scientific reports

OPEN

Proving vote correctness in the IVXV internet voting system

Taaniel Kraavi[1] & Jan Willemson[2]✉

This paper studies the practical aspects of adding zero-knowledge proofs of vote correctness to Internet voting, specifically to the IVXV system used in Estonia. We discuss various available alternatives and present a concrete instantiation based on Bulletproofs together with implementation details and benchmarking results. As IVXV currently uses the ElGamal cryptosystem with a 3072-bit prime modulus for vote encryption, but Bulletproofs work most efficiently on elliptic curves, a group switching solution is also implemented and benchmarked. Despite all the extra work required, our solution is very performant and well capable of sustaining the load of votes, even during peak vote submission periods.

Keywords Internet voting, Vote correctness, Zero-knowledge proofs, Bulletproofs

Holding regular elections is the cornerstone of a democratic society. However, with citizens becoming increasingly mobile in the contemporary world, gathering them in the same geographical location just for one day to cast their votes becomes less and less of an option[1].

Thus, a reliable remote voting mechanism is needed to ensure the generality of the elections. There are two main alternatives here – postal and Internet voting. However, the security properties of postal voting are strictly worse than those of vote casting over the Internet (i-voting)[2]. Thus, building a robust Internet voting system is inevitable for a sustainable democratic system (at least in the long run).

Of course, this task presents many challenges. We will need a reliable voter authentication mechanism to ensure eligibility and uniformity, and strong encryption to guarantee vote secrecy in transit. At the same time, the system must provide means for independent auditing of all the key steps. This is especially challenging because some of these requirements (most notably vote secrecy and system auditability) are inherently contradictory[3,4].

In this paper, we are going to concentrate on a specific aspect of verifiability, namely checking that the plaintext under ballot encryption actually corresponds to a valid candidate identifier. Note that in a typical Internet voting scheme, the vote collection server can not just verify the plaintext validity by decrypting the ballot as it does not have access to the decryption key.

On the other hand, an invalid plaintext can lead to a number of problems. In the case of homomorphic tallying systems, an incorrect ballot may contain double or negative votes. In the case of non-homomorphic (e.g. mixnet-based) voting systems, an invalid ballot plaintext can be used in a coercive scenario as a proof of forced abstention when the coercer is able to audit the plaintext votes after decryption. In more elaborate scenarios, an incorrect ballot plaintext can be used to broadcast sensitive information[5] or breach the secrecy of a whole group of voters[6].

Luckily, plaintext audits can be implemented using zero-knowledge proofs, and this paper proposes an instantiation of this technique aimed to be used in real elections. We base our study on the IVXV Internet voting system used in Estonia. On one hand, Internet voting has been used in legally binding elections in Estonia since 2005[7]. Still, even after 19 years of use, the system is missing proofs of ballot plaintext correctness. One of the reasons why this issue could have been ignored for so long is that submitting an incorrect ballot is far from being trivial. The voter would first have to implement a voting client of their own. While possible in principle, only very few actual instances are known[8]. However, during the 2024 European Parliament elections, for the first time, an incorrect ballot actually reached the decryption phase[9]. Thus, the issue of incorrect ballots can no longer be ignored, and this has been the main motivation behind our current paper.

The paper is organised as follows. Section “Background” presents the basic setup of IVXV, followed by a description of existing approaches described in Section "Prior art and existing solutions". Section "Using Bulletproofs for vote correctness" provides the details of our proposed solution and Section “Results” presents implementation and benchmarking details. Finally, Section "Conclusions and future work" draws some conclusions and sets directions for future work.

[1]Department of Software Science, TalTech, Ehitajate tee 5, 19086 Tallinn, Estonia. [2]Cybernetica, Narva mnt 20, 51009 Tartu, Estonia. ✉email: jan.willemson@cyber.ee

Background

IVXV was released in 2017 with the aim of improving the individual verifiability of many system components of Estonian Internet voting. Notably, the encryption scheme was changed from RSA-OAEP to ElGamal which enabled introducing both the proofs of correct decryption and the mixing process[10].

The four main parties involved in the scheme are the voter, the collector, the processor, and the tallying party[11]. Since the election organiser performs the tallying, the term *organiser* is used hereinafter to refer to the tallying party. I-voting itself is split into four main phases: setup, voting, processing, and tallying[11]. The phases may not overlap, and each phase begins when the previous stage ends.

During the setup phase, the election-specific key-pair used to encrypt and decrypt the votes cast in the election is generated following a (t, n)-threshold scheme. To decrypt the election results, at least t out of n key-holders must collaborate. In the current IVXV implementation, $n = 9$ and $t = 5$. The key holders are seven members of the National Electoral Committee and two employees of the State Electoral Office. During a normal election process, the key is not reassembled before the tallying phase, and no party is therefore able to decrypt arbitrary ballots during the voting and processing phases.

The key generation itself is a public ceremony where registered observers can see that the key is generated with the *key application* on an air-gapped computer with no persistent storage. (In Estonia, anyone can register themselves as an observer for an election [12 §19[4]]. Observers can observe parts of the setup, processing and tallying phases. There are also designated auditors appointed with the task of auditing all the key processes.)

Voters vote with an official *voting client*, which enables a voter to select their preferred candidate, encrypt the ballot, digitally sign it, and send it to the collector. The collector then returns to the voting client the vote qualifying elements, which the client verifies to attest whether the collector performed all required operations.

The *collector* is a combination of servers which perform multiple tasks. Upon receiving a ballot, the collector performs feasible validity checks, such as verifying that the signature on the ballot is valid and that the voter is eligible to vote. If the received ballot passes the checks, the collector registers the ballot with an external registration service and stores it in the digital ballot box. Finally, the collector returns the registration proof which the client verifies to ensure that the collector indeed registered the ballot and did not drop it.

After the voting phase concludes, the processor obtains the digital ballot box and other integrity information such as checksums and logs from the collector. In practice, the election organiser also holds the role of the processor. Using the *processing application*, the organiser verifies the integrity of the data supplied by the collector, including the integrity of the ballot box and of the signatures on the ballots. The organiser also discards votes overridden by re-i-voting and paper voting to keep only eligible i-votes, and strips the digital signatures from these votes.

Finally, the organiser uses a re-encryption mix-net to cryptographically anonymise the ballots, since otherwise they could still be correlated with the stripped digital signatures. While the processing phase is observable, the data generated by the processing application can only be made available for designated auditors working under an NDA, since it contains sensitive information. For example, it contains the ID codes of voters and ballot timestamps which could be used to determine whether a voter re-voted, hence enabling coercion attacks.

The anonymised encrypted ballots are then transferred to an air-gapped computer for decryption with the key application. In addition to decrypting the ballots, the key application also verifies whether the decrypted results are valid, tallies the valid results, and generates zero-knowledge proofs of correct decryption. While the decryption phase is observable, not all outputs can be made publicly available in this phase either.

The problem arises when a voter has managed to encrypt a value that does not belong to the list of valid candidate identifiers. Currently, the vote collector has no way of determining whether the submitted cryptogram corresponds to a valid candidate, so a discrepancy would only be detected after decryption. If incorrect identifiers would be published as part of the end result, it could ease forced abstention attacks, where the coercer essentially forces the voter to encrypt a garbage value instead of a vote. Also, some more elaborate attacks have been used in the literature also making use of invalid candidate identifiers[5,6].

To address this issue, the current practice in IVXV is to not publish the invalid votes, and these invalid votes are not made available to the observers either. As a result, the observers must trust an auditor with verifying that the key application did not arbitrarily declare votes as invalid.

It follows that the decryption process currently produces conditional outputs: if no invalid votes appear, everything can be made public. Otherwise, only the valid votes and their proofs can be made public. In the latter case, the tally is no longer verifiable by third parties, which hinders the universal verifiability of IVXV. A mechanism is therefore needed to prevent ballots from reaching the final anonymised ballot box, so that the decrypted ballot box could always be matched against the tally results and decryption proofs.

Prior art and existing solutions

The problem of identifying invalid votes before tallying is not new. Mathematical proofs of vote correctness were, to the best of our knowledge, first introduced by Cohen and Fischer in 1985[13]. The main idea of the scheme is that the voters prepare unmarked ballots which have the encryptions of a 'yes' and a 'no' value. Voters then prove that their ballot has indeed only those values without revealing which is which. To cast a vote, voters select the desired value and submit it to the election organiser. Finally, the organiser combines the votes and publishes the tally together with a proof that the latter is correct. However, the scheme is not threshold-based, and so the organiser can decrypt the individual votes[13]. Hence, the proofs of correctness are necessary only for the voters to verify the consistency of the tally, and not for the organiser, who can simply decrypt votes.

Benaloh and Yung later contributed an improvement to the scheme which addressed this problem by splitting the organiser into parties who must collaborate for computing the tally[14]. Their work also established the terminology of *marking* to designate voters selecting their choice. Benaloh then elaborated on those ideas further and introduced the use of secret sharing for privacy preservation, and of homomorphic encryption and tallying for the proof process[15].

