

# MERGE: Matching Electronic Results with Genuine Evidence

for verifiable voting in person at remote locations

DRAFT Vo.1

Arash Mirzaei, Vanessa Teague  
{ arash.mirzaei, vanessa.teague}@anu.edu.au  
CAC-Vote project\*  
February 7, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Locations . . . . .	5
1.2	Authentication using CAC cards . . . . .	6
1.3	Process . . . . .	6
1.4	Security goals . . . . .	7
1.5	Threat model and trust assumptions . . . . .	7
1.5.1	Attacker model . . . . .	8
1.5.2	Implications . . . . .	9
1.5.3	Counting the number of votes for a trustworthy ballot manifest . . . . .	10
1.6	Notations and building blocks . . . . .	10
<b>2</b>	<b>Basic version: basic-MERGE</b>	<b>12</b>
2.1	Voting process . . . . .	12
2.2	Digital path . . . . .	14

---

\*This project was funded by DARPA “Advanced Paper-based auditing techniques in deployed military settings,” Agreement No.: HRO0112290093

2.3	Paper path . . . . .	17
2.4	RLA . . . . .	18
2.4.1	Determining discrepancies . . . . .	18
2.5	Verification Summary . . . . .	20
2.5.1	Cast-as-intended verification . . . . .	20
2.5.2	BB transcript verification (public) . . . . .	20
2.5.3	Verification at the Local Counting Center . . . . .	21
2.6	Security analysis: Correctness . . . . .	22
2.7	Security analysis: Authentication . . . . .	22
2.8	Security analysis: Privacy . . . . .	22
2.9	Security analysis: Incoercibility . . . . .	23
2.10	Security analysis: End-to-end verifiability . . . . .	24
<b>3</b>	<b>Using MERGE with other RLA styles</b>	<b>24</b>
3.1	VAULT-MERGE: incorporating MERGE in to an RLA that uses VAULT for all ballots . . . . .	24
3.2	Sub-MERGE: ballot-level comparison audits dealing with very small numbers of MERGE ballots . . . . .	25
3.2.1	Using ONEAudit . . . . .	27
3.2.2	Assigning discrepancies arbitrarily . . . . .	27
3.3	Batch-MERGE: batch-level comparison audits . . . . .	27
3.4	Summary and comparison of different MERGE variants . . . . .	29
<b>4</b>	<b>Optimizations for multiple contests</b>	<b>29</b>
<b>5</b>	<b>End-To-End Verifiable version (E2E-MERGE)</b>	<b>30</b>
5.1	Strong adversarial model . . . . .	31
5.2	Setup . . . . .	31
5.3	Voting . . . . .	32
5.4	Digital path . . . . .	35
5.5	Paper path and joining it to the digital path . . . . .	37
5.6	The RLA . . . . .	37
5.7	Verification . . . . .	38
5.7.1	Matching the Receipt with BB data . . . . .	38
5.7.2	Verifying BB data for ballot construction . . . . .	39
5.8	Example of E2E-MERGE . . . . .	39
5.9	Security analysis . . . . .	43
5.9.1	End-to-end verifiability . . . . .	43
5.9.2	Receipt Freeness . . . . .	44

<b>6</b>	<b>Protocol specification</b>	<b>44</b>
6.1	Encryption scheme . . . . .	44
6.2	Hash computation . . . . .	45
6.3	Distributed ElGamal . . . . .	46
6.4	Zero-knowledge proofs . . . . .	48

DRAFT

# 1 Introduction

Slow mail is one of the main barriers to electoral participation for many overseas military personnel. Some jurisdictions send blank ballots by paper mail and require ballot return by the same channel—this can be very slow, often resulting in ballots that are not returned by the deadline. Other jurisdictions allow for paperless voting by Internet, including email or pdf upload—this is very fast, but introduces unacceptable risks to integrity and privacy. One compromise is electronic delivery and paper returns, in which voters download and print a blank ballot, fill it in and mail it back. This has similar integrity and privacy properties to vote-by-mail, and takes only half the time that a fully paper solution would take. Nevertheless, the return mail delivery is often still too slow.

This paper describes the MERGE protocol, which enhances the basic scheme of electronic delivery and paper returns in a way that increases speed without significantly detracting from privacy or integrity. The main idea is to send an untrusted electronic record back quickly, which can be confirmed if the paper ballot arrives in time for the audit, or distrusted if it does not. This extends the time available for the paper ballot to arrive.

Assume a public bulletin board (BB) which is an authenticated broadcast channel with memory. We will extensively use the assumption that people in different locations can rely on seeing the same data on the BB.

- In the polling place, the voter makes both a verifiable paper record, which is mailed, and an untrusted electronic record, which is encrypted and posted to the BB.
- The untrusted electronic record can be incorporated into preliminary results and used as the cast vote record in a Risk Limiting Audit (RLA).
- If the paper ballot arrives in time for the RLA, it is used as the ballot, just like any other RLA.
- If no paper ballot arrives that corresponds to a given electronic record, we use a variation of the phantoms-to-zombies approach of Banuelos and Stark [BS12] to ensure that the risk limit is met even though the electronic record is not trusted.

Two assumptions are unavoidable: first, there must be an accurate count of people who legitimately participated; second, the paper ballot that the voter places into the envelope must accurately reflect their intentions. These need to be supported by human verification processes. The cryptographic protocol is agnostic about whether the paper and electronic records are

produced by scanning a hand marked paper ballot, or printing a paper ballot from a ballot marking device.<sup>1</sup> See [subsection 1.5](#) for a detailed discussion of the trust assumptions.

MERGE is designed to carry a relatively small fraction of votes that are incorporated into a larger electoral process with an RLA. Compared to plain electronic delivery and paper returns, its main advantage is that, rather than needing ballot papers to arrive in time to be scanned for preliminary results, a vote can be safely counted if it arrives before the RLA. In practice, for most jurisdictions, this allows a few more weeks for mail to arrive. Its main disadvantage is that—if a large number of ballot papers do not arrive in time—the consequent large discrepancies may cause an RLA to fall back to a manual recount, when it would not have if those votes had simply been excluded from the beginning.

There is no danger to fairness resulting from the acceptance of delayed ballots after election day, because an electronic commitment to the vote must still be made before the normal close of polls.

This document provides two different versions for the MERGE protocol, where each version is suitable for a specific adversarial model. The first version (basic-MERGE) aims only for *Software Independence* [[Rivo8](#)]<sup>1</sup>—we assume that the attacker can control any electronic components, but not alter ballots in the mail. We also present an end-to-end verifiable version (E2E-MERGE) and explain the implications for the RLA.

All the cryptographic operations are performed using Microsoft’s ElectionGuard library, or some other cryptographic library openly available to the general public.

## 1.1 Locations

The election occurs in two classes of locations, each of which might have several instances.

**Remote Voting Centers** are controlled polling places in remote areas such as overseas military bases.

**Local Counting Centers** are state- or county-based electoral authorities, where the ordinary votes from most citizens are counted. These will also receive paper ballots from Remote Voting Centers and process them.

There are likely to be many Remote Voting Centers and many Local Counting Centers, but we assume only one of each in this discussion for simplicity.

---

<sup>1</sup>Scientific opinion is divided on whether human verification of BMDs is sufficiently accurate to justify the assumption that the BMD printout matches the voter’s intention. See [[ADS20](#)] and [[KBW21](#)] for examples of opposing views. At a minimum, this requires careful design to ensure that voters are motivated and encouraged to verify, and that there is something they can do if a misprint occurs.

## 1.2 Authentication using CAC cards

Voters in remote areas own a smart card called a Common Access Card (CAC) card. It provides a Public Key Infrastructure (PKI) to enable secure authentication and digital signatures. Observers and officials in each Local Counting Center know the public keys of the voters they should include. Digital signatures are used on both the electronic and the paper records.

- Signatures are used to authenticate encrypted votes on the BB.
- Signatures on encrypted votes are printed as a QR code onto a sticker that is applied to the outside of the voting envelope.

Only votes with a valid CAC signature are accepted via either the electronic or paper channels.

This allows officials at the Local Counting Center to notify the voter when a properly-signed ballot from them has arrived. This is expected to be part of the practical implementation. It does not, however, form an important part of the cryptographic protocol because it does not include evidence that the paper ballot matched the electronic one.<sup>2</sup>

## 1.3 Process

The process will consist of the usual risk-limiting audit at the Local Counting Center. We assume that most of the votes are cast locally near the Local Counting Center, with extras incorporated from Remote Voting Centers as needed.

The ballot manifest—available to observers at the Local Counting Center—includes both the ordinary local ballots and all the ballots from Remote Voting Centers. So do the collections of preliminary Cast Vote Records (CVRs). This data is returned to the Local Counting Center electronically via the BB, while the ballot papers are mailed to the Local Counting Center.

The Risk Limiting Audit proceeds exactly as it would if all the votes had been cast locally: observers see the ballot manifest and (a commitment to) preliminary CVRs, then they watch a transparent process for seeding the PRNG that will be used to generate ballot samples, then they check the sequence of sampled ballots. For local ballots, preliminary CVRs are made in the usual way; for MERGE ballots, preliminary CVRs come from the BB in encrypted form. If the sampled ballot is local, they can retrieve it in the usual way and compute its effect on the audit statistics. If the ballot was cast remotely, it will require some extra work to locate the mailed ballot and compare it with its electronic counterpart—this is described below. It is also more likely to be unavailable because it was delayed in the mail—we describe below how to deal conservatively with this situation.

---

<sup>2</sup>The protocol could be extended, with some extra mixing, to notify each voter of whether their specific ballot matched. The current version simply produces a collective tally of how many of the received ballots matched.

The BB is the communication mechanism between the observers at the Local Counting Center and the Remote Voting Center. Assuming that the BB shows them all a consistent transcript, they need only trust each other (and possibly the paper ballot transport, depending on our threat model), not any of the other processes.

## 1.4 Security goals

We desire an election system to satisfy the following properties:

- Correctness: Everyone should have the capability to verify whether the election tally correctly reflects the votes cast.
- Authentication: Only the eligible voters should be able to cast their vote.
- Privacy: The value of a voter's cast ballot should remain undisclosed to any party other than the voter themselves.
- Incoercibility: It should be impossible for a voter to convince anyone of the value of their vote, even if they actively collude with the coercer.<sup>3</sup>
- Paper-based cast-as-intended verification: Each voter can see that their paper vote accurately reflects their intention and refrain from casting it if it does not.<sup>4</sup>
- End-to-end verifiability: Each voter should be able to verify that their vote was cast as intended, collected as cast and tallied as collected. (This applies only to the end-to-end verifiable version—see [section 5](#))
- Risk-limiting: under reasonable assumptions, the RLA should not confirm a wrong election result except with probability at most the risk limit.

## 1.5 Threat model and trust assumptions

Here, we define our trust assumptions as follows:

1. Cryptographic primitives used in our protocols meet the necessary security criteria.
2. Each voter verifies that the paper ballot accurately reflects their intention.

---

<sup>3</sup>*Receipt Freeness* is the more precise term for the property we aim for. Our protocol will reveal who participated, so it will be possible to coerce someone to refrain from participating altogether.

<sup>4</sup>Our cryptographic protocol is agnostic about whether this paper vote is filled in by hand by the voter or printed by a BMD.

3. Each printed signature on the sticker is verified before sending, either by the voter or by some other trustworthy assistant at the Remote Voting Center.
4. The list of identifiers for MERGE ballots that have been validly signed on the BB aligns with the identifier list of voters who attempted to vote using the MERGE system.
5. If the voting system provides the voter audit capability and a voter chooses to audit their ballot, they diligently adhere to the provided audit instructions. At least a portion of them diligently perform cast-as-intended verification and engage in auditing their votes. (Relevant only for the end-to-end verifiable version—See [section 5](#).)
6. The voter remains hidden from others within the confines of the voting booth. The adversary's visibility is restricted to solely observing
  - the BB,
  - the receipt (printed out by the voting machine) that the voter retrieves from the booth (also relevant only to the end-to-end verifiable version),
  - other pieces of evidence (remembered or captured by the voter), which might be susceptible to forgery.
7. A pre-determined fraction of tallying authorities are trusted (relevant for privacy but not integrity),
8. Voters' eligibility is properly enforced at the remote voting center.

