf Haenni, Philipp Locher and Jurlind Budurushi, for helpful discussions and suggestions about this work, and to the anonymous reviewers for their very useful comments.

References

[1] Ben Adida. Helios: web-based open-audit voting. In *SS'08: Proceedings of the 17th USENIX Security symposium*, pages 335–348, Berkeley, CA, USA, 2008. USENIX Association.

[2] Ben Adida. Helios: Web-based open-audit voting. In *17th USENIX Security Symposium*, 2008.

[3] Ben Adida and Andrew Neff. Ballot casting assurance. In *EVT' 06*, Vancouver, B.C., Canada, August 2006. USENIX/ACCURATE.

[4] Ben Adida and Andrew Neff. Efficient receipt-free ballot casting resistant to covert channels. In *EVT/WOTE' 09*, Montreal, Canada, August 2009. USENIX/ACCURATE/IAVOSS.

[5] Ben Adida and Ronald L. Rivest. Scratch & vote: self-contained paper-based cryptographic voting. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES '06, pages 29–40, New York, NY, USA, 2006. ACM.

[6] Roberto Araújo, Ricardo Custodio, and Jeroen van de Graaf. A verifiable voting protocol based on farnel. In David Chaum, Markus Jakobsson, Ronald Rivest, Peter Ryan, Josh Benaloh, Mirosław Kutyłowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 274–288. Springer Berlin / Heidelberg, 2010.

[7] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer Berlin Heidelberg, 2012.

[8] Josh Benaloh. Simple verifiable elections. In *EVT 2006*, Berkeley, CA, USA, 2006. USENIX Association.

[9] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT' 07*, Boston, MA, USA, August 2007. USENIX/ACCURATE.

[10] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In Walter Fumy, editor, *Advances in Cryptology EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 318–333. Springer Berlin Heidelberg, 1997.

[11] Felix Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In DongHo Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 32–47. Springer Berlin Heidelberg, 2006.

[12] Emmanuel Bresson and Jacques Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In AgnesHui Chan and Virgil Gligor, editors, *Information Security*, volume 2433 of *Lecture Notes in Computer Science*, pages 272–288. Springer Berlin Heidelberg, 2002.

[13] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer Berlin Heidelberg, 2003.

[14] David Chaum. Secret-ballot receipts: True voter-verifiable election. *IEEE Security & Privacy*, 02(1):38–47, 2004.

[15] David Chaum. Punchscan, 2011. `http://www.punchscan.org/`.

[16] David Chaum, Richard Carback, Jeremy Clark, Aleksander Essex, Stefan Popoveniuc, Ronald L. Rivest, Peter Y. A. Ryan, Emily Shen, and Alan T. Sherman. Scantegrity ii: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *Usenix/Accurate EVT 08*, 2008.

[17] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi Vora. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security & Privacy*, May/June, 2008.

[18] David Chaum, Eugne Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *Advances in Cryptology CRYPTO 91*, volume 576 of *Lecture Notes in Computer Science*, pages 470–484. Springer Berlin Heidelberg, 1992.

[19] David Chaum and Torben Pedersen. Wallet databases with observers. In *CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.

[20] David Chaum, Peter Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679 of *LNCS*, pages 118–139, Milan, Italy, September 2005. Springer.

[21] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[22] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-

authority election scheme. In Walter Fumy, editor, *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 103–118, Konstanz, Germany, May 1997. Springer.
[23] Service Central de Legislation Luxembourg. Texte coordonn de la loi lectorale.
[24] Denise Demirel, Richard Frankland, and Melanie Volkamer. Readiness of various evoting systems for complex elections. *Technische Universtität Darmstadt, Tech. Rep. TUD-CS-2011-0193*, pages 1–14, 2011.
[25] T. ElGamal. A public-key cryptosystem and signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4):469–472, July 1985.
[26] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.
[27] Kristian Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010. `http://eprint.iacr.org/`.
[28] Jens Groth. Non-interactive zero-knowledge arguments for voting. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 467–482. Springer Berlin Heidelberg, 2005.
[29] James Heather, Peter Ryan, and Vanessa Teague. Pretty good democracy for more expressive voting schemes. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security ESORICS 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 405–423. Springer Berlin / Heidelberg, 2010.
[30] Sven Heiberg, Helger Lipmaa, and Filip Van Laenen. On e-vote integrity in the case of malicious voter computers. In *Proceedings of the 15th European conference on Research in computer security*, ESORICS'10, pages 373–388, Berlin, Heidelberg, 2010. Springer-Verlag.
[31] Rui Joaquim, Paulo Ferreira, and Carlos Ribeiro. Eviv: An end-to-end verifiable internet voting system. *Computers & Security*, 32(0):170 – 191, 2013.
[32] Rui Joaquim and Carlos Ribeiro. An efficient and highly sound voter verification technique and its implementation. In *VOTE-ID 2011*, volume 7187 of *LNCS*, pages 104–121, Tallinn, Estonia, 2012. Springer.
[33] Mirosław Kutyłowski and Filip Zagórski. Scratch, click & vote: E2e voting over the internet. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*, pages 343–356. Springer, 2010.
[34] Victor Morales-Rocha, Miguel Soriano, and Jordi Puiggalí. New voter verification scheme using pre-encrypted ballots. *Comput. Commun.*, 32:1219–1227, May 2009.
[35] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer, September 2006.
[36] Tal Moran and Moni Naor. Split-ballot voting: everlasting privacy with distributed trust. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 246–255, New York, NY, USA, 2007. ACM.
[37] Moni Naor and Adi Shamir. Visual cryptography. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 1995.
[38] C. Andrew Neff. Practical high certainty intent verification for encrypted votes, 2004.
[39] Ersin Öksüzoğlu and Dan S. Wallach. Votebox nano: a smaller, stronger fpga-based voting machine. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, EVT/WOTE'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.
[40] Stefan Popoveniuc and Ben Hosp. An introduction to punchscan. In David Chaum, Markus Jakobsson, Ronald Rivest, Peter Ryan, Josh Benaloh, Mirosław Kutyłowski, and Ben Adida, editors, *Towards Trustworthy Elections*, volume 6000 of *Lecture Notes in Computer Science*, pages 242–259. Springer Berlin / Heidelberg, 2010.
[41] Ronald Rivest. The three ballot voting system, September 2006. `http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf`.
[42] Ronald L. Rivest and Warren D. Smith. Three voting protocols: Threeballot, vav, and twin. In *EVT 2007*, pages 16–16, Berkeley, CA, USA, 2007. USENIX Association.
[43] Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, and Zhe Xia. Prêt à voter: a voter-verifiable voting system. *Trans. Info. For. Sec.*, 4:662–673, December 2009.
[44] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In *17th International Workshop on Security Protocols*. Springer-Verlag, 2009.
[45] Daniel Sandler, Kyle Derr, and Dan S. Wallach. Votebox: a tamper-evident, verifiable electronic voting system. In *Proceedings of the 17th conference on Security symposium*, pages 349–364, Berke-

ley, CA, USA, 2008. USENIX Association.

[46] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[47] Bjrn Terelius and Douglas Wikstrm. Proofs of restricted shuffles. In DanielJ. Bernstein and Tanja Lange, editors, *Progress in Cryptology AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer Berlin Heidelberg, 2010.

[48] Prêt à Voter. Prêt à voter web site, 2014. `http://www.pretavoter.com`.