All three schemes require interactive proofs of vote correctness[13–15] and serve to prove the correctness of the tally. The schemes therefore do not consider that a party may need to verify vote correctness before any tallying. In 1997, Cramer, Gennaro, and Schoenmakers first used zero-knowledge vote correctness proofs in the context of ElGamal ciphertexts[16]. Notably, the proofs are non-interactive (NIZKP), and verifiable without access to any secret values, but are still only usable with homomorphically tallied votes. This approach remains the most widely used proof system in homomorphic tally voting systems[17].

Unlike for voting systems with homomorphic tally[18–22], such proofs of correctness are not strictly required for the validity of traditional 'decrypt then tally' schemes. E-voting systems that do not rely on homomorphic tallying therefore often rely on other techniques.

In the Civitas voting scheme[18], the re-encryption proofs of Hirt and Sako[23] are used to prove that a vote is the re-encryption of an existing ciphertext. That is, encryptions for all valid candidates are published by the election authority, and the voter proves that their ballot is a re-encryption of one of these votes, without showing which. This paradigm was revisited and extended by Joaquim[24] who proposed the addition of an additional proof of structure. In the scheme, the ballot is obtained from the re-encryptions of multiple options and an additional ZKP is used to prove the structure of the combination. The structure proof is based on the proof of knowledge of representation by Brands[25], but is a novel addition for ElGamal by Joaquim..

In the Kryvos voting system[26], zero-knowledge succinct arguments of knowledge (zk-SNARK) are used to prove ballot validity. More specifically, the proof system due to Groth[27] is used to prove that the committed vote shares correspond to the valid choice space. VoteAgain[28] uses the zero-knowledge set-membership proofs by Bayer and Groth[29] to prove that a ballot represents a valid candidate. In the code-based SwissPost voting scheme[30] developed in Switzerland, voters use codes delivered to them by post to vote for specific candidates. A mechanism specific to their code-based implementation is also used to verify that a voter has cast a vote for a candidate they are allowed to vote for.

Selection criteria

The desired goal is to prove that an encrypted ballot contains a valid candidate identifier, without leaking which candidate the ballot is for. To simplify the problem, the encrypted ballot can be viewed as a binding and hiding commitment to a candidate identifier. This enables the use of 'commit-and-prove' techniques[31], where a statement is proven in zero-knowledge relative to a committed value. The stated problem can thus be reduced to the general task of proving set membership in zero-knowledge, i.e. that a secret value belongs to a set.

For minimal changes to IVXV, the chosen scheme should be compatible with finite field arithmetic, the (lifted) ElGamal cryptosystem, and rely only on the discrete logarithm problem (DLOG). Relying on the DLOG avoids introducing new security assumptions into IVXV. It follows that schemes based on the ElGamal cryptosystem and Pedersen commitments are preferable over other approaches.

We considered the state of the art zero-knowledge set membership proofs of Benarroch et al.[32] and of Campanelli, Hall-Andersen, and Kamp[33] which are both based on cryptographic accumulators. However, both approaches require an underlying commit-and-prove system for either range proofs, or proving arbitrary statements using constraint systems. As such, the use of zero-knowledge range proofs (ZKRP) instead of proving zero-knowledge set membership (ZKSM) could potentially reduce verification times and proof complexity even further, at the cost of losing the flexibility of arbitrary sets.

While tree-based approaches can yield constant-size proofs with fast verification times, they generally require the use of general-purpose zk-SNARKS[32]. As such, they do not satisfy our selection criteria. We also did not pursue pairing-based ZKSMs due to the additional hardness assumptions of pairings and since the ZKRP approach seemed promising.

Christ et al. summarised the state of the art regarding ZKRPs in a recent survey[34], although approaches based on lookup arguments[35] are not mentioned, and neither are some recent ZKRP protocols[36,37] which combine the covered techniques in interesting ways. The paper also omits the SwiftRange range proofs[38], and Flashproofs[39] by the same authors. There is also the survey by Deng et al.[40] which includes a comparative analysis of different ZKRP approaches, although it does not contain any benchmarks. It also provides a more complete history of range proofs than the survey by Christ et al., while being already a bit out-dated as of 2024.

By discarding hash-based range proofs and general polynomial commitments, the state of the art narrows down to Sharp[41] and Bulletproofs[42] with its potential improvements. While variants of Sharp appear more efficient than Bulletproofs[34,41] on paper, there are important nuances to consider because of differing security guarantees. By default, Sharp only provides 'relaxed' soundness, where the prover is only bound to a rational in the target range, instead of an integer[41]. Bulletproofs do not suffer form this limitation. However, Bulletproofs can only directly be used to prove that a number belongs to $[0, 2^n - 1]$ for some integer n while Sharp works for arbitrary ranges. Since Bulletproofs are homomorphic, it is possible to overcome this limitation in practice.

For Sharp, in the interactive setting, satisfying full soundness with a knowledge error of 2^{-128} would require 128 repetitions of the protocol[41]. In the non-interactive setting, Sharp can achieve full soundness with an additional commitment in a hidden order group (class group or RSA group). As such, the performance advantages of Sharp over Bulletproofs are likely lost when requiring full soundness. However, this remains unclear as Couteau et al. provided computational benchmarks only for their most optimised variant of Sharp, which does not make use of the hidden order group commitment.

Additionally, Sharp is a very new protocol, and it does not appear to have been implemented outside of research. Couteau et al. have not published their benchmarked Sharp implementation either. The existence of a reference and third party implementations of Bulletproofs[43,44] as well as its practical use in protocols[45] are strong arguments for Bulletproofs over Sharp.

While the original Bulletproofs are already efficient in practice, optimisations such as Bulletproofs+[46] and Bulletproofs++[47] have been proposed since. Bulletproofs+ reduce the proof size for a 32-bit range by 16%, while the performance remains comparable to that of Bulletproofs for non-aggregated proofs[46]. Bulletproofs++ reduce the $O(n)$ scalar multiplications asymptotically needed by Bulletproofs(+) to $O(n/\log(n))$ multiplications, while also further reducing the proof size[47].

Using Bulletproofs for vote correctness

Setting

Let $\mathbb{G}_p, \mathbb{G}_q$ be two DLOG-groups such that $q \ll p$. Let b_q denote the bit-length of q.

Let x represent a vote and let b_x be the maximal bit-length of a vote such that $2^{b_x} \ll q$. Let a, b be two integers with $0 \leq a < b < 2^{b_x}$. Thus, for any $x \in [a, b]$, it holds that x is the same in both $\mathbb{Z}_p$ and $\mathbb{Z}_q$, and that $x < 2^{b_x}$. Let a (resp. b) represent the lowest (resp. highest) candidate number among candidates who the voters can vote for. Without loss of generality, let the candidate numbers be consecutive between a and b. On the off-chance that this is not the case, the available candidate numbers can be mapped onto a consecutive integer range.

Let $\mathrm{pk} \in \mathbb{G}_p$ be the public key of the lifted ElGamal cryptosystem and let g_p be a generator of $\mathbb{G}_p$. The encryption of $x \in [a, b]$ with randomness $r \xleftarrow{\$} \mathbb{Z}_p$ is then

$$\mathrm{Enc}_{\mathrm{pk}}(x; r) = \left(g_p^r, g_p^x \cdot \mathrm{pk}^r\right) = (y, X).$$

More generally, we will use capital letters to denote Pedersen commitments. Standalone, X can be viewed as a Pedersen commitment of x with randomness r in $\mathbb{G}_p$.

Let b_c represent the bit-length of a verifier's challenge in a Σ-protocol. For some security parameter λ, it must hold that $b_c \geq \lambda$ for security with no repetitions of the protocol. In practice, $\lambda \geq 128$. Let $b_\perp$ be a parameter controlling the probability of aborts, i.e. the probability that the Σ-protocol must be restarted to avoid leaking information about the secret. Finally, the parameters $b_x, b_c, b_\perp, b_q$ must satisfy the relation $b_x + b_c + b_\perp < b_q$ which is necessary to avoid modular reductions in $\mathbb{G}_p$ and $\mathbb{G}_q$ for the protocol computations.

Range proof for concrete ranges

Bulletproofs are a zero-knowledge proof protocol without a trusted setup which can be used for range proofs, but also for proofs for arithmetic circuits. Bulletproofs rely only on the discrete logarithm assumption, and are made non-interactive using the Fiat-Shamir heuristic[42]. In this work, we only consider Bulletproof range proofs in the non-interactive setting.

Bulletproofs do not directly prove that a value lies in an arbitrary integer range. Rather, given a Pedersen commitment X to x with randomness r, they prove the following relation[42]:

$$\left\{(g, h \in \mathbb{G}, X, b_x; x, r \in \mathbb{Z}_q) : X = g^x h^r \wedge x \in [0, 2^{b_x} - 1]\right\},$$

where $\mathbb{G}$ is a DLOG-group. However,

$$a \leq x \leq b < 2^{b_x} \iff 0 \leq x - a < 2^{b_x} \wedge 0 \leq b - x < 2^{b_x} \wedge b < 2^{b_x}$$

and we can use the additive homomorphism of Pedersen commitments to combine two Bulletproofs and prove that $x \in [a, b]$ for arbitrary $a, b \in \mathbb{N}$ as required. Note that $2^{b_x} \ll q$ guarantees that indeed $0 \leq x - a < 2^{b_x}$ even when taken modulo q since $2^{b_x} < q - (x - a) \leq q$, and similarly for $b - x$.