### 1.5.1 Attacker model

We define three distinct adversarial models, each characterized by the attacker's capabilities over the paper channel. In all three models, the adversary may control all computers and do any polynomial-time computation. The variations among these models lie in the extent of control the adversary possesses over the paper channel. Specifically:

- *Electronic-only*: The adversary has no control over the paper channel. Every envelope arrives at the Local Counting Center unaltered within a specified timeframe and stuffing new envelopes is impossible
- *Electronic&Drop*: Any envelope, mailed by a legitimate sender, may be selected by the adversary to experience extended delays or even go missing. However, if delivered, the envelope remains unaltered.



**Table 1:** Capabilities of the adversary in different models.

Adv. Model	Electronic Control	Env. Stuffing	Env. Drop	Content View
Electronic-only	✓	-	-	-
Electronic&Stuff	✓	✓	-	-
Electronic&Drop	✓	-	✓	-

- *Electronic&Stuff*: Every envelope arrives at the Local Counting Center unaltered within a specified timeframe. Nonetheless, the adversary can stuff new envelopes on behalf of any legitimate voter.

If ballots are collected by a responsible authority (for example, in a ballot box) and either transported to the Local Counting Center under supervision or audited on the spot with the Remote Voting Center acting as its own Local Counting Center, this corresponds to the Electronic-only model.

Electronic&Stuff model covers the cases where envelopes are mailed individually, but there is a reliable postal system which guarantees on-time mail delivery. Envelope stuffing is not prevented in this model though.

signing the envelope by the voter is a method which can be used to realize the Electronic&Drop model.

The adversary in these models is unable to read the contents of a mailed envelope. Table 1 summarizes these models.

In [section 5](#) we introduce a strong attacker model that assumes the adversary possesses all capabilities of both Electronic&Stuff and Electronic&Drop adversaries.

### 1.5.2 Implications

Our protocol strives to detect any errors that may occur during the voting process. However, complete recovery from these errors cannot be guaranteed. It is important to acknowledge that if any assumptions fail, the errors or attacks in the given adversarial model may become undetectable or it might be impossible to differentiate one type of failure from another. For instance, if the voters do not carefully check that their paper matches their intention, or do not verify that the digital signature they are about to place in the mail is valid (and theirs), then manipulation may be undetectable.

### 1.5.3 Counting the number of votes for a trustworthy ballot manifest

As voters cast their ballots using their CAC cards, the digital signatures associated with each vote on the BB offer a dependable method to authenticate the voter. This robust authentication process ensures that only authorized votes are recorded on the BB, effectively preventing any unauthorized or fraudulent ballots from being included in the tally.

However, it does not guarantee protection against dropping ballots from the electronic record. This could be disastrous because, if the paper records are also delayed or dropped, there is no way to know how many ballots have been dropped, and consequently what the implications for the accuracy of the election result might be.

Assumptions 3 and 4 are arguably enough to prevent ballot dropping in any aforementioned attacker models. However, we will demonstrate that within each attacker model, it is possible to relax these assumptions.

In the Electronic-only model, each paper ballot in the paper manifest reflects the intention of its legitimate voter. The situation is almost the same in the Electronic&Drop model. The envelope cast by any legitimate voter might get lost or experience extended delay. However, if delivered, the envelope remains unaltered and its ballot reflects its sender's intention. Therefore, as a requirement of all RLAs: there needs to be a record of how many ballot papers there are, derived independently of the scanners that are being audited. This assumption needs to be supported by practical work:

- officials at the Remote Voting Center could keep a record of who voted there and verify that it (eventually) matches the set of valid signatures on votes on the BB, or
- every voter would need to verify the inclusion of their vote on the BB.

For the latter case, the sticker on the envelope can be used as the participation evidence.

However, the situation for Electronic&Stuff model is different. Since stuffing new envelopes on behalf of the legitimate voter is possible, we need a way to guarantee that the envelope that is received at the Local Counting Center is accepted by the officials. Otherwise, the adversary could insert an envelope that is eventually accepted in place of the voter's actual envelope, which is rejected due to being identified as invalid. So, in this model, validity of the voting machine's signature on the envelope must be checked either by the voter or a trusted authority.

## 1.6 Notations and building blocks

The selections of voter  $i$ , choosing from  $m$  candidates, are represented as a binary vector  $\mathbf{b}_i$  of length  $m$ , consisting of zeros for unselected options and ones for selections. In cases where it is clear and doesn't cause confusion, the index  $i$  can be omitted for simplicity. The binary

vector  $\mathbf{b}$  corresponds with a *selection option* vector  $\mathbf{o}$  where each vector entry represents a selection option. Then,  $\{\mathbf{o}_j | \mathbf{b}_j = 1\}$  is another way to represent the voter's vote. For example, Alice is another way to represent the vote vector  $\mathbf{b} = [1 \ 0 \ 0]$  in a contest with the following candidate list [Alice Bob Charlie]. To simplify the notations, we often employ the vote vector representation to refer to the voter's selections. However, depending on the context, particularly when the selections are displayed on the screen or printed on the paper, the selection option representation might be a more user-friendly method.

The encryption of the vector  $\mathbf{b}$  is denoted as  $\mathbf{e} = E(\mathbf{b})$ , wherein each entry  $\mathbf{e}_j$  for  $j = \{1, \dots, m\}$  represents the encryption of the corresponding vote entry  $\mathbf{b}_j$ .

**Encryption scheme** Similar to ElectionGuard, encryption of votes in this work is performed using a variant exponential form of the ElGamal cryptosystem. The cryptosystem parameters  $(p, q, g)$  are defined as follows:  $p$  is a prime number equal to  $2kq + 1$ , where  $q$  is also a prime number, and  $g$  is a generator of the order  $q$  subgroup of  $\mathbb{Z}_p^*$ . To generate a public-private key pair, a private key  $s \in \mathbb{Z}_q$  is randomly selected. The corresponding public key,  $K$ , is then computed as  $K = g^s \mod p$  and made public. To encrypt a vote  $b$ , a nonce  $\xi$  is selected randomly such that  $0 \leq \xi < q$ . Then, encryption of  $b$ , denoted by  $E(b)$ , is computed as

$$E(b) = (g^\xi \mod p, K^b \cdot K^\xi \mod p) = (g^\xi \mod p, K^{b+\xi} \mod p).$$

The entity possessing the corresponding secret key  $s$  can decrypt  $(\alpha, \beta)$  as  $\beta/\alpha^s \mod p = K^b \mod p$ .

This encryption scheme is additively homomorphic. Namely,

$$E(b_1, \xi_1) \cdot E(b_2, \xi_2) = E(b_1 + b_2, \xi_1 + \xi_2)$$

We can use the following equation to re-encrypt the ballot  $b$

$$E(b, \xi_1) \cdot E(0, \xi_2) = E(b, \xi_1 + \xi_2)$$

**Mixnet** To maintain the confidentiality of each voter's ballot, a mixnet is used to mix that ballot with those of others. It takes a set of encrypted ballots and outputs a shuffled set of re-encrypted ballots, making it challenging to link any specific encrypted ballot in its output to any of its input encrypted ballots. The mixnet operation on the input encrypted ballot set  $\{E(b_i)\}_i$  is denoted by  $Mix(\{E(b_i)\}_i)$ . We might also apply the mixnet to a batch of encrypted values to process multiple elements together. For instance, it can be applied to the set of encrypted ballot vectors  $\{E(\mathbf{b}_i)\}_i$ . Then, its output is denoted by  $Mix(\{E(\mathbf{b}_i)\}_i)$ .

**Non-interactive zero-knowledge (NIZK) proofs** Similar to ElectionGuard, we use NIZK proofs to demonstrate that an encrypted vote vector  $\mathbf{e}$  is the encryption of a properly formed vote vector  $\mathbf{b}$ . Namely:

- The encryption linked to each option (e.g.  $\mathbf{e}_j$ ) typically represents a valid value and is commonly either an encryption of zero or an encryption of one (i.e.  $\mathbf{b}_j = 0$  or  $\mathbf{b}_j = 1$ ).
- The sum of all encrypted options within each contest falls within a specific range.

NIZK proofs for an encrypted vote vector  $\mathbf{e} = E(\mathbf{b})$  is denoted by  $ZKP_{\text{VALID}}(\mathbf{b})$ .

Similar to ElectionGuard, we also use NIZK proofs to demonstrate that an encrypted value (e.g.  $e = E(b)$ ) is decrypted to a specific value (e.g.  $b$ ), while keeping the encryption nonce and secret decryption keys undisclosed. Such a proof is denoted by  $ZKP_{\text{DECRYPT}}(b, e)$ .

Sometimes a plaintext, with a proof of proper decryption, will be supplied to some participants but not published on the BB. We call this *private decryption*.

Furthermore, we also use NIZK proofs to demonstrate that the Mixnet input  $\{\mathcal{I}_i\}_i$  is correctly and honestly shuffled and re-encrypted to  $\{\mathcal{O}_i\}_i$  without revealing any information about the order and correlation between input and output. Such a proof is denoted by  $ZKP_{\text{MIX}}(\{\mathcal{I}_i\}_i, \{\mathcal{O}_i\}_i)$

**Digital signature** The digital signature on message  $m$  by voter  $i$  is represented as  $sig_i(m)$ .

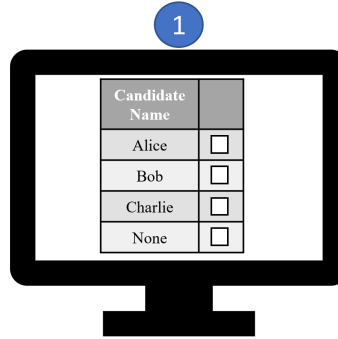
**Serial Number** Each ballot includes a unique serial number  $sn$  which is randomly generated. The role of the serial number is to allow one-to-one matching between the paper ballot and its electronic counterpart.

## 2 Basic version: basic-MERGE

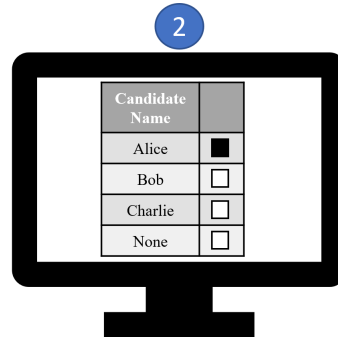
The process is as follows. The example given here uses a BMD to print a paper record, but the protocol would work just as well with a hand-marked paper ballot interpreted by a scanner. In that case, the “voting computer” would be a scanner with the capacity to produce an encrypted record and upload it to the BB. We will use the term “voting computer” to mean either a BMD or a scanner with computation and communication capacity.

### 2.1 Voting process

1. The voting machine displays the options to the voter.



2. The voter interacts with the machine and makes their selection.



Let their vote vector be  $\mathbf{b}$ .