Let g_q, h be two generators of $\mathbb{G}_q$ such that $\log_{g_q}(h)$ is not known. Let π_a, π_b be the Bulletproofs asserting that $x - a \in [0, 2^{b_x} - 1]$ and $b - x \in [0, 2^{b_x} - 1]$ with commitments

$$\begin{aligned} r_a &\xleftarrow{\$} \mathbb{Z}_q, & C_a &\leftarrow g_q^{x-a} \cdot h^{r_a} \\ r_b &\xleftarrow{\$} \mathbb{Z}_q, & C_b &\leftarrow g_q^{b-x} \cdot h^{-r_b}. \end{aligned}$$

To prove that $x \in [a, b]$, it remains to show that x is the same for both $X_a \leftarrow g_q^a \cdot C_a$ and $X_b \leftarrow g_q^b \cdot (C_b)^{-1}$, since

$$\begin{aligned} g_q^a \cdot C_a &= g_q^a \cdot g_q^{x-a} \cdot h^{r_a} = g_q^x \cdot h^{r_a} \\ g_q^b \cdot \left(C_b\right)^{-1} &= g_q^b \cdot g_q^{x-b} \cdot h^{r_b} = g_q^x \cdot h^{r_b}. \end{aligned}$$

A ZKP of discrete logarithm equality is thus needed to prove that X_a and X_b are commitments to the same x. Note that given C_a, C_b, anyone can compute X_a, X_b since g_q, h, a, b are publicly known.

Discrete logarithm equality across groups

Let $(y, X), (C_a, \pi_a), (C_b, \pi_b)$ be given to the verifier. By verifying π_a, π_b and computing X_a, X_b, the verifier gains assurance that

1. X_a is a commitment for x_a such that $x_a \geq a$,
2. X_b is a commitment for x_b such that $x_b \leq b$.

Then, to verify that (y, X) is an encryption of $x \in [a, b]$, the verifier must additionally be able to verify in zero-knowledge that

$$x = x_a = x_b$$

for the committed values. More formally, to prove that (y, X) is an encryption of $x \in [a, b]$, a zero-knowledge proof must be given for the relation

$$\mathcal{R}_{\mathrm{Veq}} = \left\{ \begin{array}{l} \big((y, X, X_a, X_b), \\ (x, r, r_a, r_b)\big) \end{array} \middle| \begin{array}{l} y = g_p^r \wedge X = g_p^x \cdot \mathrm{pk}^r \wedge \\ X_a = g_q^x \cdot h^{r_a} \wedge \\ X_b = g_q^x \cdot h^{r_b} \end{array} \right\}.$$

A ZKP for $\mathcal{R}_{\mathrm{Veq}}$ therefore proves that a vote for an eligible candidate was encrypted without leaking who the vote is for.

When $p = q$, the proof is a variant of proving discrete logarithm equality in a group, for which an efficient approach has been given by Chaum and Pedersen[48]. However, the difficulty lies in proving this efficiently when $p \neq q$, which is the case here, i.e. proving discrete logarithm equality across different groups. For the latter problem, an approach using Pedersen commitments was presented by Chase et al.[49]. More formally, the technique by Chase et al. gives a ZKP for the relation

$$\mathcal{R}_{\mathrm{DLeq}} = \left\{ \begin{array}{l} \big((X_1, X_2), \\ (x, r_1, r_2)\big) \end{array} \middle| X_1 = g_1^x \cdot h_1^{r_1} \wedge X_2 = g_2^x \cdot h_2^{r_2} \right\}$$

although a range proof for $x \in [0, 2^{b_x} - 1]$ is needed as part of the technique. This range proof must be knowledge-sound, and the authors themselves propose to use Bulletproofs for this[49], 5. The protocol is therefore ideal for the current use case, since such a range proof is required regardless, i.e, π_a, π_b. Moreover, since the protocol in[49] is a public-coin protocol, it can also be made non-interactive with the Fiat-Shamir transform.

For efficiency, instead of first proving $x_a = x_b$ for commitments in $\mathbb{G}_q$ and then proving $\mathcal{R}_{\mathrm{DLeq}}$, the technique by Chase et al. can be extended to prove the entirety of $\mathcal{R}_{\mathrm{Veq}}$. The resulting Σ-protocol Π_{Veq} for proving the relation $\mathcal{R}_{\mathrm{Veq}}$ is given on Figure 1.

Security guarantees

The mask *k* is necessary to hide information with random noise, otherwise the verifier could trivially extract *x* from *z* with $x = z/c$. Since operations are performed over the integers, the traditional approach of picking the mask uniformly at random from the underlying group is not feasible. As such, not all values of *k* mask *cx* 'sufficiently'. Indeed, not all values of *z* are equally likely to occur for $z < 2^{b_x + b_c}$ or $z \geq 2^{b_x + b_c + b_\perp}$, and so the protocol must be aborted if *z* leaks information about *x*.

More formally, the abort condition follows from the following lemma[49,50]:

Lemma 1 ([50], Lemma 1) In the non-aborting case, the value *z* in the transcript of an honest protocol execution is uniformly distributed in $\{2^{b_x + b_c}, \ldots, 2^{b_x + b_c + b_\perp} - 1\}$. An honest prover aborts with probability $2^{-b_\perp}$.

Notably, the verifier gets no information about *k* due to the hiding property of the Pedersen commitments transmitted with the first message. Thus, it is infeasible for the verifier to learn anything about *cx* regardless of whether the prover aborts the protocol.

Interactive proofs have certain shortcomings compared to non-interactive proofs. For example, non-transferability restricts the auditability of i-voting, which is the very situation that this work aims to improve. Moreover, the three move communication might not be practical from an implementation viewpoint, especially when aborts are involved.

The protocol can be made non-interactive using the Fiat-Shamir transform[51,52] with aborts[53]. To prevent the prover from creating unsound proofs by adaptively choosing their statement, the statement and protocol messages preceding the challenge generation must all be included in the challenge seed[54].

The proofs that follow are based on the proofs in[49], adapted for the proposed protocol in the non-interactive setting. This simplifies the proofs, since the prover only outputs a non-aborting transcript and the abort cases no longer need to be considered[49], §4.2. For our proofs, challenges must be obtained from the programmable random oracle $\mathcal{O}$ with $c \leftarrow \mathcal{O}(\mathrm{st}, \alpha)$, where

- st is the statement $\big((y, X), (g_p, \mathrm{pk}), (g_q, h), (a, b)\big)$,
- α is the first message of the protocol, i.e. (w, K, K_a, K_b).

Additionally, we require setting $b_c \geq 2\lambda$ for collision resistance with Fiat-Shamir due to the birthday attack[55], §7.3.

Theorem 1 The non-interactive protocol Π_{Veq} is perfectly complete.

Proof In the non-interactive setting, the prover never aborts. First, for an honest prover, the equation (i)

$$g_p^s = g_p^{t+cr} = g_p^t \cdot g_p^{cr} = w \cdot \left(g_p^r\right)^c = w \cdot y^c$$

Prover		**Verifier**
Input: x, r, r_a, r_b		Input: $y, X, X_a, X_b, \pi_a, \pi_b$
$k \xleftarrow{\$} [0, 2^{b_x+b_c+b_\perp} - 1]$		
$t \xleftarrow{\$} \mathbb{Z}_p$		
$(t_a, t_b) \xleftarrow{\$} \mathbb{Z}_q^2$		
$w \leftarrow g_p^t$		
$K \leftarrow g_p^k \cdot \mathsf{pk}^t$		
$K_a \leftarrow g_q^k \cdot h^{t_a}$		
$K_b \leftarrow g_q^k \cdot h^{t_b}$		
	$\xrightarrow{w, K, K_a, K_b}$	
		$c \xleftarrow{\$} [0, 2^{b_c} - 1]$
	$\xleftarrow{c}$	
Check $c \in [0, \ldots, 2^{b_c} - 1]$		
$z \leftarrow k + c \cdot x$ (in $\mathbb{Z}$)		
Abort if $z \notin [2^{b_x+b_c}, 2^{b_x+b_c+b_\perp} - 1]$		
$s \leftarrow t + c \cdot r \pmod p$		
$s_a \leftarrow t_a + c \cdot r_a \pmod q$		
$s_b \leftarrow t_b + c \cdot r_b \pmod q$		
	$\xrightarrow{z, s, s_a, s_b}$	
		Accept iff
		$z \in [2^{b_x+b_c}, 2^{b_x+b_c+b_\perp} - 1]$,
		$s \in \mathbb{Z}_p$, $(s_a, s_b) \in \mathbb{Z}_q^2$,
		$g_p^s = w \cdot y^c$, (i)
		$g_p^z \cdot \mathsf{pk}^s = K \cdot X^c$, (ii)
		$g_q^z \cdot h^{s_a} = K_a \cdot X_a^c$, (iii)
		$g_q^z \cdot h^{s_b} = K_b \cdot X_b^c$ (iv)

Fig. 1. Π_{Veq}: a Σ-protocol for proving $\mathcal{R}_{\text{Veq}}$ based on the proof of cross-group discrete logarithm equality by Chase et al.[49].

is satisfied. Additionally, equation (ii)

$$g_p^z \cdot \text{pk}^s = g_p^{k+cx} \cdot \text{pk}^{t+cr} = g_p^k \cdot \text{pk}^t \cdot \left(g_p^x \cdot \text{pk}^r\right)^c = K_p \cdot X_p^c$$

is also satisfied. Equations (iii) and (iv) are proven similarly. □

Theorem 2 The non-interactive protocol for the relation $\mathcal{R}_{\text{Veq}}$ is honest verifier zero-knowledge under the DLOG assumption for $\mathbb{G}_p$ and $\mathbb{G}_q$ in the (programmable) random oracle model.

Proof The simulator takes as input the public parameters, including the group descriptions and the public key pk. It also has access to the random oracle and can program it with input-output pairs.

First, the simulator samples uniformly at random

$$c \xleftarrow{\$} [0, 2^{b_c} - 1], z \xleftarrow{\$} [2^{b_x+b_c}, 2^{b_x+b_c+b_\perp} - 1],$$

$$s \xleftarrow{\$} \mathbb{Z}_p, s_a \xleftarrow{\$} \mathbb{Z}_q, s_b \xleftarrow{\$} \mathbb{Z}_q.$$

Since K satisfies the equation

$$K = g_p^k \cdot \text{pk}^t = g_p^k \cdot \text{pk}^{s-cr} \cdot \left(g_p^{cx} \cdot g_p^{-cx}\right) = g_p^z \cdot \text{pk}^s \cdot \left(g_p^x \cdot \text{pk}^r\right)^{-c},$$

the simulator computes the commitment K as

$$K \leftarrow g_p^z \cdot \mathrm{pk}^s \cdot \left(X_p^c\right)^{-1}$$

and commitments K_a and K_b analogously. It also computes the commitment w as

$$w \leftarrow g_p^s \cdot \left(y^c\right)^{-1}.$$

Finally, the simulator programs the random oracle $\mathcal{O}$ such that

$$c \leftarrow \mathcal{O}\big(\mathrm{st}, (w, K, K_a, K_b)\big)$$

and then outputs the transcript

$$\big((w, K, K_a, K_b), c, (z, s, s_a, s_b)\big).$$

Since non-interactive Bulletproofs are fully zero-knowledge in the random oracle model[42], §4.4, π_a and π_b do not affect the zero-knowledge property of the protocol. However, an unbounded adversary can recover r_p from y and therefore decrypt X_p to recover x, thus breaking the zero-knowledge property. This is not feasible for a computationally bounded adversary, and thus the protocol can only satisfy computational zero-knowledge.

It remains to show that the distributions of the real and simulated transcripts are computationally indistinguishable in the programmable random oracle model.

1. In both the real and simulated executions, the challenge c is independently and uniformly sampled from $[0, 2^{b_c} - 1]$.
2. By Lemma 1, z is distributed uniformly in $[2^{b_x+b_c}, 2^{b_x+b_c+b_\perp} - 1]$ in both real and simulated transcripts since there can be no aborted transcripts.
3. In both transcripts, the values s, s_a, and s_b are uniformly distributed. Indeed, in the real protocol execution, cr is perfectly masked in $\mathbb{Z}_p$ by t, since t is sampled uniformly and independently from c, and similarly for s_a, s_b in $\mathbb{Z}_q$.
4. Since s is uniformly distributed in $\mathbb{Z}_p$, $K = g_p^k \cdot \mathrm{pk}^t = g_p^k \cdot \mathrm{pk}^{s-cr}$ is uniformly distributed in $\mathbb{G}_p$. Uniform distribution can be similarly shown for K_a, K_b. Furthermore, $w = g_p^t = g_p^{s-cr}$, and so w is also uniformly distributed in $\mathbb{G}_p$.□

Theorem 3 Let κ_{rp} be the knowledge error of $\pi_r p$. The non-interactive protocol for the relation $\mathcal{R}_{\mathrm{Veq}}$ is 2-special sound with knowledge error $\kappa = 2^{-b_c} + \kappa_{rp}$ under the DLOG assumption for $\mathbb{G}_p$ and $\mathbb{G}_q$ in the (programmable) random oracle model.

Proof Let there be an extractor algorithm Ext which is given as input the range proof π_a (alternatively π_b) and two accepting transcripts $((w, K, K_a, K_b), c, (z, s, s_a, s_b))$ and $((w, K, K_a, K_b), \dot{c}, (\dot{z}, \dot{s}, \dot{s_a}, \dot{s_b}))$ with $c \neq \dot{c}$. Such transcripts can exist due to the programmability of the random oracle, where, after outputting c, the oracle is reprogrammed to output c' for the same input. The extractor also has access to the public parameters, including pk.

The extractor then recovers x_p, x_q, r, r_a, r_b such that $y = g_p^r$ and

$$X = g_p^{x_p} \cdot \mathrm{pk}^r, \quad X_a = g_q^{x_q} \cdot h^{r_a}, \quad X_b = g_q^{x_q} \cdot h^{r_b}$$

with the following steps:

1. Bulletproof range proofs are arguments of knowledge[42] and are therefore extractable. Ext can thus extract $((x_q - a)^*, r_a^*)$ with the knowledge-extractor of π_a, except with some small failure probability κ_{rp}. Since a is publicly known, the extractor can further recover (x_q^*, r_a^*) such that $X_a = g_q^{x_q^*} \cdot h^{r_a^*}$ and $x_q^* < 2^{b_x}$.
2. From the two accepting transcripts with distinct challenges $c \neq \dot{c}$, the extractor selects the pairs

- $\big((K, c, z, s), (K, \dot{c}, \dot{z}, \dot{s})\big)$,
- $\big((K_a, K_b, c, z, s_a, s_b), (K_a, K_b, \dot{c}, \dot{z}, \dot{s_a}, \dot{s_b})\big)$.

By defining

$$x_p = \frac{z - \dot{z}}{c - \dot{c}}, \quad r = \frac{s - \dot{s}}{c - \dot{c}}$$

Ext extracts an opening of $X = g_p^{x_p} \cdot \mathrm{pk}^r$ since

$$\frac{z - \dot{z}}{c - \dot{c}} = \frac{k + cx_p - (k + \dot{c}x_p)}{c - \dot{c}} = \frac{x_p(c - \dot{c})}{c - \dot{c}} = x_p$$

and the recovery of *r* holds similarly. Ext then extracts openings for $X_a = g_q^{x_q} \cdot h^{r_a}$ and $X_b = g_q^{x_q} \cdot h^{r_b}$ in the same manner. The extraction fails with probability 2^{-b_c} which is a standard result[56], Theorem 1. By the binding property of Pedersen commitments X, K, K_a, K_b and of ElGamal,

(a) $y = g_p^r$
(b) $(x_q, r_a) = (x_q^*, r_a^*)$
(c) $(z - cx_p, s - cr) = (\dot{z} - \dot{c}x_p, \dot{s} - \dot{c}r)$
(d) $(z - cx_q, s_a - cr_a, s_b - cr_b) = (\dot{z} - \dot{c}x_q, \dot{s_a} - \dot{c}r_a, \dot{s_b} - \dot{c}r_b)$

must all hold under the DLOG assumption.

3. Finally, the extractor returns (x_p, x_q, r, r_a, r_b).

It remains to show that $x_p = x_q$ holds over the integers. From verification checks (ii) and (iii), there must exist $k_p, k_q, u, \dot{u}, v, \dot{v} \in \mathbb{Z}$ such that

$$\begin{aligned} z &= k_p + c \cdot x_p + u \cdot p \qquad & z &= k_q + c \cdot x_q + v \cdot q \\ \dot{z} &= k_p + \dot{c} \cdot x_p + \dot{u} \cdot p \qquad & \dot{z} &= k_q + \dot{c} \cdot x_q + \dot{v} \cdot q\ . \end{aligned}$$

Because $z \in [2^{b_x+b_c}, 2^{b_x+b_c+b_\perp} - 1]$ for an accepting transcript, it follows that $(u, \dot{u}, v, \dot{v})$ are non-negative. By linear combination with respect to *p* and *q*,

$$\begin{aligned} (z - \dot{z}) &= (c - \dot{c})x_p + (u - \dot{u})p, \\ (z - \dot{z}) &= (c - \dot{c})x_q + (v - \dot{v})q. \end{aligned}$$

W.l.o.g., let $z - \dot{z}$ be positive, since if it is negative, then $\dot{z} - z$ is positive instead. From the choice of parameters and verification checks $z - \dot{z} < 2^{b_x+b_c+b_\perp} < 2^{b_q}$. Moreover, π_a ensures that $x_q < 2^{b_x}$, and so $|c - \dot{c}|x_q < 2^{b_x+b_c}$. It follows that $(z - \dot{z}) - |c - \dot{c}|x_q < 2^{b_q}$ and so $(v - \dot{v}) = 0$.