3. The voting machine assigns a unique serial number  $sn_i$  to each ballot paper. The voting machine produces:

- a signed, encrypted electronic record and serial number, with a proof of ballot validity, which goes on the BB:

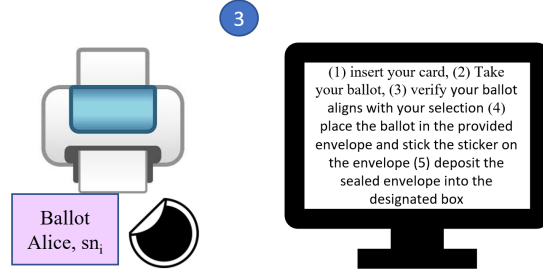
$$BB \leftarrow sig_i(E(sn_i), E(\mathbf{b}_i), ZKP_{VALID}(\mathbf{b}_i), Domain\_separator\_BB),$$

- a plain paper record of the ballot  $(sn_i, \mathbf{b}_i)$ , which is placed in a ballot box or in an envelope for mailing, and
- a sticker that includes an address, a traditional mail tracking number, a space for a pen-and-ink signature (if required by regulation—not relevant to our cryptographic protocol) and a digital signature

$$sig_{Voting\ Computer} sig_i(E(sn_i), E(\mathbf{b}_i), ZKP_{VALID}(\mathbf{b}_i), Domain\_separator\_mail).$$

The voter must verify that her plaintext printout matches her intention, then stick the sticker on her envelope, then mail it. Someone also needs to verify that the digital signature on the sticker is valid, and that a matching encrypted vote eventually arrives on the BB, but this does not impact privacy and may be done by voters, officials or bystanders.

The purpose of the digital signatures is to prevent ballot stuffing—only ballots (either electronic or paper) with a valid accompanying digital signature will be accepted. The Voting computer’s digital signature needs to be explicit on the mail sticker, but is implicit on the BB because that channel is already authenticated. The purpose of the domain separators is to ensure that an attacker cannot see a signature on the BB and copy it onto a sticker in the mail.



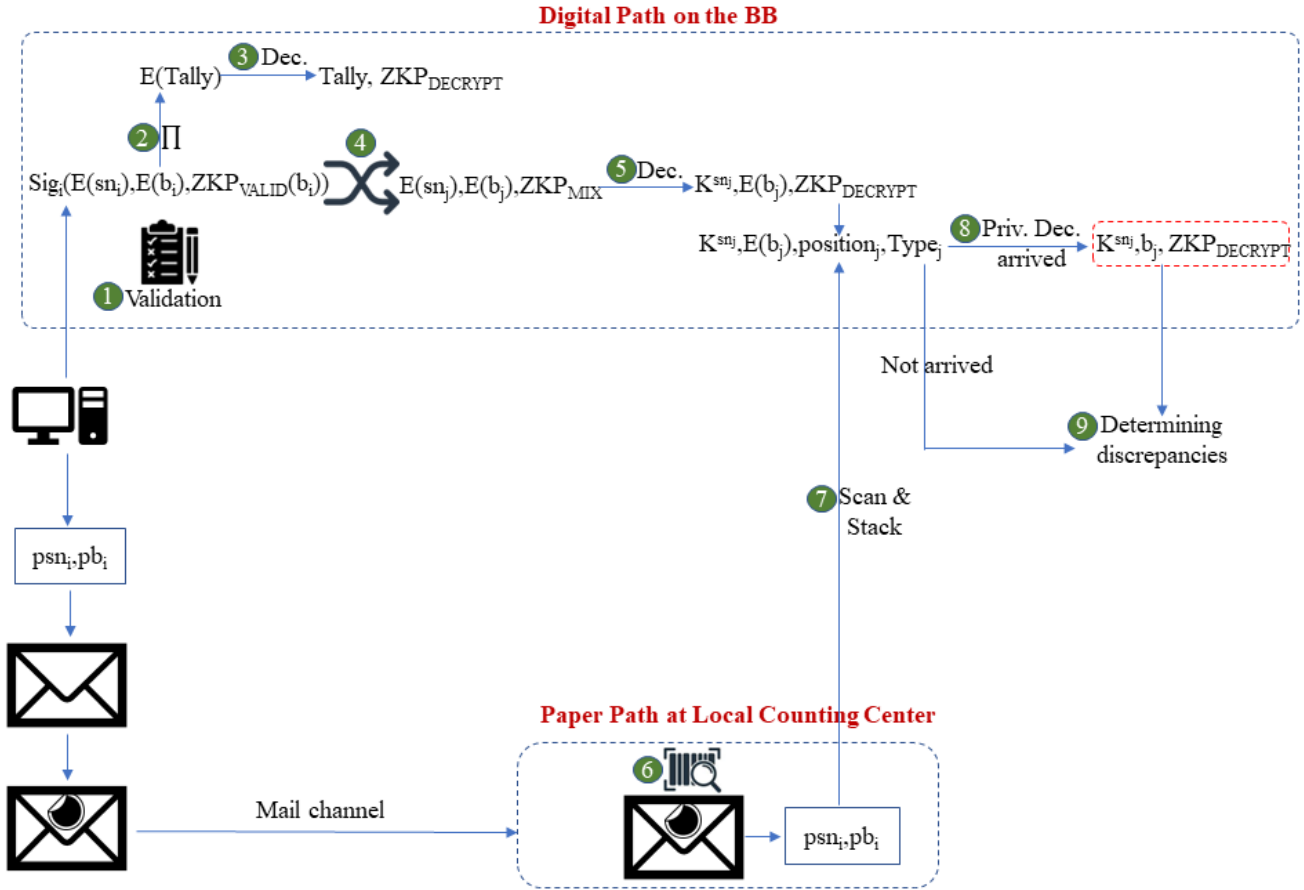
This entire flow could work just as well with a more standard ElectionGuard setup, with a hand-marked paper ballot input into a scanner. The serial numbers could either be generated by the scanner, or printed in advance.

After the voting period ends, electronic records undergo a series of processing steps. Additionally, paper ballots are sent to the Local Counting Center for auditing purposes. A diagram illustrating the whole process is in Figure 1. Details of all processing steps will be provided in the following sections.

## 2.2 Digital path

We explain digital processing of electronic ballots for one Local Counting Center’s votes. Every other Local Counting Center’s data is processed similarly—they do not interact because their results need to be delivered to the appropriate Local Counting Center, even for statewide contests. At the conclusion of the voting process, the following digital record containing the information of all  $n$  voters becomes publicly available on the BB, where the record with index  $i$  denotes the ballot from voter  $i$ , where  $i \in \{1, \dots, n\}$ .

$$BB \leftarrow \mathcal{D}_1 := \{sig_i(E(sn_i), E(\mathbf{b}_i), ZKP_{VALID}(\mathbf{b}_i), Domain\_separator\_BB)) : i \in \{1, \dots, n\}\}$$



**Figure 1:** The whole process for basic-MERGE based on ballot comparison. Each Local Counting Center's votes are dealt with separately—the diagram shows the process for only one Local Counting Center.

Subsequently, this digital record undergoes a series of processing steps as follows:

1. The validity of the signature and ZKP for each electronic ballot is verified.
2. The encryption of the total tally is calculated and published on the BB

$$BB \leftarrow E(\mathbf{Tally}) := \prod_i E(\mathbf{b}_i).$$

3.  $E(\mathbf{Tally})$  is decrypted and published on the BB along with the proof of the correct decryption.

$$BB \leftarrow E(\mathbf{Tally}), \mathbf{Tally}, ZKP_{\text{DECRYPT}}(\mathbf{Tally}, E(\mathbf{Tally}))$$

(In some jurisdictions, this value may be sensitive because of a small set of MERGE voters. See [section 3](#) for variations that allow public verifiability without publication of the separate tally.)

4. Let  $\mathcal{V} \subseteq \{1, \dots, n\}$  indicate the set of voters whose corresponding envelope with the valid signature is received at the Local Counting Center. The pairs  $\mathcal{D}_2 := \{(E(sn_i), E(\mathbf{b}_i)) : i \in \mathcal{V}\}$  are extracted from  $\mathcal{D}_1$  and then are mixed. The pairs  $\mathcal{D}'_2 := \{(E(sn_i), E(\mathbf{b}_i)) : i \in \{1, \dots, n\} \setminus \mathcal{V}\}$  are also extracted from  $\mathcal{D}_1$  but do not go through the mixing process. Ballots corresponding to  $\mathcal{D}'_2$  are categorized as “not arrived” stack. So, at the end of this step, we have the following data on the BB:

$$BB \leftarrow \mathcal{D}_3 := \{(E(sn_j), E(\mathbf{b}_j)) : j = \pi(i), i \in \mathcal{V}, ZKP_{\text{MIX}}(\mathcal{D}_2, \mathcal{D}_3)\}$$

where  $\pi$  reflects the secret permutation from  $\mathcal{V}$  to  $\mathcal{V}$  applied in the Mixnet operation.

5. Serial numbers for the mixed, arrived stack are decrypted and then the proof of the correct decryption is published. Therefore, at the end of this step, we have the following data on the BB.

$$BB \leftarrow \mathcal{D}_4 := \{(K^{sn_j}, E(\mathbf{b}_j), ZKP_{\text{DECRYPT}}(K^{sn_j}, E(sn_j))) : j = \pi(i), i \in \mathcal{V}\}$$

It is also checked that there are no duplicate serial numbers. Any occurrence of duplicate serial numbers indicates significant problems with a voting computer—if there are any, the process should stop.



## 2.3 Paper path

At the conclusion of the voting process, the ballot papers are conveyed to the Local Counting Center, either individually in separate envelopes, or collectively in some organized way. Subsequently, this paper record undergoes a series of processing steps as follows:

6. If the jurisdiction has an existing process of scanning traditional handwritten signatures, this would also apply at this point. Envelopes with invalid signatures must be set aside and handled properly.

Each incoming envelope's sticker is scanned and its corresponding digital signatures (voter and voting computer) are verified. It is necessary to check that the data being signed is the same as that on the BB for this voter, the domain separator is the right one, BMD's signature is valid and it is signed with an eligible voter's CAC card. If any of these verifications fails, the process continues but the envelopes are set aside in a stack called "Rejected Envelopes" for further investigations. If it is signed with an eligible voter's CAC card but another validly signed envelope has already been received from the same voter, it is set aside in a stack called the "Eligibility Problem" stack.

For all validly signed envelopes (excluding the ones inside the "Eligibility Problem" stack), the envelope contents are removed and physically shuffled, similar to standard postal voting procedures.

7. Each incoming ballot's serial number  $psn$  is scanned. For each serial number,  $K^{psn}$  is computed and it is checked if there is a digital ballot with a matching  $K^{sn}$  on the BB.<sup>5</sup> Then, the location of the paper ballot is appended to the corresponding digital ballot on the BB.<sup>6</sup> Each digital ballot is accompanied by a parameter called  $Type$  where  $Type = \text{"not matching"}$ ,  $Type = \text{"one-to-one"}$  and  $Type = \text{"duplicated"}$  respectively indicate if there exist zero, one or multiple paper ballots with the matching serial number. Therefore, we have the following data on the BB:

$$BB \leftarrow \mathcal{D}_5 := \{(K^{sn_j}, E(\mathbf{b}_j), \{position_{j'}\}_{j'}, Type_j) : j = \pi(i), j' = \pi'(i), i \in \mathcal{V}\}$$

The mapping  $\pi'$  represents the fact that paper ballots have been physically shuffled and  $position_{j'}$  denotes the location of the associated paper ballots in paper storage. For digital

<sup>5</sup>Because of the way ElectionGuard encryption works, the number that naturally drops out after decryption is  $K^{sn}$ , not  $sn$ . Because we care only about exact matches, it does not matter whether we work in the exponential or plain form. We do not assume that  $K^{sn}$  hides  $sn$ , and do not need to hide the values of  $sn$ , because these are not publicly associated with the voter.

<sup>6</sup>This is its physical location in paper ballot storage, as in a ballot manifest. For example, "23rd ballot in batch 7, cabinet 12."

ballots with multiple corresponding paper ballots (i.e., ballots with  $Type = \text{“duplicated”}$ ), the positions of all matching ballots are recorded on the BB. For digital ballots with  $Type = \text{“not matching”}$ , the parameter  $position$  is null.

Any paper ballot with no matching  $K^{sn}$  on the BB is set aside. Call this the “problem” stack. These need to be dealt with outside the cryptographic protocol.