Equating the two representations of $z - \dot{z}$ with $(v - \dot{v}) = 0$ yields

$$\begin{aligned} (c - \dot{c})x_p + (u - \dot{u})p &= (c - \dot{c})x_q \\ (u - \dot{u})p &= (c - \dot{c})(x_q - x_p)\ . \end{aligned}$$

Since p is prime, it must divide $(c - \dot{c})$ or $(x_q - x_p)$, but $p > 2^{b_q} > 2^{b_c}$ and so it cannot divide $|c - \dot{c}|$. Therefore $p|(x_q - x_p)$ and so $x_q \equiv x_p \pmod p$. Since $x_q < p$ and $x_p < p$, no modular reduction takes place and so $x_q = x_p$ in $\mathbb{Z}$ as well. □

Proof size

A single Bulletproof uses $2 \cdot \lceil \log_2(b_x) \rceil + 4$ elements of $\mathbb{G}_q$ and 5 elements of $\mathbb{Z}_q$[42], §4.2. In our protocol, these numbers are doubled for non-aggregated Bulletproofs. Two aggregated Bulletproofs use only $2 \cdot \lceil \log_2(b_x) + 1 \rceil + 4$ elements of $\mathbb{G}_q$, while the number of elements of $\mathbb{Z}_q$ remains unchanged[42], §4.3.

By compressing the transcript of our protocol into (c, z, s, s_a, s_b), the transcript size is $\tau(2b_c + b_x + b_\perp + \lceil \log_2(p) \rceil + 2\lceil \log_2(q) \rceil)$, where τ represents the number of repetitions. In the non-interactive setting, $\tau = 1$ since the prover only sends a non-aborting transcript. For simplicity, we upper bound our non-interactive transcript size by 4 elements of $\mathbb{Z}_q$, and one element of $\mathbb{Z}_p$.

The full proof is formed by our protocol transcript, the two Bulletproofs, and X_a, X_b, which are two additional elements of $\mathbb{G}_q$. The total proof size, assuming aggregated Bulletproofs, is thus not larger than

- $2 + (2 \cdot \lceil \log_2(b_x) + 1 \rceil + 4)$ elements of $\mathbb{G}_q$,
- $4 + (5)$ elements of $\mathbb{Z}_q$,
- 1 element of $\mathbb{Z}_p$,

where the numbers in parentheses are due to Bulletproofs. We refer to Section "Benchmarks" for some concrete numbers based on these estimates.

Concrete instantiation

From Estonian election statistics[57], the largest number of candidates in an election since 1992 is 15322. However, this is the cumulative candidate count across all local governments. In any concrete municipality, the number of candidates a voter can vote for is much less, and candidate numbers are not global for local elections. The largest number of candidates unified throughout the country is only 1885[57]. Regardless of the election type, it is reasonable to assume that for the foreseeable future, no election will have more than 2^{16} candidates, and so $b_x = 16$.

IVXV uses finite field ElGamal in Group 15 from RFC3526[58], and so $b_p = 3071$. Given the current state of classical cryptanalysis, this corresponds to a security level of 128 bits, and so $\lambda = 128$. Since the protocol requires $b_c \geq \lambda$ for soundness without repetitions, $b_c = 128$ satisfies this requirement, but only in the interactive setting. In the non-interactive setting, the hash function must have a range of $\{0, 1\}^{2\lambda}$ to achieve a collision resistance of λ bits[55], §7.3. In practice, $b_c = 256$ is therefore required for a 128-bit security level against confidentiality and soundness attacks.

It remains to find values for $b_\perp$ and b_q such that $b_x + b_c + b_\perp < b_q$ and $b_q \ll b_p$. A popular and performant implementation of Bulletproofs[59] is available over the ristretto255 curve[60], which is itself built on top of Curve25519. While using ristretto255 would be beneficial for the performance, it has a group size of 2^{252}, which is incompatible with $b_c = 256$. By weakening the soundness guarantees to 112 bits of soundness, the challenge size can be set to $b_c = 224$. Then, $b_\perp = 251 - 16 - 224 = 11$ and the probability of aborts becomes 2^{-11}. This remains small and the computational burden of occasional aborts is only borne by the prover. Since the non-interactive version of the protocol is full computational zero-knowledge, the zero-knowledge guarantees are not impacted and a 128-bit security level against confidentiality attacks is therefore maintained.

To achieve 128-bit soundness guarantees, a larger curve such as P-384 should be used. By setting $b_q = 384$, it follows that $b_\perp < 112$, and so the probability of aborts is 2^{-112}. While $b_\perp$ could be lowered if needed for compatibility with a smaller curve that still accommodates $b_c = 256$, P-384 is a popular curve choice. Notably P-384 is used by the Estonian ID card for digital signatures[61], 135.

Results

To analyse and benchmark the computational cost of verification, we developed a prototype of the full protocol in Go[62]. We chose Go since the IVXV vote collector is also written in Go[63], and the common language therefore simplified real-system benchmarking. In addition, the prototype can hopefully serve as a template for a production implementation of the protocol should the approach be approved by the election organiser.

We did not implement the Bulletproofs needed as part of the protocol from scratch. Rather, we took the open-source implementation by ING Bank[64]—hereinafter 'library'—as a basis. We did not find a Bulletproofs+ implementation for Go. Since the library uses secp2561k as a hardcoded elliptic curve, we created a group abstraction inspired by the CIRCL library[65]. We subsequently replaced the hardcoded curve with the abstraction for the ease of testing the protocol with different curves. Additionally, we refactored and consolidated the library code in places for additional clarity and fidelity to the Bulletproofs paper[42].

A shortcoming of the library is that it lacks the batching technique which allows for greater efficiency for proving and verifying two (or more) Bulletproofs. Due to time restrictions, we did not implement batching on top of the library either.

Some prime-field NIST elliptic curves (e.g. P-256, P-384) are available in Go as part of the standard library, however, the direct use of the curve operations has been deprecated[66]. The CIRCL library enhances the P-256 and P-384 implementations provided by the Go standard library, and provides optimised operations on P-384[65]. CIRCL also supports the use of the ristretto255 group implemented by the go-ristretto library[67]. To benchmark the full protocol we used the versions of P-256, P-384 and ristretto255 provided by CIRCL, and ING Bank's secp256k1 implementation.

Benchmarks

We ran the initial benchmarks on a MacBook Pro with a 2.42 GHz M2 Pro processor. Both the prover and verifier were part of the same program, all data was held in memory and no data was serialised. We took the average of 1000 runs where we set the proof range to [101, 2000] with $b_x = 16$, and voted for the candidate number 1500. We did not use batching for generating or verifying Bulletproofs. The results are presented in Table 1.

It is clear from the benchmarks that on the prover's side, the proof generation is unlikely to impact the voting experience. While the voting client is written in C++ and not Go, implementation performance in C++ should be comparable. While verification times are lower than proving times, the server must be able to handle and process many concurrent requests. As such, the direct impact is more difficult to assess based on these benchmarks alone. However, it is clear that the curve choice may have a significant impact on the verifier, whereas the impact on the prover is negligible in practice.

To better determine the impact of the protocol on the vote collection server, i.e. the verifier, we ran a benchmark from the standard IVXV benchmark suite with the verification function added in. According to the data obtained from the IVXV developers, the peak concurrent load for IVXV has been 12 votes per second. A typical benchmark target is therefore to process 40 votes per second until the target number of votes has been cast[68]. As such, if the vote collector can keep up with this rate with the additional verification added in, the protocol can be deemed practical.

					Verifier's work (ms)		
λ	κ	b_c	Curve group	Prover's work (ms)	BP	RP	Total
128	112	224	secp256k1	39.10	10.81	20.34	31.15
128	112	224	ristretto255	41.19	11.08	20.88	31.96
128	112	224	P-256	40.30	10.77	20.01	30.78
128	112	224	P-384	171.8	79.06	22.76	101.8
128	128	256	P-384	169.4	77.89	23.17	101.1

Table 1. Prototype benchmarks on a MacBook Pro with a 2.42 GHz M2 Pro processor. All timings are in milliseconds. 'BP' represents the time required to verify both Bulletproofs, while 'RP' represents the time needed to verify Π_{Veq}. λ indicates the security level against confidentiality attacks and is upper bounded by the 128-bit security level of the ElGamal group. κ indicates the soundness level of the scheme.

Category	Duration	Processing rate	DB error count
Reference	06:21:03	38.59	16
P-256	06:20:28	38.66	228
P-384	06:24:47	38.22	1205

Table 2. Load tests with proof verification on the IVXV vote collector with 882366 ballots cast. The duration is in hh:mm:ss format, the processing rate is in ballots per second, and the benchmark target was 40 ballots/s. DB error count represents the number of errors related to the etcd database of the vote collector.

The goal was to run benchmarks on the same servers that are used during the actual elections, which are physical servers in Estonia hosted by the Estonian Information System Authority (RIA). The servers run Ubuntu 22.04, with 8 CPU cores with a base frequency of 2.90 GHz, and 16 GB of RAM. The processor itself is the Intel® Xeon® Gold 6326 Processor with 16 physical cores, however the vote collector is virtualised, with 8 CPU cores available to the virtualised container. For load balancing and redundancy, the vote collector is comprised of three servers. Unfortunately, due to the proximity in time to the 2024 elections, there was little availability for benchmarking on this hardware. As a result, we could only run two benchmarks for which we chose the P-256 and P-384 curves.