## 2.4 RLA

8. At the Local Counting Center the usual local Cast Vote Records (CVRs) are joined with the MERGE CVRs for auditing purposes, and the ballot manifest is updated to include both kinds of electronic records. When a local ballot is sampled, it is retrieved and dealt with as usual. When a MERGE ballot is sampled, if its type is either “one-to-one” or “duplicated”, its encrypted vote  $E(\mathbf{b})$  is privately decrypted to  $\mathbf{b}$  and, alongside the proof of correct decryption, is supplied to all auditors and observers in the Local Counting Center. Therefore, they have access to the following data for the selected ballot:

**Observers:**  $(K^{sn_j}, E(\mathbf{b}_j), \mathbf{b}_j, ZKP_{\text{DECRYPT}}(\mathbf{b}_j, E(\mathbf{b}_j)), \{position_{j'}\}_{j'}, Type_j \neq \text{“not matching”})$

Then, the discrepancy is determined according to the guidelines outlined in Section 2.4.1 and then RLA statistics are updated accordingly, in exactly the same way for remote ballots as they would be for local ones. The discrepancies can be made public in whatever way the Local Counting Center usually publishes its RLA calculations, or they can be appended to the relevant record on the BB.

If the selected ballot’s type is “not matching”, its discrepancy can be determined according to the process described in Section 2.4.1.

If the index for selected ballot is not a member of the set  $\mathcal{V}$ , its discrepancy can be determined according to the process described in Section 2.4.1.

Alternatively, rather than private decryption of the ballot that is selected for RLA, we can privately decrypt all digital ballots before initiating the RLA process.

Local officials make local decisions about whether to accept the result or escalate to a larger sample, according to whatever RLA calculations they usually conduct.

### 2.4.1 Determining discrepancies

Assigning discrepancies to digital ballots is primarily determined by the stack to which the ballot belongs.

9.1. Let's start with the ballots with  $Type = \text{"one-to-one"}$ . Assume a digital ballot with the following data is selected for RLA:

$$(K^{sn_j}, \mathbf{b}_j, position_{j'}, Type_j = \text{"one-to-one"})$$

Then, we follow the below guidelines:

- Identify the paper ballot situated at position  $position_{j'}$ . Given that its serial number and content are represented by  $psn$  and  $\mathbf{pb}$  respectively, verify whether  $K^{psn}$  matches  $K^{sn_j}$ . If not, output "serial number error" and terminate.
- Compare  $\mathbf{pb}$  to  $\mathbf{b}_j$  and record the discrepancy, exactly like any othe RLA.

Any occurrences of the "serial number error" indicate a significant problem with the scanning device. Therefore, encountering either of these errors necessitates restarting the entire process from step 7 onward using another scanning device.

9.2. Let's continue with ballots with  $Type = \text{"duplicated"}$ . Assume a digital ballot with the following data is selected for RLA:

$$(K^{sn_j}, \mathbf{b}_j, \{position_{j'}\}_{j'}, Type_j = \text{"duplicated"})$$

Then, the 2-step process described in step 9-1 is performed for all  $j'$  values and the final discrepancy is set to the *minimum* discrepancy among different  $j'$  values for the digital ballot.

Any occurrences of the "serial number error" indicate a significant problem with the scanning device.

9.3. The following options are available for determining discrepancies for ballots with  $Type = \text{"not matching"}$ :

worst-given-tally All encrypted ballots with  $Type = \text{"not matching"}$  are homomorphically added and the tally privately decrypted for the auditors and observers in the Local Counting Center.<sup>7</sup> Let this sub-tally be denoted by  $\mathbf{Tally}_{n-a}$ . The corresponding value  $\mathbf{Tally}'_{n-a}$  is set to the worst possible tally for  $\#\{j | Type_j = \text{"not matching"}\}$  number of votes. The total discrepancy  $\mathbf{dis}_{n-a}$  is computed as  $\mathbf{dis}_{n-a} := \mathbf{Tally}_{n-a} - \mathbf{Tally}'_{n-a}$  and arbitrarily assigned to non-accept ballots.

<sup>7</sup>This could also be published on the BB if the anonymity set is sufficiently large. Not clear what the advantage would be.

worst-given-ballot The encrypted vote  $E(\mathbf{b}_j)$  is privately decrypted to  $\mathbf{b}_j$  and alongside the proof of correct decryption is supplied to all auditors and observers in the Local Voting Center. Then, the largest discrepancy based on the value of the  $\mathbf{b}_j$  is considered.

max-possible Regardless of the value of the given ballot, its discrepancy is set to the maximum value. This will usually be a +2 overstatement for plurality voting, but may be something different for other social choice functions.

9.4. Envelopes in the “Eligibility Problem” stack require further investigations as outlined later. Furthermore, based on the paper channel security model, validly signed envelopes in the “Eligibility Problem” might also impact the RLA process as follows:

- Electronic-only model: Any envelope in the “Eligibility Problem” stack demonstrates a major problem that halts the process.
- Electronic&Stuff and Electronic&Drop models: For any validly signed envelope in the “Eligibility Problem” with a distinct voter’s identifier, a maximum discrepancy (typically +2) is added to their county votes.

9.5. If the ballot that is selected for the RLA is in “not arrived” stack, its discrepancy is set to +2.

## 2.5 Verification Summary

There are three separate verification stages, which can be verified by different people: cast-as-intended verification, which is mostly done by the voter, universal (BB) verification, which can be done by any member of the public, verification by observers at the Local Counting Center that the data on the BB matches the ballot papers and the data included in the tally and the RLA matches the BB.

Each of these is detailed below.

Note that this version does *not* allow for individual recorded-as-cast verification: individual voters do not get evidence that their electronic record matches their intention. This evidence—or rather, evidence about whether the discrepancy is large enough to alter the result—is provided collectively through the matching and subsequent audit processes.

### 2.5.1 Cast-as-intended verification

### 2.5.2 BB transcript verification (public)

1. For each vote on the BB:

### By the voter

Verify that the ballot paper matches the intended vote.

### By the voter or anyone else at the Remote Voting Center

Verify the digital signature on the sticker stuck onto the envelope.

Verify that a properly-signed vote from this voter is (eventually) on the BB.

- (a) Verify digital signatures by voter and voting computer
  - (b) Verify ZKP of proper ballot construction
2. Verify mixing ZKP
  3. Verify decryption of all serial numbers  $sn_i$ .
  4. Verify that each  $sn_i$  is unique.
  5. Verify aggregation of  $E(\mathbf{Tally})$ .
  6. Verify the decryption  $\mathbf{Tally}$  of  $E(\mathbf{Tally})$ .

### 2.5.3 Verification at the Local Counting Center

1. Verify digital signatures on the stickers, and verify that they're signing the same data with appropriate domain separator.
2. Verify that the *Eligibility Problem Stack* includes all (and only) duplicate envelopes from the same voter.
3. Verify that the *Rejected Envelopes Stack* includes all (and only) envelopes with the invalid signatures.
4. Verify that all ballots on the BB are correctly categorized into "not matching", "one-to-one" or "duplicated".
5. Verify that *Problem Stack* includes all (and only) ballots with no matching  $sn_i$  on the BB.
6. Verify that  $\mathbf{Tally}$  is properly added to the local CVRs or tallies.
7. Verify  $ZKP_{\text{DECRYPT}}(,)$  for each ballot that is privately decrypted during the RLA process.

8. Verify aggregation of  $E(\mathbf{Tally}_{n-a})$  and its decryption.
9. Verify that discrepancies are properly determined and then correctly incorporated into the RLA.
10. Verify there are no “serial number error”s.

## 2.6 Security analysis: Correctness

TBD.

## 2.7 Security analysis: Authentication

Only the eligible voters are able to cast their vote.

## 2.8 Security analysis: Privacy

In this section, we delve into the privacy aspects of our system, focusing on safeguarding the confidentiality of voter’s intention. Privacy, in this context, involves preventing unauthorized access to information regarding individual voters’ choices. The general public, i.e, anyone who views the BB, knows the list of identifiers and serial numbers for the participating voters. Due to the mixnet properties, they cannot link these two lists though. Also, the vote contents are encrypted on the BB, ensuring that no information about the voter’s intention is revealed. Subsequently, after the mixnet, the decrypted vote contents are made accessible to the RLA auditor and observers. However, these decrypted ballots cannot be linked to the voters’ identities, making each voter indistinguishable from others whose ballots contain the same voting options. Furthermore, the anonymity is maintained throughout the process at the Local Counting Center, as there are no unique markings on the paper ballot that would identify the voter. While the digital signature (and optionally the handwritten signature) on the envelope serves to identify the voter, the serial number and the ballot’s vote contents remain hidden from postal workers. The BMD machine possesses complete information, including the voter’s identifier, serial number, and vote contents. Table 2 provides an overview of the access levels granted to various stakeholders, aiding them in comprehending and analyzing individual voter choices.

According to this table, exclusive access to both the voter’s identifier and their votes is limited to the BMD machine alone; no other single entity possesses this combination of information for any given voter.

**Table 2:** Data accessible to various entities across different stages of the voting process.

Entity	ID	SN	ID&SN	Vote	Tally
General public	✓	✓	×	×	✓ <sup>†</sup>
BMD machine	✓	✓	✓	✓	✓
RLA observers	×	✓	×	✓	✓ <sup>†</sup>
Postal worker	✓	×	×	×	✓ <sup>†</sup>

<sup>†</sup>: visible on the BB

Section 3.3 elaborates on the removal of serial numbers from our protocol, particularly in scenarios involving batch auditing. This ensures the preservation of voter privacy, even in the event of collusion between a coercer and auditors or observers at Local Counting Center.

However, there are still some privacy issues worth considering.

- **Small anonymity sets** may reveal individuals' preferences, for example if there are only a very small number of remote votes sent to one Local Counting Center, and they all make the same choices. Section 3.2 presents a solution to this problem. Our proposed solution involves merging our protocol's ballots with those from non-basic-MERGE voters, thereby enlarging the anonymity set.
- **Italian attacks** may allow for coercion, if a person has many options on one ballot. However, our protocol resists this attack due to the fact that ballots on the BB are encrypted and contest-wise homomorphic tallying is performed on them. RLA auditors do have access to individual paper ballots, thereby presenting a coercion opportunity that appears inherent in any RLA process.
- **Group privacy** of the whole set of remote voters. This may cause concern even if the group is large, for example if the collective choices are strongly skewed relative to the rest of the population.

## 2.9 Security analysis: Incoercibility

We assume the presence of a coercer outside the voting booth who is already aware of the target voter's identity and requests their serial number. Given that (1) serial numbers are ultimately disclosed on the BB and (2) serial numbers are sufficiently long, the voter is unable to provide false information about their serial number. Nevertheless, the coercer cannot rely on the voter's honesty regarding their cast vote. In other words, the voter cannot convince the coercer that the encrypted value  $E(\mathbf{b}_i)$  printed on the the sticker or published on the BB before

**Table 3:** Data accessible to the coercer and the RLA observers.

Entity	ID	SN	ID&SN	Vote	Tally
Coercer	✓	✓	✓	×	✓ <sup>†</sup>
RLA observers	×	✓	×	✓	✓ <sup>†</sup>

<sup>†</sup>: visible on the BB

the mixnet or the  $E(\mathbf{b}_i)$  in the Remote committed CVR (i.e.  $(sn_i, E(\mathbf{b}_i))$  after the mixnet) is decrypted to the  $\mathbf{b}_i$  that is claimed by the voter as her vote. However, as Table 3 shows, collusion between a coercer and auditors or observers at the Local Counting Center could enable the use of serial number data to connect voters' identities with their votes.

## 2.10 Security analysis: End-to-end verifiability

Not achieved. The electronic channel has no cast-as-intended verifiability, while the paper channel has no individual recorded-as-cast verifiability. The alignment between the electronic and paper channels relies on observers in the Local Counting Center to check the RLA.

## 3 Using MERGE with other RLA styles

The description above assumes a Local Counting Center using ballot-level comparison RLAs for all ballots, including those from MERGE. However, there are several other variants that work well with other RLA styles and may be more appropriate in some scenarios, particularly when the number of MERGE ballots is small.