For the benchmarks, a proof of vote correctness was serialised into JSON and stored as a Go variable in the server-side code. For each received vote, after performing the habitual verification checks, the server de-serialised the JSON proof and verified it, therefore simulating the actual work of the verifier. In a production setting, the proof and the ballot will need to be read from the ASiC-E signature container containing the encrypted ballot instead. (ASiC-E is a container format defined in ETSI EN 319 162-1, used to bind digital signatures or time assertions to the file objects they apply to. The ASiC-E format is commonly used for digital signatures in Estonia.) However, this overhead is likely to be marginal. Additionally, for the benchmarking, the server re-generated the public parameters before verifying the proof since this allowed easily hooking the prototype to the vote collector. This can easily be avoided in production since the public parameters are known in advance and are the same for every proof.

A total of 882366 ballots were cast, which was the size of the list of eligible voters in the testing environment, and for which a baseline benchmark already existed. The results are presented in Table 2. While the impact of the proof verification on the processing rate of received ballots is negligible, some overhead is still introduced as shown by the increased database error rate.

The vote collector uses etcd[69] as its database for storing ballots. While the common practice is to allocate dedicated resources to etcd[70], the vote collector's etcd shares its resources with the rest of the collector's services, including the verifier service. The higher etcd error count could therefore be explained by the proof verification requiring some of the processing power that was previously used by etcd only. However, it was not possible to determine with certainty whether this was the case, or whether the etcd errors were caused by unrelated state-changes to the system in-between benchmarks.

If an etcd error happens before the received ballot is stored, the collector will retry the operation. However, if the ballot cannot be stored before the configured timeout—typically between 5–10 seconds—is reached, the voting process will fail and the voter will have to restart the voting process. While the number of etcd errors is higher with the proof verifications added in, the error rates remain marginal compared to the total number of votes cast. Furthermore, processing 40 ballots per second is three to eight times more than the expected voting rate during the elections. Even if such a rate is achieved during a peak, it is unlikely to be sustained for a period of several hours and the real impact is therefore likely to be negligible. This remains a speculation however, since a benchmark with a target rate of 10 ballots per second could not be performed due to the unavailability of the infrastructure.

A possible improvement would be to deploy etcd on its own dedicated resources. Not only could this improve or solve the performance problem that caused etcd transactions to fail, it could be a worthwhile architecture improvement in general. While this may require purchasing or leasing additional hardware, the resource requirements for etcd are modest[71] and the cost should therefore not be prohibitive. Three 4-core machines with 16 GB of RAM each should be sufficient for IVXV's etcd needs.

Since the proofs were hardcoded as JSON objects into the collector for the benchmarks, we do not have empirical data available regarding the additional space needed for storing the proofs. However, the theoretical proof size given in Section "Proof size" shows that space is not a limiting factor for practical parameters. Indeed, for the P-384 curve, points can be represented with 49 bytes where the extra byte represents whether the y coordinate is even or odd. Thus, for $b_p = 3072, b_q = 384, b_x = 16$, the individual proof size is $16 \cdot 49B + 9 \cdot 48B + 384B = 1.6kB$, assuming elliptic curve point compression and aggregated Bulletproofs. The overhead of encoding the proof using ASN.1 DER is negligible.

Discussion

While we chose the protocol for proving vote correctness with simplicity in mind, the resulting protocol is not exactly simple. The complexity is twofold: there is the complexity due to the underlying range proof protocol, but also the complexity due to the group switching strategy.

Group switching is seemingly unavoidable as long as the ElGamal ciphertext is generated in the prime-order group of quadratic residues. Because operations in this group are computationally expensive, it is unlikely that range proofs could be efficiently generated in this group, regardless of the technique used. Range proofs must therefore be generated in a group different than the one used for the ElGamal encryption, and a group switching strategy must be used to bind the proof to the ciphertext. Regarding range proofs, it is our opinion that Bulletproofs(+) are the most understandable of the considered approaches.

To avoid the need for group switching, the ElGamal ciphertext could be generated in a computationally more performant group instead. In practice, this means changing the currently used prime-order group of quadratic residues modulo a prime in favour of lifted ElGamal in an elliptic curve group. In lifted elliptic curve ElGamal, the message is represented as an integer which is used as a scalar and is multiplied with the group's base point (the generator). In additive notation, this gives

$$\mathrm{Enc}_H(m; r) = (rG, mG + rH)$$

where G is the group's base point, H is the public key (a point on the curve), m is the message scalar, and r is a random scalar. By only encrypting an integer in the range $[0, 2^{16} - 1]$ it is certain that the message scalar will not overflow the group order.

By picking an elliptic curve compatible with the range proof, the encryption and the range proofs could then be generated in the same group. While this may not replace the need for generating a range proof separately from the ciphertext, proving the equality of discrete logarithms in the same group is simpler than using the cross-group proof, e.g. as in[48]. Since there are less operations to do, this will also improve overall performance on top of simplifying the vote correctness protocol. However, a complete migration to elliptic curves requires re-implementing the current IVXV cryptosystem, thus losing the 'tried and tested' advantage of the current implementation. The potential development effort is therefore much higher than solely using elliptic curves for the vote correctness proof.

One change that will be inevitable when utilizing the Bulletproofs will be the candidate encoding under the encryption. Currently, IVXV uses a data structure including the candidate name, number, precinct number and election event identifier. As Bulletproofs are range proofs, these data structures will need to be remapped into a continuous range of integers. This is technically, of course, doable, but it introduces another layer of indirection, potentially making auditing of the system more complex and thus reducing transparency.

While the benchmark results show that the current prototype should already be practical performance-wise, there are a number of ways to potentially improve the performance. Performance improvements are especially important if the election organiser decides that the current buffer zone between real loads and test loads is not large enough.

The first change would be to implement Bulletproof batching to verify the two range proofs simultaneously. This would replace the $4\lceil\log_2(n)\rceil + 8$ group elements needed for verifying the Bulletproofs separately with $2\lceil\log_2(n) + 1\rceil + 4$ elements, thus shortening the proofs. While the exact time-improvement is hard to estimate, verification time should improve at least by a quarter judging by data from the original paper[42], §6.3. Furthermore, since Bulletproofs+ can be used as a drop-in replacement to Bulletproofs, Bulletproofs+ could be used instead, although this would improve the proof size rather than verification time. While Bulletproofs++ are also a drop-in replacement to Bulletproofs(+), we find the protocol itself to be more complicated than the former two and thus do not recommend this approach. It might also be possible to solely replace the inner product argument with future improvements in the literature without replacing the Bulletproofs(+) themselves; however, this is not guaranteed.

Finally, other than the potential improvements that could be obtained by optimising the Go code, a more radical change would be to use another programming language for the proof verification. In particular, Rust[72] could be used since many production-grade open source proof system implementations have been written in Rust (e.g.[59,73–75]). In turn, this would remove the need for an in-house implementation of the underlying proof system and is likely to also improve performance. Developers could therefore treat the implementation as a black-box, although any third-party library should still be audited.

The proofs of vote correctness must be included in the digitally signed ASiC-E container alongside the encrypted ballot. These proofs must be verified by the vote collector upon receiving each vote, as this enables it to provide immediate feedback to the voter when something is amiss. It remains to determine what should the verification application, the processing application, the key application, and the auditor do.

Since the verification application has the capability to decrypt the vote by using the randomness used during encryption, it can verify the range directly without needing the range proofs. In fact, the verification application currently already verifies whether the ballot contains an eligible candidate identifier.

A necessary change to the verification application, however, is to display the full candidate information as usual. That is, the verification application should map the integer in the ballot back to the full candidate identifier. In principle, it should not be possible to cast a vote with an invalid range proof without the collector refusing to accept the vote. However, the purpose of the verification application is to determine whether both the voting client and the collector behaved correctly. Since the verification application can only be used if a vote has been successfully cast, the verification application should still verify the range proofs. If the range proofs do not validate, the application should alert the voter that the collector misbehaved.

Similarly, the processing application should verify the range proofs to also make sure that the collector did not misbehave. The after-the-fact verification of proofs is the main reason why the proofs should be non-interactive and transferable. If the processing application does not verify that all range proofs are valid, ballots with illegitimate contents could still reach the decryption phase if they slipped by the collector. Since the processing application performs all necessary verifications and is operated by the election organiser, the output of the application can be considered trusted. As such, the key application needs not verify the range proofs, similarly to how the key application does not currently verify whether ballots are well formed, as this is already done by the processor. In fact, it is not possible to verify the vote correctness proofs after ballots have been mixed since the ballot randomness is altered. Therefore, the proofs can only be verified with pre-mixnet ballots. The auditor should verify all the proofs as well.

The processing application and the Android verification application are written in Java, while the iOS verification application is written in Objective-C. This means that Bulletproofs will also have to be implemented in these other languages, which increases the development effort significantly. While this development effort may outweigh the benefits of proof verification by the verification applications, the verification by the processing application is a must. For Java, the implementation by Bünz[76] could be taken as a basis.

Conclusions and future work

Even though all NP-languages are known to have zero-knowledge arguments[77], their performance in specific situations is still a question that has to be addressed case by case. While a proof of ballot correctness may indeed be simple when there are only two choices[16], the solution for larger elections is much less straightforward.