### 3.1 VAULT-MERGE: incorporating MERGE in to an RLA that uses VAULT for all ballots

The VAULT RLA scheme [BST19] protects against pattern-based coercion attacks (often called “Italian attacks”) by hiding individual ballots unless they are selected for RLA. MERGE ballots can easily be incorporated into a VAULT RLA. This protects against both the problem of small sub-tallies for MERGE ballots and also the individual pattern-based coercion attacks on all ballots (at least, all ballots that are not selected for audit).

There are two main steps. In the first step, all the votes are tabulated on a VAULT BB. This step is simple: the MERGE ballots that are *output* by the mix are in exactly the right form for



VAULT. The ordinary ballots from the Local Counting Center can simply be appended—at this step, it is perfectly acceptable for the different types of votes to be distinguishable because they are all encrypted. The tallies are then computed over the whole set of votes. This obviates the need to publish the sub-tally of MERGE ballots in [item 3](#) of the MERGE digital path.

In the second step, the RLA is conducted using the ballot-level comparison method, with samples taken at random from the VAULT-tabulated ballots. Discrepancies are calculated as follows:

- if the ciphertext is a MERGE ballot, it should be treated as described in [subsection 2.4.1](#), using a ballot paper of matching serial number if possible, and a worst-case assumption if there is no ballot paper,
- if the ciphertext is an ordinary ballot, the paper ballot should be found and the discrepancy calculated as in a standard VAULT RLA.

The main restriction, compared with standard VAULT, is that for the MERGE ballots, the ciphertext must be decrypted rather than opened, because the authorities do not know the random factors used to generate it. The ordinary votes need to use the same encryption scheme as the MERGE ones, so that they can be aggregated together—they may be either decrypted like the MERGE ballots or opened (from the randomness used to generate them) like standard VAULT ballots.

This method allows for a very efficient audit (ballot-level comparison for all ballots) with good privacy properties, even if the MERGE ballots are only a small set.

### **3.2 Sub-MERGE: ballot-level comparison audits dealing with very small numbers of MERGE ballots**

If a Local Counting Center has only a very small number of MERGE ballots, it may not be acceptable to publish their tally in [item 3](#). By exposing the tallies for the subset of MERGE ballots, the protocol may reveal significant information about individual votes. They may be unanimous or near-unanimous, or there may be some contests in which only one MERGE voter was eligible.

When an attacker has direct access to the arriving mail ballot, we do not have a solution to this issue. However, if we consider a remote attacker who is looking at the sub-tallies on the BB, the problem can be addressed by including the MERGE tally into a larger tally of ballots. The best way to do this is to use VAULT (See [subsection 3.1](#)). This section describes a lightweight alternative that may be easier to implement for Local Counting Centers that do not already use VAULT.

**Table 4:** Data accessible to various entities in Sub-MERGE.

Entity	ID	SN	ID&SN	Vote	MERGE Tally
General public	✓	✓	×	×	×
BMD machine	✓	✓	✓	✓	✓
Coercer	✓	✓	✓	×	×
RLA observers	×	✓	×	✓	×
Postal worker	✓	×	×	×	×

The main idea is to make a VAULT-like batch that includes all the MERGE ballots and enough ordinary ballots to constitute a reasonable anonymity set. We tally the batch both electronically and on paper, then use one of several possible methods for incorporating the batch into an otherwise ballot-level comparison audit. Table 4 presents an overview of the access levels granted to various stakeholders in Sub-MERGE. Clearly, sub-tallies from MERGE ballots are not visible on the BB. This is explicitly depicted in the last column of the table.

Let’s provide more details regarding this method. Call the (small) set of MERGE ballots on the BB  $M$ . When  $|M|$  is small, mixing does not have much benefit. We could use the ballots output from the mix, but it is just as valid to use the signed, encrypted votes from *before* the mixing step. Either way, this gives us the encryption of the total tally, either the one published in item 2 of the digital path, or an equal one derived from the ballots output by the mix.

1. Gather a collection of local ballots  $L$  from those available in the Local Counting Center. They may have been cast in person, or they may be other (non-MERGE) absentee ballots.  $L$  should be chosen so that, when combined with the MERGE ballots, the anonymity set is large enough for the tallies to be published. Probably  $|L \cup M| \approx 30$ .

We assume that ballots in  $L$  are already disassociated from the voter’s name.

2. Encrypt the ballots in  $L$  using the encryption scheme from subsection 1.6, including validity proofs. Post them on the BB.
3. Use the homomorphic property to compute the combined aggregate of  $L \cup M$ . Decrypt it and publish the ciphertext with proof of correct decryption on the BB. This gives us, on the BB:

$$E(\mathbf{Tally}_{L \cup M}), \mathbf{Tally}_{L \cup M}, ZKP_{\text{DECRYPT}}(\mathbf{Tally}_{L \cup M}, E(\mathbf{Tally}_{L \cup M}))$$

4. Manually tally the combined batch comprising  $L$  and the MERGE ballots. Call the resulting tally  $\mathbf{Tally}_{\text{MANUAL}}$ .
5. Compute the overall discrepancy  $D$  between the electronic and manual tallies.

We now have the discrepancy between a plaintext electronic tally and a manual tally of paper ballots, each comprising both MERGE and ordinary ballots. Ideally, these would correspond perfectly. However, there may be fewer paper MERGE paper ballots than electronic ones, since mail may be dropped or delayed, or other discrepancies.

The resulting batch discrepancy can be incorporated in to a ballot-level comparison RLA, either by adopting the ONEAudit approach, or by assigning worst-case discrepancies like the strategy described in [subsection 2.4](#).

If a MERGE ballot is selected for audit, but no corresponding paper ballot has arrived, it is important to ensure the right worst-case assumption in the case that overstatements of +2 or more are possible. We count the total number of MERGE electronic ballots, and subtract the number of MERGE ballots that have arrived, then explicitly add a “not arrived” zombie ballot to the collection of ballot papers. For the RLA, we then sample randomly from the collection of paper ballots, which includes ordinary paper ballots from the Local Counting Center, MERGE paper ballots that have arrived, and zombies representing those that have not arrived.

### 3.2.1 Using ONEAudit

A high-level description of ONEAudit is given below—for details see [[Sta23](#)].

- Create an *Overstatement Net Equivalent* CVR for each electronic record in  $L \cup M$ . This effectively spreads any discrepancy evenly across  $L \cup M$ .<sup>8</sup>
- If a ballot in  $L \cup M$  is chosen for audit, use the discrepancy between its paper ballot and its *Overstatement Net Equivalent* CVR.

### 3.2.2 Assigning discrepancies arbitrarily

Another strategy is to assign the discrepancies arbitrarily, in advance of the audit, ensuring that the total overstatements are correct, and then use the assigned discrepancy if that ballot is selected for audit. This strategy is similar to that described in [subsection 2.4](#).

## 3.3 Batch-MERGE: batch-level comparison audits

Everything described above would work just as well for batch-level comparison audits. It would make sense to define the batches according to some physical convenience function (for exam-

---

<sup>8</sup>The exact definition of a ONEAudit CVR depends on the audit method being used, but for a two-candidate contest, take the winner’s tally minus the loser’s and divide by the total number of ballots. This becomes a virtual CVR against which each sampled paper ballot is compared..

**Table 5:** Data accessible to various entities in Batch-MERGE.

Entity	ID	Batch No.	ID&Batch No.	Vote	Tally
General public	✓	✓	✓	×	✓ <sup>†</sup>
BMD machine	✓	✓	✓	✓	✓
Coercer	✓	✓	✓	×	✓ <sup>†</sup>
RLA observers	×	✓	×	✓	✓ <sup>†</sup>
Postal worker	✓	✓	✓	×	✓ <sup>†</sup>

<sup>†</sup>: visible on the BB

ple, all the MERGE ballots that arrived on a certain date) because the corresponding electronic records should be easy to identify and need not be together on the BB.

This would make sense if the Local Counting Center already used batch-level comparison audits for their RLAs. If so, MERGE could easily be (slightly) modified to fit that auditing strategy.

It may even be possible to determine in advance which paper ballots will be in which batch—for example, if so few MERGE ballots are sent to one Local Counting Center that they are all treated as a single batch. In these cases, it may be possible to omit the serial numbers entirely, relying instead on the identity of the Local Counting Center to form the batch. The same sort of worst-case discrepancy calculation could be performed at a batch level.

Extending this idea further, if it was known in advance how many batches the MERGE ballots would comprise for a given Local Counting Center, it would not be necessary to put individual serial numbers on each ballot. It would suffice to label them with their intended batch, which would then need to be physically collated at the Local Counting Center. This would significantly improve privacy against an attacker who views the vote receiving process.

Note that it does not matter whether the attacker knows, or can change, which ballots are contained in the same batch—this worst-case attack assumption is already part of the attacker model of batch RLAs. As long as the attacker does not know which batches will be chosen, the risk limit is met.

Table 5 summarizes the access levels granted to various stakeholders in Batch-MERGE.

In the context of this table, we operate under the assumption that the batch number assigned to each participating voter is publicly disclosed on the BB. As indicated in the table, apart from the BMD machine, exclusive access to each ballot’s vote content is granted only to the observers at the Local Counting Center. Nevertheless, the observers possess solely the batch number without any additional information to establish a connection between the vote contents and the respective voter’s identity. Consequently, with a sufficiently large batch size, the privacy of the voter is preserved.

### 3.4 Summary and comparison of different MERGE variants

The main reason for choosing one variant or another is to fit with existing audit processes at the Local Counting Center. If VAULT is already being used, VAULT-MERGE fits easily into the existing process, retains the high efficiency of a ballot-level comparison audit, and inherits the coercion-resistant properties of VAULT. The main remaining disadvantage is that serial numbers are still required, and the protocol therefore has the same susceptibility to coercion as plain MERGE if the coercer colludes with RLA observers.

The other variants batch the MERGE ballots, obviating the need for serial numbers. They have larger anonymity sets but possibly less auditing efficiency. The Sub-MERGE approach has pros and cons compared with plain MERGE. One advantage is that there is no need for serial numbers on ballots. A disadvantage is that the resulting audit may be less efficient than ballot-level comparison audits, though this is unlikely to be a problem for small sets. This is worth considering for jurisdictions that currently do ballot-level comparison audits but are concerned about the privacy implications for their small sets of MERGE voters.

Batch-MERGE is ideal for jurisdictions that already perform batch-level comparison audits.

## 4 Optimizations for multiple contests

Up to now, we have assumed that the election consists of a single contest, resulting in the vote vector  $\mathbf{b} = b_{n-1} \parallel \dots \parallel b_0$  containing one bit per candidate within that contest. Therefore, the encrypted vote vector  $\mathbf{e}$  is computed as:

$$\mathbf{e} = E(\mathbf{b}) = (E(b_{n-1}), \dots, E(b_0))$$

To expand our design to accommodate multiple contests while maintaining a common ballot style, we can consider  $\mathbf{b}$  as a concatenation of multiple vote vectors, with each vote vector corresponding to a distinct contest. For example, in an election featuring two contests, we express  $\mathbf{b}$  as  $\mathbf{b}_1 \parallel \mathbf{b}_0$ , where  $\mathbf{b}_0$  and  $\mathbf{b}_1$  represent the vote vectors for contests 0 and 1, respectively. The encrypted vote vector  $\mathbf{e}$  is accordingly computed as

$$\mathbf{e} = E(\mathbf{b}) = (E(\mathbf{b}_1), E(\mathbf{b}_0))$$

In this configuration, the overall process remains largely unchanged compared to a single-contest election, with the exception that NIZK proofs and discrepancy assignments must be conducted separately for each contest on a ballot.

Furthermore, to extend our design to accommodate elections with multiple ballot styles, it's required to store the ballot style in plaintext alongside each ballot. This is necessary for tallying the ballots within each contest. We must also ensure that the ballot style for each ballot undergoes the same mixnet process as the ballot itself and is presented in plaintext format before the RLA. Otherwise, if the digital ballot that is selected for the RLA lacks a corresponding paper ballot, it would be unclear which contest's RLA statistics should be updated.

Consider  $\mathbf{b} = \mathbf{b}_{k-1} || \dots || \mathbf{b}_0$ , representing a set of  $k$  distinct contests, where each  $\mathbf{b}_i$  is an  $n_i$  vote vector representing the  $i^{\text{th}}$  contest. The total number of options in total is denoted as  $n$ , where  $n = \sum_i n_i$ . In the typical scenario, we would require  $n$  encryption values to represent  $\mathbf{e} = E(\mathbf{b})$ . As mentioned earlier, it's possible to conduct the tally and the RLA process separately for each contest and for elections with different ballot styles. While this approach offers flexibility, it also entails a higher number of encryption values per ballot, namely  $n$ , which can potentially complicate and slow down the mixnet process.

Now, we provide some optimizations for elections with multiple contests.

Let  $\mathbf{e}$  comprise  $n$  ElGamal encryption values corresponding to an  $n$ -bit vote vector. To accelerate the mixnet process, following the tally phase, we apply the following COMP transformation to  $\mathbf{e} = (e_{n-1}, \dots, e_0)$ , with each  $e_i = (\alpha_i, \beta_i)$ , in order to compact it into a single ElGamal encryption value:

$$e' = \text{COMP}(\mathbf{e}) = \left( \prod_i \alpha_i^{2^i} \mod p, \prod_i \beta_i^{2^i} \mod p \right)$$

Our assumption here is that  $n$  is smaller than the size of the group  $|q|$  in bits. For larger vote vectors, we must partition the vector  $\mathbf{e}$  and construct separate ciphertexts for each segment. While this approach is highly efficient, it lacks flexibility for RLA purposes, as outlined below:

- For digital ballots with non-matching serial numbers on paper ballots (i.e., digital ballots with *Type* = "not matching"), worst-given-tally would not be an alternative any more.

## 5 End-To-End Verifiable version (E2E-MERGE)

In the preceding sections, we explained how inconsistencies resulting from flaws in the voting machine can be identified during the audit process and reflected in the RLA results. However, basic-MERGE does not address the strong adversarial model where the adversary fully controls both the voting machine and the mail system. In such a case, the adversary could record an electronic ballot that aligns with their own interests and manipulate the corresponding paper ballot while it is in transit. This particular attack would not be detectable through the standard audit process. To mitigate this risk, we offer an end-to-end verifiable voting system—E2E-MERGE—that empowers voters to detect any discrepancies between their intended votes

and the digital ballots recorded by the voting machines, with a probability of 0.5. The RLA parameters will be accordingly configured based on the (estimated) number of voters participating in the verification process. By introducing this end-to-end verifiable voting system, we aim to enhance transparency and ensure that voters can have confidence in the integrity of the electoral process, even in the face of potential adversarial control over both the voting machine and mail system.

## 5.1 Strong adversarial model

In the *strong adversarial* model, the adversary gains control over both ways of sending the vote: the voting machine and the mail system. The adversary may do *both of*:

- drop any envelope, mailed by a legitimate voter, and stuff new envelopes on behalf of any legitimate voter, and
- control any of the computers used for voting.

The adversary may also control the machines used to process votes at the Local Counting Center.

The adversary may *not* control the devices used for verification at the Remote Voting Center—we assume that each voter has access to at least one trustworthy device that can be used for verification. This device need not be trusted for privacy, because vote verification does not require any information about the vote, so voters can safely ask others to verify their receipts.

## 5.2 Setup

The format of ballots in this version mirrors that of pre-encrypted ballots in ElectionGuard V2.0. Our ballots use the same cryptographic construction as ElectionGuard V2.0, and use similar verification, but are not printed in advance—only the voted ballot will be printed.

A pre-encrypted ballot with  $m$  selection options for voter  $i$ , denoted as  $\mathbf{E}_i$ , is computed as encryption of the  $m \times m$  binary identity matrix  $\mathbf{I}$ , wherein each entry  $\mathbf{E}_{i,j,k}$  for  $j = \{1, \dots, m\}$  and  $k = \{1, \dots, m\}$  represents the encryption of the corresponding entry  $\mathbf{I}_{j,k}$ . The  $j^{\text{th}}$  row of the matrix  $\mathbf{E}_i$  is represented as  $\mathbf{E}_{i,j}$  and corresponds with the  $j^{\text{th}}$  selection option. The encryption vector  $\mathbf{e}_i$  for voter  $i$  is then computed as the product of their vote vector  $\mathbf{b}_i$  and the pre-encrypted ballot  $\mathbf{E}_i$ , as follows:

$$\mathbf{e}_i = \mathbf{b}_i \cdot \mathbf{E}_i$$

To enhance clarity, moving forward, we will omit the index  $i$  unless it is necessary for reference.

Each selection option is hashed to form a *selection hash*, building a selection hash vector  $\mathbf{h}$ , where  $\mathbf{h}_j$  denotes the selection hash for the  $j^{\text{th}}$  row of  $\mathbf{E}$ . Additionally, a condensed representation called the *short code*  $\mathbf{s}_j$  is derived from the selection hash  $\mathbf{h}_j$  using a specialized *hash trimming function*  $\Omega$ . For instance,  $\Omega$  could extract the last byte of its input and represent it in a specified form. All selection hashes within a contest are sorted and then hashed to generate the *contest hash*  $\chi$  specific to that contest. The *confirmation code* for a pre-encrypted ballot is computed as the hash of all the contest hashes present on the ballot in sequential order and is denoted as  $\mathcal{C}$ . Selection hashes, contest hashes and confirmation code are computed as specified by ElectionGuard.

The next section describes the voting process from the voter's perspective. The voting machine creates two pre-encrypted ballots  $\mathbf{E}^1$  and  $\mathbf{E}^2$  for each voter where  $\mathbf{E}^1$ 's short codes are green and  $\mathbf{E}^2$ 's codes are orange.

### 5.3 Voting

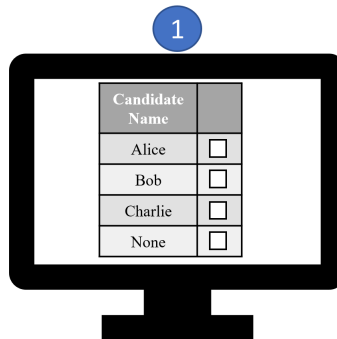
The voting process is as follows:

1. The voting machine creates two ballots  $\mathbf{E}^1$  and  $\mathbf{E}^2$  for voter  $i$  and prints the corresponding confirmation codes  $\mathcal{C}^1$  and  $\mathcal{C}^2$  on the receipt.

(Optional) Verification step. The voter checks that two confirmation codes have been printed.

2. The following steps are repeated for each contest.

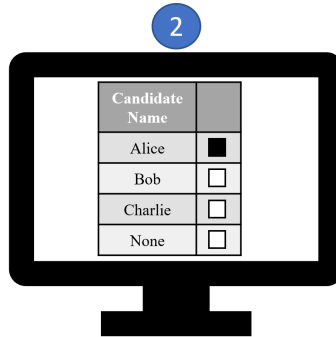
(a) The voting machine displays the options to the voter.



(b) The voter interacts with the machine and makes their selection.<sup>9</sup>

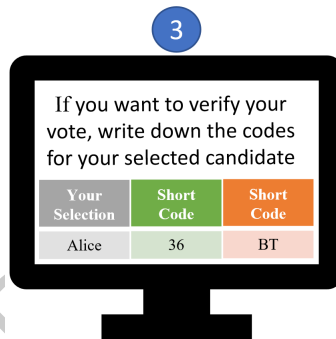
<sup>9</sup>ElectionGuard ballot construction and codes can incorporate multiple selections in the same contest, but not preferential voting or other scoring or ranking systems. We assume in this discussion that only one selection is allowed, and leave the details of multiple selections for later.





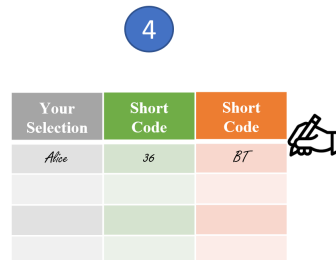
Let their vote vector be  $\mathbf{b}$ .

(c) Then, the set  $\{(\mathbf{o}_j, \mathbf{s}_j^1, \mathbf{s}_j^2) | \mathbf{b}_j = 1\}$  are displayed on the screen.



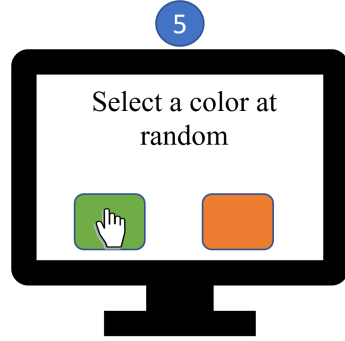
There needs to be an option here, which could be quite unobtrusive, to display all the short codes for the other options along with their corresponding plaintext ballot options (i.e.  $\{(\mathbf{o}_j, \mathbf{s}_j^1, \mathbf{s}_j^2) | \mathbf{b}_j = 0\}$ ). The rationale behind this will be further explored in Section 5.9.

(d) (Optional) Verification step. The voter writes down the short codes corresponding with their intended selection options, for both colors.



We make some blank, colour coded notebooks freely available for everyone.

3. When all their selections are made, the voter randomly selects either  $\mathbf{E}^1$  or  $\mathbf{E}^2$  (green or orange) to cast.



The other will be opened for audit. Let the one selected by the voter be denoted by  $\mathbf{E}^c$  with  $c \in \{1, 2\}$ .

4. The voting machine assigns a unique serial number  $sn_i$  to each ballot paper. The voting machine produces:

- a signed, encrypted electronic record and serial number and a proof of ballot validity:

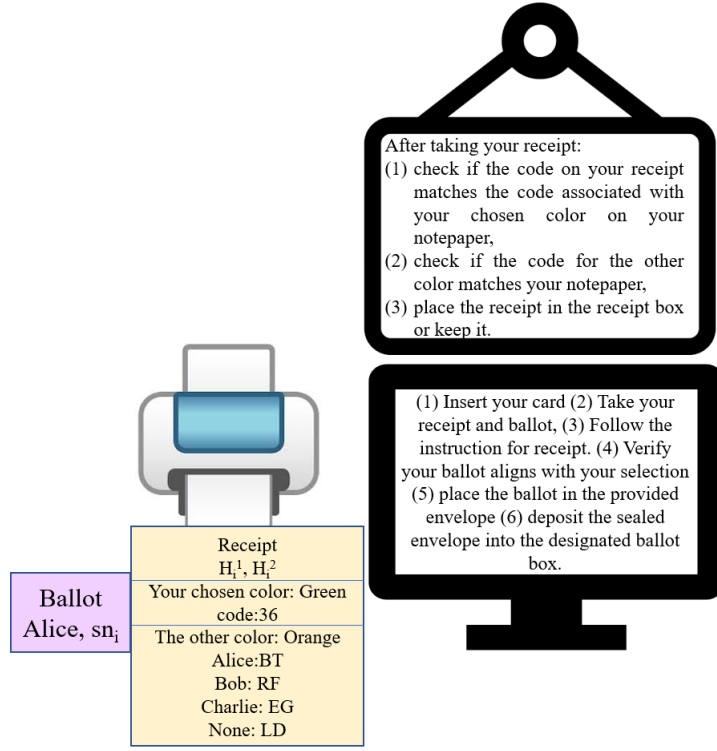
$$sig_i(E(sn), \mathbf{E}^c, ZKP_{VALID}(\mathbf{E}^c)), Domain\_separator\_BB),$$

which goes on the BB,

- a plain paper record of the ballot including  $sn$  and  $\mathbf{b}$ , which is placed in a ballot box or in an envelope for mailing,
- a sticker that includes an address, a traditional mail tracking number, a space for a pen-and-ink signature (if required by regulation—not relevant to our cryptographic protocol) and a digital signature

$$sig_{Voting\ Computer} sig_i(E(sn_i), \mathbf{E}^c, ZKP_{VALID}(\mathbf{E}^c)), Domain\_separator\_mail).$$

- a receipt record of  $\mathcal{C}^1$  and  $\mathcal{C}^2$  (previously printed in Step 1) along with  $\{\mathbf{s}_j^c | \mathbf{b}_j = 1\}$  and  $\mathbf{o}, \mathbf{s}^{3-c}$ , that is, the selected codes for the voted ballot and all the codes for the opened ballot with their corresponding option name.
5. The voter must verify that her plaintext printout matches her intention.
  6. (Optional) Verification step. For cast-as-intended verification of the encrypted vote, the voter should verify that the short codes for her chosen selection on the receipt match what the machine showed her in Step 2c and the voter documented in Step 2d.