For example, in the case of Estonian elections, there may easily be a few hundred candidates in one precinct. Thus, the solution for the Estonian IVXV Internet voting system must handle this volume efficiency, while supporting hundreds of thousands of voters and dozens of parallel vote casting sessions.

The current paper proposes one possible solution that meets all these criteria. It utilizes Bulletproofs, a technique recently popularized in blockchain applications. One of the challenges with Bulletproofs is that in order to benefit from their efficiency, they should be run on top of elliptic curves, while IVXV is using mod-p ElGamal encryption. Thus, we additionally implemented and benchmarked the group switching mechanism by Chase et al.[49].

The benchmarks confirm that the strategy is performant in practice, even on the current IVXV infrastructure. Indeed, under expected i-voting loads, the server-side ballot processing rate is not impacted, and the proof generation's impact is on the voter side is negligible. By implementing Bulletproofs over the P-384 curve, the individual ballot correctness proof size is approximately 1.6 kB. Had the proofs been used in IVXV for the 2024 European Parliament elections, the cumulative proof size would have been approximately 252 MB. We have also discussed some additional improvements to potentially upgrade the IVXV infrastructure, and further reduce the proof size.

Even though Bulletproofs are efficient, their construction is relatively involved. While this is not a problem mathematically, a solution with a conceptually simpler setup would be preferable from the societal transparency point of view. Also, Bulletproofs are range proofs which presumes changing the current ballot encoding to a continuous range of integers. Again, this is not a problem mathematically, but the current IVXV encoding that includes the candidate ID and a precinct number may provide easier auditability. Thus, full set membership proofs[29,78] could be considered as viable alternatives, even at the cost of reduced performance.

Data availability

The software code created during this research is available at https://github.com/takakv/msc-poc . No other kinds of data have been created nor collected for this research.

Received: 5 March 2025; Accepted: 19 August 2025
Published online: 29 August 2025

References

1. Willemson, J. Bits or paper: Which should get to carry your vote?. *In J. Inf. Secur. Appl.* **38**, 124–131. https://doi.org/10.1016/J.JISA.2017.11.007 (2018).
2. Snetkov, N., Vakarjuk, J. & Willemson, J. Comparing security levels of postal and internet voting. *In Inform. Secur. J. : A Global Perspect.* **34**(4), 265–285. https://doi.org/10.1080/19393555.2024.2410332 (2024).
3. Chevallier-Mames, B., et al. On some incompatible properties of voting schemes. In *Towards Trustworthy Elections, New Directions in Electronic Voting. Lecture Notes in Computer Science* Vol. 6000 (ed. David Chaum et al.) 191–199 (Springer, 2010). https://doi.org/10.1007/978-3-642-12980-3_11.
4. Pankova, A. & Willemson, J. Relations between privacy, verifiability, accountability and coercion-resistance in voting protocols. In *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science.* Vol. 13269 (ed. Giuseppe A. & Daniele V.) 313–333 (Springer, 2022).https://doi.org/10.1007/978-3-031-09234-3_16.
5. Wikström, D., et al. How could snowden attack an election? In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings. Lecture Notes in Computer Science* Vol. 10615 (ed. Robert K. et al.) 280–291 (Springer, 2017).https://doi.org/10.1007/978-3-319-68687-5_17.
6. Müller, J. Breaking and fixing vote privacy of the estonian E-voting protocol IVXV. In *FC 2022 International Workshops, Revised Selected Papers. Lecture Notes in Computer Science* Vol. 13412 (ed. Shin'ichiro Matsuo et al.) 325–334 (Springer, 2022).https://doi.org/10.1007/978-3-031-32415-4_22.
7. Ehin, P. et al. Internet voting in Estonia 2005–2019: Evidence from eleven elections. *In Gov. Inf. Q.* **39**(4), 101718. https://doi.org/10.1016/J.GIQ.2022.101718 (2022).
8. Farzaliyev, V., Krips, K. & Willemson, J. Developing a personal voting machine for the Estonian internet voting system. In *SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021.*. (ed. Chih-Cheng Hung et al.) 1607–1616 (ACM, 2021). https://doi.org/10.1145/3412841.3442034.
9. https://gafgaf.infoaed.ee/posts/kaks-kehtetut/.
10. Heiberg, S., et al. Improving the Verifiability of the Estonian Internet Voting Scheme. In *E-Vote-ID 2016, Proceedings. Lecture Notes in Computer Science* Vol. 10141(ed. Robert Krimmer et al.) 92–107 (Springer, 2016). https://doi.org/10.1007/978-3-319-52240-1_6.
11. Elektroonilise h ä ä letamise ü ldraamistik ja selle kasutamine Eesti riiklikel valimistel. Tech. rep. Version 1.1. Feb. 3, 2023, IVXV-ÜK-1.1.
12. Riigikogu. Riigikogu valimise seadus. Riigi Teataja. (2020). rt: 103012020013.
13. Cohen, J. D. & Fischer, M. J. A robust and verifiable cryptographically secure election scheme. In *26th Annual Symposium on Foundations of Computer Science.* 372–382 (IEEE Computer Society, 1985).https://doi.org/10.1109/SFCS.1985.2.