## 5.4 Digital path

Digital path is mostly the same as that in the basic-MERGE. The only distinction is that for each voter, the following information is additionally presented on the BB:

- cast ballot additional data:
  - all the selection hashes on the ballot, sorted numerically within each contest, and
  - short codes associated with all selections on the cast ballot made by the voter
- uncast ballot additional data:
  - all the selection hashes on the ballot, sorted numerically within each contest,
  - all options on the ballot,
  - all encrypted values on the ballot, and
  - all short codes,

For example, given that the voter  $i$ 's vote is the  $j^{\text{th}}$  selection option in a contest with  $m$  selection options, the following data is presented on the BB for voter  $i$  ( $c' = 3 - c$ ):

$$\begin{aligned}
& \mathbf{h}_{\sigma(1)}^c \\
& \vdots \\
& \mathbf{h}_{\sigma(j-1)}^c \\
& \mathbf{h}_{\sigma(j)}^c, \mathbf{s}_{\sigma(j)}^c \\
& \mathbf{h}_{\sigma(j+1)}^c \\
& \vdots \\
& \mathbf{h}_{\sigma(1)}^c
\end{aligned}$$

and

$$\begin{aligned}
& \mathbf{o}_{\sigma(1)}, \mathbf{E}_{\sigma(1)}^{c'}, \mathbf{h}_{\sigma(1)}^{c'}, \mathbf{s}_{\sigma(1)}^{c'} \\
& \vdots \\
& \mathbf{o}_{\sigma(m)}, \mathbf{E}_{\sigma(m)}^{c'}, \mathbf{h}_{\sigma(m)}^{c'}, \mathbf{s}_{\sigma(m)}^{c'}
\end{aligned}$$

where  $\sigma$  is a permutation that represents the sorting of the selection hashes. This means that  $\sigma(j) < \sigma(k)$  implies that  $\mathbf{h}_{\sigma(j)} < \mathbf{h}_{\sigma(k)}$ .

Figure 2 illustrates the data published on the BB for E2E-MERGE.

BB							
Voter i							
$\text{Sig}_i(\text{E}(\text{sn}_i, \text{r}_i), \text{E}(A), \text{ZKP})$							
Cast Ballot Conf. code				Uncast Ballot Conf. code			
H1				Bob	E(B)	H1	CD
H2	36			Alice	E(A)	H2	BT
H3				None	E(N)	H3	RG
H4				Charlie	E(C)	H4	OE

**Figure 2:** Bulletin Board data for E2E-MERGE.

## 5.5 Paper path and joining it to the digital path

Paper path and the way paper path's data is joined to the digital path's data similar to those in basic-MERGE.

## 5.6 The RLA

In the strong adversarial model, where the adversary gains control over both the paper channel and the voting machine, conventional RLA prove ineffective. The challenge lies in the adversary's ability to manipulate the voting machine to record their desired vote while simultaneously altering the paper ballot during transportation to match the falsified vote. Even Furthermore, in electronic-only or even a weak adversarial model, if a substantial number of paper ballots are missing, standard RLA or even a complete manual tally may fail to verify the accuracy of the election results. Consider a scenario where electronic records indicate a winning candidate with a lead of 5000 votes over their opponent. However, there are no corresponding paper ballots for at least 2500 electronic records, all of which purportedly indicate votes for the winning candidate. This discrepancy raises serious doubts about the integrity of the election outcome. In such scenarios, we might depend on voters' end-to-end verification to validate the election results.

In our analysis, we assume that the malicious voting machine can only manipulate votes based on the voters' actual intended choices. In other words, the adversary cannot use voters' identities or behavior as criteria for deciding whether to cheat or not. Let the electronic records reflect the victory of the reported winner over their opponent by a margin of  $m$  votes. In this scenario, for the actual loser to appear as the winning candidate, the voting machine would need to manipulate the results by at least  $\frac{m}{2}$  votes. Assuming each voter diligently engages in end-to-end verification with a probability of  $r_{e2e}$ , each instance of fraud is detected and reported by the voter with a probability of  $\frac{r_{e2e}}{2}$  where  $\frac{1}{2}$  reflects this fact that each end-to-end verification conducted during a fraudulent event has a 0.5 chance of detection. Therefore, if there are at least  $\frac{m}{2}$  errors, the probability of detecting  $k \leq \frac{m}{2}$  errors can be expressed as:

$$\binom{\frac{m}{2}}{k} \left(\frac{r_{e2e}}{2}\right)^k \left(1 - \frac{r_{e2e}}{2}\right)^{\frac{m}{2}-k}$$

Similar to RLA, there exists a risk that the end-to-end verification process may confirm an incorrect election result. Our goal is to limit this risk to an acceptable level, ensuring a high probability of rejecting the outcome if it's incorrect. Let  $\alpha$  represent the acceptable risk. Hence, we seek an appropriate threshold, denoted by  $T$ , for the acceptable number of reported errors,

**Table 6:** Reported error threshold for various values of  $m$ ,  $r_{e2e}$  and  $\alpha$ .

$m$	$r_{e2e} = 0.001$ and $\alpha$			$r_{e2e} = 0.002, \alpha$			$r_{e2e} = 0.005, \alpha$			$r_{e2e} = 0.01, \alpha$		
	0.02	0.05	0.10	0.02	0.05	0.10	0.02	0.05	0.10	0.02	0.05	0.10
500	-	-	-	-	-	-	-	-	-	-	-	-
1000	-	-	-	-	-	-	-	-	-	-	-	0
2000	-	-	-	-	-	-	-	-	0	0	1	1
5000	-	-	-	-	-	0	1	1	2	5	6	7
10000	-	-	0	0	1	1	5	6	7	14	16	18
20000	0	1	1	3	4	5	14	16	18	35	38	40

where the associated risk is equal to or lower than  $\alpha$ . The threshold  $T$  is calculated as:

$$\max_T \left\{ \sum_{k=0}^T \binom{\frac{m}{2}}{k} \left( \frac{r_{e2e}}{2} \right)^k \left( 1 - \frac{r_{e2e}}{2} \right)^{\frac{m}{2}-k} \leq \alpha \right\}$$

Table 6 shows some sample figures.

## 5.7 Verification

compared with basic-MERGE, we have some extra verification steps.

### 5.7.1 Matching the Receipt with BB data

The following verification steps are performed using each voter's receipt. However, these steps reveal nothing about how the person voted and can therefore be delegated to any other interested party. For example, voters could be offered the chance to leave their receipts in a special tub where politically-motivated observers at the Remote Voting Center could gather them all and verify them as a set.

Therefore, the voter  $i$  or anyone holding the voter  $i$ 's receipt checks if the information on the receipt match the data presented on the BB:

- the signed, encrypted vote appears on the BB (just like in basic-MERGE),
- short codes  $\{s_j^c | b_j = 1\}$  on her receipt matches short codes from voter  $i$ 's cast ballot additional data on the BB,
- short codes  $s^{3-c}$  on the receipt match the corresponding short codes of voter's  $i$  uncast ballot additional data on the BB.

Voter  $i$  completes the above checks by comparing the receipt with their own electronic data on the BB. Anyone holding the voter  $i$ 's receipt can complete the above checks by comparing the receipt with any voter's electronic records on the BB with matching confirmation codes.

### 5.7.2 Verifying BB data for ballot construction

In addition to the verification steps performed in basic-MERGE, everyone (including voter  $i$  or election authorities) verifies if

- All confirmation codes on the BB are unique.
- The short codes  $\{s_j^c | \mathbf{b}_j = 1\}$  in voter  $i$ 's cast ballot additional data is correctly computed from its corresponding selection hash which is, in turn, correctly computed from the encryption ballot  $E(\mathbf{b})$  on the BB.
- All short codes  $s^{c'}$  with  $c' = 3 - c$  in voter  $i$ 's uncast ballot additional data are correctly computed from their corresponding selection hashes which are, in turn, correctly computed from their corresponding encryption values which are, in turn, correctly computed from their corresponding plaintext options.

## 5.8 Example of E2E-MERGE

To enhance comprehension of E2E-MERGE, we present a simplified illustration in the form of a simple election with two contests. The first contest contains 4 selection options as Alice, Bob, Charlie and None. The second contest contains 2 selection options as YES and NO. Each selection option within a contest corresponds with a vote vector as follows:

Alice: [1 0 0 0]  
 Bob: [0 1 0 0]  
 Charlie: [0 0 1 0]  
 None: [0 0 0 1]  
 YES: [1 0]  
 NO: [0 1]

Two pre-encrypted ballots are created for voter  $i$ . For the sake of simplicity assume that each encrypted or hash value is represented by a 4-character hexadecimal value (i.e., a value

between 0x0000 and 0xFFFF) and each short code is represented by a combination of a digit and a capital letter (e.g., 4M). Then, the first pre-encrypted ballot is created as follows:

$$\begin{aligned}
\text{Alice: } [1\ 0\ 0\ 0] &\xrightarrow{E} [E(1)\ E(0)\ E(0)\ E(0)] = [0x5148\ 0x3B68\ 0x9F42\ 0xA943] \xrightarrow{H} \\
&H(0x5148, 0x3B68, 0x9F42, 0xA943) = 0x8731 \xrightarrow{\Omega} 3M \\
\text{Bob: } [0\ 1\ 0\ 0] &\xrightarrow{E} [0x128D\ 0x0642\ 0x7B44\ 0x91C4] \xrightarrow{H} 0x309C \xrightarrow{\Omega} 6H \\
\text{Charlie: } [0\ 0\ 1\ 0] &\xrightarrow{E} [0x387F\ 0x13C6\ 0x8E45\ 0xDE40] \xrightarrow{H} 0x9B31 \xrightarrow{\Omega} 9L \\
\text{None: } [0\ 0\ 0\ 1] &\xrightarrow{E} [0x3927\ 0x168C\ 0xBB73\ 0x12E4] \xrightarrow{H} 0x28E0 \xrightarrow{\Omega} 9U \\
\text{YES: } [1\ 0] &\xrightarrow{E} [0xFA59\ 0xCB07] \xrightarrow{H} 0x928F \xrightarrow{\Omega} 1J \\
\text{NO: } [0\ 1] &\xrightarrow{E} [0x66A3\ 0xF15D] \xrightarrow{H} 0x5B28 \xrightarrow{\Omega} 7W
\end{aligned}$$

where the contest hash value for the first and second contests are computed as follows:

$$\begin{aligned}
\chi_1^1 &= H(0x28E0, 0x309C, 0x8731, 0x9B31) = 0x7044 \\
\chi_2^1 &= H(0x5B28, 0x928F) = 0x32FB
\end{aligned}$$