14. Benaloh, J. C. & Yung, M. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing.* (ed. Joseph Y. Halpern.) 52–62 (ACM, 1986). https://doi.org/10.1145/10590.10595.
15. Benaloh, J. C. Verifiable secret-ballot elections. PhD thesis. Yale University, (1987).
16. Cramer, R., Gennaro, R. & Schoenmakers, B. A Secure and Optimally Efficient Multi-Authority Election Scheme. In *EUROCRYPT '97, Proceedings. Lecture Notes in Computer Science.* Vol. 1233 (ed. Walter Fumy.) 103–118 (Springer, 1997). https://doi.org/10.1007/3-540-69053-0_9.
17. Devillez, H., Pereira, O. & Peters, T. How to Verifiably Encrypt Many Bits for an Election? In *ESORICS 2022, Proceedings, Part II. Lecture Notes in Computer Science.* Vol. 13555 (ed. Vijayalakshmi Atluri et al.) 653–671 (Springer, 2022).https://doi.org/10.1007/978-3-031-17146-8_32.
18. Clarkson, M. R., Chong, S. & Myers, A. C. Civitas: Toward a secure voting system. In *2008 IEEE Symposium on Security and Privacy (SP 2008).* 354–368 (IEEE Computer Society, 2008).https://doi.org/10.1109/SP.2008.32.
19. Sandler, D., Derr, K. & Wallach, D. S. VoteBox: A tamper-evident, verifiable electronic voting system. In *Proceedings of the 17th USENIX Security Symposium.* (ed. by Paul C. van Oorschot.) 349–364 (USENIX Association, 2008)
20. Benaloh, J., et al. STAR-Vote: A secure, transparent, auditable, and reliable voting system. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections.* (USENIX Association, 2013).
21. Cortier, V., Gaudry, P. & Glondu, S. Belenios: A Simple Private and Verifiable Electronic Voting System. In *Foundations of Security, Protocols, and Equational Reasoning - Essays Dedicated to Catherine A. Meadows. Lecture Notes in Computer Science.* Vol. 11565. (ed. by Joshua D. Guttman et al.) 214–238 (Springer, 2019). https://doi.org/10.1007/978-3-030-19052-1_14.
22. Benaloh, J. & Naehrig, M. Election Guard Design Specification. Tech. rep. Version 2.0.0. Microsoft Reserach, Aug. 18, (2023).
23. Martin H. & Kazue S. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT 2000, Proceedings. Lecture Notes in Computer Science.* Vol. 1807 (ed. Bart Preneel.) 539–556 (Springer, 2000). https://doi.org/10.1007/3-540-45539-6_38.
24. Joaquim, Rui. How to prove the validity of a complex ballot encryption to the voter and the public. *In J. Inform. Secur. Appl.* **19**(2), 130–142. https://doi.org/10.1016/J.JISA.2014.04.004 (2014).
25. Brands, S. Rapid demonstration of linear relations connected by Boolean operators. In *EUROCRYPT '97, Proceedings. Lecture Notes in Computer Science.* Vol. 1233 (ed. Walter Fumy.) 318–333 (Springer, 1997). https://doi.org/10.1007/3-540-69053-0_22.
26. Huber, N., et al. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022.* (ed. by Heng Yin et al.) 1443–1457 (ACM, 2022).https://doi.org/10.1145/3548606.3560701.
27. Groth, J. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016, Proceedings, Part II. Lecture Notes in Computer Science.* Vol. 9666 (Ed. Marc Fischlin and Jean-Sébastien Coron.) 305–326 (Springer, 2016). https://doi.org/10.1007/978-3-662-49896-5_11.
28. Lueks, W., Querejeta-Azurmendi, I. & Troncoso, C. VoteAgain: A scalable coercion-resistant voting system. In *29th USENIX Security Symposium, USENIX Security 2020.* (ed. Srdjan Capkun and Franziska Roesner.) 1553–1570 (USENIX Association, 2020).
29. Bayer, S. & Groth, J. Zero-knowledge argument for polynomial evaluation with application to blacklists. In *EUROCRYPT 2013, Proceedings. Lecture Notes in Computer Science.* Vol. 7881 (ed. Thomas Johansson and Phong Q. Nguyen.) 646–663 (Springer, 2013).https://doi.org/10.1007/978-3-642-38348-9_38.
30. Haenni, R., et al. CHVote Protocol Specification. Cryptology ePrint Archive. (2017). iacr: 2017/325.
31. Canetti, R., et al. Universally composable two-party and multi-party secure computation. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing.* (ed. John H. Reif.) 494–503 (ACM, 2002).https://doi.org/10.1145/509907.509980.
32. Benarroch, D. et al. Zero-knowledge proofs for set membership: efficient, succinct, modular. *In Des. Codes Cryptogr.* **91**(11), 3457–3525. https://doi.org/10.1007/S10623-023-01245-1 (2023).
33. Campanelli, M., Hall-Andersen, M. & Kamp, S. H. Curve trees: practical and transparent zero-knowledge accumulators. In *32nd USENIX Security Symposium, USENIX Security 2023.* (ed. by Joseph A. Calandrino and Carmela Troncoso.) 4391–4408 (USENIX Association, 2023).
34. Christ, M., et al. SoK: Zero-Knowledge Range Proofs. Cryptology ePrint Archive. (2024). iacr: 2024/430.
35. Bootle, J., et al. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *ASIACRYPT 2018, Proceedings, Part I. Lecture Notes in Computer Science.* Vol. 11272 (ed. Thomas Peyrin and Steven D. Galbraith.) 595–626 (Springer, 2018). https://doi.org/10.1007/978-3-030-03326-2_20.
36. Deng, C. et al. Cuproof: Range proof with constant size. *In Entropy* **24**(3), 334. https://doi.org/10.3390/E24030334 (2022).
37. Cho, K., Cho, S. & Kim, S. Lightweight signature-based range proof. In *13th International Conference on Information and Communication Technology Convergence, ICTC 2022.* 1862–1865, (IEEE, 2022) https://doi.org/10.1109/ICTC55196.2022.9952590.
38. Wang, N., Chau, S. C-K. & Liu, D. SwiftRange: A short and efficient zero-knowledge range argument for confidential transactions and more. In *IEEE Symposium on Security and Privacy, SP 2024, Proceedings.* 1832–1848 (IEEE, 2024). https://doi.org/10.1109/SP54263.2024.00054.
39. Wang, N. & Chau, S. C-K. Flashproofs: Efficient zero-knowledge arguments of range and polynomial evaluation with transparent setup. In *ASIACRYPT 2022, Proceedings. Lecture Notes in Computer Science.* Vol. 13792 (ed. by Shweta Agrawal and Dongdai Lin.) 219–248 (Springer, 2022). https://doi.org/10.1007/978-3-031-22966-4_8.
40. Deng, C., et al. A survey on range proof and its applications on blockchain. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2019.* 1–8 (IEEE, 2019).https://doi.org/10.1109/CYBERC.2019.00011.
41. Couteau, G., et al. Sharp: Short relaxed range proofs. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022.* (ed. Heng Yin et al.) 609–622 (ACM, 2022). https://doi.org/10.1145/3548606.3560628.
42. Bünz, B., et al. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings.* 315–334 (IEEE Computer Society, 2018). https://doi.org/10.1109/SP.2018.00020.
43. https://crypto.stanford.edu/bulletproofs/.
44. https://github.com/dalek-cryptography/bulletproofs.
45. https://tlu.tarilabs.com/protocols/mimblewimble-mb-bp-utxo.
46. Chung, H. W. et al. Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger. *In: IEEE Access* **10**, 42067–42082. https://doi.org/10.1109/ACCESS.2022.3167806 (2022).
47. Eagen, L., et al. Bulletproofs++: Next Generation Confidential Transactions via Reciprocal Set Membership Arguments. Cryptology ePrint Archive. (2022). iacr: 2022/510.
48. Chaum, D. & Pedersen, T. P. Wallet databases with observers. In *CRYPTO '92, Proceedings. Lecture Notes in Computer Science.* Vol. 740 (ed. Ernest F. Brickell.) 89–105 (Springer, 1992). https://doi.org/10.1007/3-540-48071-4_7.
49. Chase, M., et al. Proofs of discrete logarithm equality across groups. Cryptology ePrint Archive. (2022). iacr: 2022/1593.
50. Abdalla, M. et al. Tightly secure signatures from lossy identification schemes. *In J. Cryptol.* **29**(3), 597–631. https://doi.org/10.1007/S00145-015-9203-7 (2016).
51. Fiat, A. & Shamir, A. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO '86, Proceedings. Lecture Notes in Computer Science.* Vol. 263 (ed. Andrew M. Odlyzko.) 186–194 (Springer, 1986). https://doi.org/10.1007/3-540-47721-7_12.
52. Bellare, M. & Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.* (ed. Dorothy E. Denning et al.) 62–73 (ACM, 1993).https://doi.org/10.1145/168588.168596.

53. Lyubashevsky, V. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT 2009, Proceedings. Lecture Notes in Computer Science.* Vol. 5912 (ed. Mitsuru Matsui.) 598–616 (Springer, 2009). https://doi.org/10.1007/978-3-642-10366-7_35.
54. Bernhard, D., Pereira, O. & Warinschi, B. How not to prove yourself: pitfalls of the fiat-shamir heuristic and applications to helios. In *ASIACRYPT 2012, Proceedings. Lecture Notes in Computer Science.* Vol. 7658 (ed. Xiaoyun Wang and Kazue Sako.) 626–643 (Springer, 2012).https://doi.org/10.1007/978-3-642-34961-4_38.
55. Stinson, Douglas R. Cryptography - theory and practice. Discrete mathematics and its applications series. CRC Press, (1995).
56. Damgård, I. On Σ-protocols. 2010. https://cs.au.dk/%7Eivan/Sigma.pdf (visited on 05/12/2024).
57. State Electoral Office. Statistics. (2023). https://www.valimised.ee/en/archive/statistics (visited on 05/12/2024).
58. Willemson, J. Creating a decryption proof verifier for the Estonian internet voting system. In *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023.* 58:1–58:7 (ACM, 2023).https://doi.org/10.1145/3600160.3605467.
59. de Valence, H., Yun, C. & Andreev, O. Bulletproofs (Dalek Cryptography). https://github.com/dalek-cryptography/bulletproofs (visited on 05/12/2024).
60. https://ristretto.group.
61. Parsovs, A. Estonian electronic identity card and its security challenges. PhD thesis. University of Tartu, (2021).
62. Kraavi, T. ZKRPs of ballot correctness. (2024). https://github.com/takakv/msc-poc (visited on 05/12/2024).
63. Smartmatic-Cybernetica. IVXV Voting Service. Version 1.8.2-RK2023. (2023). https://github.com/valimised/ivxv/tree/master/voting (visited on 05/12/2024).
64. ING Bank. zkrp. (2021). https://pkg.go.dev/github.com/ing-bank/zkrp (visited on 05/12/2024).
65. Faz-Hernández, A. & Kwiatkowski, K. Introducing CIRCL: An Advanced Cryptographic Library. Version 1.3.7. July (2019). https://github.com/cloudflare/circl/ (visited on 05/12/2024).
66. The Go Authors. Package elliptic. https://pkg.go.dev/crypto/elliptic (visited on 05/12/2024).
67. Bas Westerbaan. go-ristretto. https://github.com/bwesterb/go-ristretto (visited on 05/12/2024).
68. Sven Heiberg. Private communication. May (2024).
69. https://etcd.io.
70. https://etcd.io/docs/v3.5/faq/#deployment.
71. https://etcd.io/docs/v3.5/op-guide/hardware/.
72. https://www.rust-lang.org.
73. The Tari Project. Bulletproofs+. https://github.com/tari-project/bulletproofs-plus (visited on 05/12/2024).
74. Polygon Zero Team. plonky2. https://github.com/0xPolygonZero/plonky2 (visited on 05/12/2024).
75. 0xProject. OpenZKP. https://github.com/0xProject/OpenZKP (visited on 05/12/2024).
76. Bünz, B. BulletProofLib. (2017). https://github.com/bbuenz/BulletProofLib (visited on 05/12/2024).
77. Goldreich, O., Micali, S. & Wigderson, A. Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems. *In J. ACM* **38**(3), 691–729. https://doi.org/10.1145/116825.116852 (1991).
78. Groth, J. & Kohlweiss, M. One-out-of-many proofs: Or how to leak a secret and spend a coin. In: *EUROCRYPT 2015, Proceedings, Part II. Lecture Notes in Computer Science.* Vol. 9057 (ed. Elisabeth Oswald and Marc Fischlin.) 253–280 (Springer, 2015). https://doi.org/10.1007/978-3-662-46803-6_9.

Acknowledgements
This paper has been supported by the Estonian Research Council under the grant number PRG2177.

Author contributions
T.K. was the main researcher working on the project, developing the proof-of-concept implementation, benchmarking it and writing the report. J.W. was T.K.'s supervisor, setting the original research problem, suggesting the literature, editing the paper, providing the funding for research, and acting as the contact author.

Declarations

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to J.W.