The confirmation code for this ballot is computed as:

$$C^1 = H(0x7044, 0x32FB) = 0x1A75$$

Similarly, the second pre-encrypted ballot is created as follows:



$$\begin{aligned}
\text{Alice: } [1 \ 0 \ 0 \ 0] &\xrightarrow{E} \\
&[E(1) \ E(0) \ E(0) \ E(0)] = [0x49D3 \ 0xA6F9 \ 0xBF34 \ 0x10F4] \xrightarrow{H} \\
&H(0x49D3, 0xA6F9, 0xBF34, 0x10F4) = 0x3450 \xrightarrow{\Omega} 7R \\
\text{Bob: } [0 \ 1 \ 0 \ 0] &\xrightarrow{E} [0x9C3A \ 0x1F5A \ 0x999C \ 0x735A] \xrightarrow{H} 0xB23C \xrightarrow{\Omega} 1K \\
\text{Charlie: } [0 \ 0 \ 1 \ 0] &\xrightarrow{E} [0x26A2 \ 0x5A35 \ 0x3A20 \ 0x0500] \xrightarrow{H} 0xA366 \xrightarrow{\Omega} 6V \\
\text{None: } [0 \ 0 \ 0 \ 1] &\xrightarrow{E} [0x55D3 \ 0x29A8 \ 0x4F3A \ 0x9745] \xrightarrow{H} 0xED4F \xrightarrow{\Omega} 8H \\
\text{YES: } [1 \ 0] &\xrightarrow{E} [0x84D7 \ 0xCE48] \xrightarrow{H} 0x185D \xrightarrow{\Omega} 9N \\
\text{NO: } [0 \ 1] &\xrightarrow{E} [0x4868 \ 0x195D] \xrightarrow{H} 0x3A8F \xrightarrow{\Omega} 3Q
\end{aligned}$$

where the contest hash value for the first and second contests are computed as follows:

$$\begin{aligned}
\chi_1^2 &= H(0x3450, 0x9B31, 0xA366, 0xB23C) = 0x55D3 \\
\chi_2^2 &= H(0x185D, 0x3A8F) = 0xDA49
\end{aligned}$$

The confirmation code for this ballot is computed as:


$$C^2 = H(0x55D3, 0xDA49) = 0x308A$$

The ballots are displayed to the voter as illustrated below:

Your Selection	Short Code	Short Code
<input type="checkbox"/> Alice	3M	7R
<input type="checkbox"/> Bob	6H	1K
<input type="checkbox"/> Charlie	9L	6V
<input type="checkbox"/> None	9U	8H
<input type="checkbox"/> YES	1J	9N
<input type="checkbox"/> NO	7W	3Q

Let the voter select Alice and YES. Then she writes down the corresponding short codes as illustrated below:

Your Selection	Short Code	Short Code
Alice	3M	7R
YES	1J	9N



Suppose she selects green to cast and orange to audit. Then the receipt for her vote would contain the following information:

Confirmation codes:  $C^1 = 0x1A75, C^2 = 0x308A$   
Chosen color: Green  
Green codes for your selections: 3M, 1J  
Other color: Orange  
Orange Codes: Alice:7R, Bob:1K, Charlie:6V, None:8H  
YES:9N, NO:3Q

The voter checks if the selected color matches her intention and codes on the receipt match the ones written on her notepaper. The corresponding encrypted vote that is stored on the BB for the voter would be

$[0x5148 \ 0x3B68 \ 0x9F42 \ 0xA943], [0xFA59 \ 0xCB07]$

which corresponds with Alice and YES on the first pre-encrypted ballot. From this ballot, the confirmation code, all selection hashes, sorted numerically within each contest, along with the short codes corresponding with the voter's selection options are also published on the BB. So,

for our sample ballots, the data below is published on the BB:

Conf. Code: 0x1A75  
 Contest 1  
 $0x28E0, 0x309C, 0x8731 \xrightarrow{\Omega} 3M, 0x9B31$   
 Contest 2  
 $0x5B28, 0x928F \xrightarrow{\Omega} 1J$

For the second ballot, which is the audit ballot, the confirmation code, selection options, encrypted values, selection hashes and short codes are recorded on the BB as follows:

Conf. Code: 0x1A75  
 Contest 1  
 Alice:  $[0x49D3 \ 0xA6F9 \ 0xBF34 \ 0x10F4], 0x3450, 7R$   
 Charlie:  $[0x26A2 \ 0x5A35 \ 0x3A20 \ 0x0500], 0xA366, 6V$   
 Bob:  $[0x9C3A \ 0x1F5A \ 0x999C \ 0x735A], 0xB23C, 1K$   
 None:  $[0x55D3 \ 0x29A8 \ 0x4F3A \ 0x9745], 0xED4F, 8H$   
 Contest 2  
 NO:  $[0x4868 \ 0x195D]0x3A8F, 3Q$   
 YES:  $[0x84D7 \ 0xCE48], 0x185D, 9N$

## 5.9 Security analysis

The correctness, authentication and privacy of this version remain consistent with those of basic-MERGE.

### 5.9.1 End-to-end verifiability

This version provides end-to-end verifiability because the voter is able to verify that their votes are cast as intended, collected as cast and tallied as collected. In more detail, each voter is able to challenge the voting machine to ensure that her vote is cast as intended. If the encrypted values on the paper ballots (e.g.  $E(b_{i,j})$ ) do not match with their corresponding unencrypted values on the paper ballot (e.g.  $b_{i,j}$ ), then there is a 50% probability that the voter detects it

when the non-cast ballot set is decrypted on the BB. The voter also verifies that their ballot was collected as cast by comparing their receipt with the information published on the BB alongside their name and verifying that the ciphertexts were correctly constructed. Everyone, including the voter, will be able to verify that the votes are tallied as collected by verifying the correctness of all operations (i.e. mix and decryption) performed on the cast electronic ballots which is published on the BB.

### 5.9.2 Receipt Freeness

The coercibility will be discussed further. Assuming that the coercer requests the receipt and short code of the uncast ballot, as documented by the voter during the voting process, certain implications arise. If the voting machine solely displays the short codes for the voter's selected options, the remaining short codes on the uncast ballot would remain unknown to the voter. Consequently, the voter is faced with a dilemma of either providing the actual documented short code to the coercer, which would compromise the secrecy of their vote once uncast ballots are revealed on the BB, or providing a fake short code, which would likely be detected by the coercer as it is improbable for a fake short code to appear on the BB and align with the coercer's intention. To address this issue, the voting machine should present all short codes and their corresponding plaintext ballot options for the uncast ballot to the voter.

## 6 Protocol specification

### 6.1 Encryption scheme

The parameters of the ElGamal cryptosystem  $\mathcal{G} = (p, q, g)$  are specified as follows:  $p$  is a prime number given by  $2kq + 1$ , where  $q$  is also a prime number, and  $g$  represents a generator of the order  $q$  subgroup of  $\mathbb{Z}_p^*$ . ElGamal cryptosystem includes four main algorithms as follows:

---

#### ElGamal key generation $\text{KeyGen}(\mathcal{G})$

---

Input: ElGamal parameters  $\mathcal{G} = (p, q, g)$

Output: Public-private key pair  $(K, s)$

1.  $s \xleftarrow{\$} \mathbb{Z}_q^*$
2.  $K = g^s \bmod p$
3. Output  $(K, s)$

---

### ElGamal encryption

---

Input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , message  $\sigma \in \mathbb{Z}_q^*$ , public key  $K$

Output:  $E(\sigma)$

1.  $\xi \xleftarrow{\$} \mathbb{Z}_q^*$
2.  $\alpha = g^\xi \bmod p, \beta = K^\sigma K^\xi \bmod p$
3. Output  $(\alpha, \beta)$

---

### ElGamal encryption aggregation

---

Input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , encryption values  $(\alpha_i, \beta_i) = E(\sigma_i)$

Output:  $E(\sum_i \sigma_i \bmod q)$

1.  $A = \prod_i \alpha_i \bmod p, B = \prod_i \beta_i \bmod p$
2. Output  $(A, B)$

---

### ElGamal decryption

---

Input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $C = (A, B)$ , private key  $s$

Output:  $D(C; s)$

1.  $T = \frac{B}{A^s} \bmod p$
2. Output  $T$

## 6.2 Hash computation

The function  $H$  that is used in this whole section is the same as one specified in ElectionGuard. It is based on hashed message authentication code HMAC-SHA-256. The function  $H$  takes two inputs,  $B_0$  and  $B_1$ , where  $B_0$  is 256 bit long and corresponds to the key in HMAC and  $B_1$ , which is of arbitrary length, is the actual input to the HMAC. These inputs are separated by a semicolon. If  $B_1$  is comprised of multiple elements (e.g.  $B_1 = a||b||c$ ), then these elements are separated by commas (e.g.  $H(B_0; a, b, c)$ ).

---

## Hash function

---

Input:  $B_0$  and  $B_1 = B_{11} || B_{12} || \dots || B_{1n}$

Output:  $H(B_0; B_1)$

1. Output  $\text{HMAC-SHA-256}(B_0, B_1)$

## 6.3 Distributed ElGamal

Distributed ElGamal is a variant of the traditional ElGamal encryption scheme that involves multiple parties collaboratively generating the encryption keys and decrypting the ciphertexts.

---

### Key generation

---

Principals:  $n$  guardians  $G_i$  with  $i = \{1, \dots, n\}$

Input: ElGamal parameters  $\mathcal{G} = (p, q, g)$

Output: Public key shares  $K_i$  for  $i = \{1, \dots, n\}$ , aggregated public key  $K$

Output ( $G_i$ ): Private key share  $s_i$

1.  $G_i$ :  $(K_i, s_i) \leftarrow \text{KeyGen}(\mathcal{G})$ .
2.  $G_i$ : Send  $K_i$  and  $\text{KnowDlog}(g, K_i)$  to  $G_j$  for any  $j \neq i$
3.  $G_i$ : Proceed if all proofs  $\text{KnowDlog}(g, K_i)$  for  $i = \{1, \dots, n\}$  are verified. Otherwise, stop.
4.  $G_i$ : Output private key share  $s_i$
5. All guardians: Output public key shares  $K_i$  for  $i = \{1, \dots, n\}$  the aggregated public key  $K = \prod_i K_i \mod p$

---

### Decryption

---

Principals:  $n$  guardians  $G_i$  with  $i = \{1, \dots, n\}$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $C = (A, B)$ , public key shares  $K_i$  for  $i = \{1, \dots, n\}$

Private input ( $G_i$ ): Private key share  $s_i$

Output:  $\text{DistDec}(C, \sum_{i=1}^n s_i)$

1.  $G_i$ : Publish share  $M_i = A^{s_i} \bmod p$  and proof  $\text{EqDlogs}(g, A, K_i, M_i)$ .
2. Proceed if all proofs  $\text{EqDlogs}(g, A, K_i, M_i)$  for  $i = \{1, \dots, n\}$  are verified. Otherwise, stop.
3. Compute  $M = \prod_i M_i \bmod p$
4. Output  $T = B/M \bmod p$

Required operations per guardian: 1 exp., 1 EqDlogs,  $n - 1$  EqDlogs verification: in total 3 exp,  $2(n - 1)$  multi-exp

Required operations for verification:  $n$  EqDlogs verification:  $2n$  multi-exp

Note that in the protocol presented above, anyone with access to the public input and the proofs  $\text{EqDlogs}(g, A, K_i, M_i)$  for  $i = \{1, \dots, n\}$  can verify the correctness of the distributed decryption operation by following steps 2-4 of the protocol. With the following protocol, guardians can collaboratively provide a proof of decryption correctness that can be verified by anyone who has access to the aggregated public key  $K$  rather than each individual guardian's public key share.

---

### Proof of decryption correctness

---

Principals:  $n$  guardians  $G_i$  with  $i = \{1, \dots, n\}$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $C = (A, B)$ , public key shares  $K_i$  for  $i = \{1, \dots, n\}$ , aggregated public key  $K$  and two deployment-dependant constants  $cons_0$  and  $cons_1$

Private input ( $G_i$ ): Private key share  $s_i$

Output (guardians): Plaintext  $T := \text{DistDec}(C; K)$  and proof  $\text{ZKP}_{\text{DECRYPT}}(T, C)$

1.  $G_i$ : Compute share  $M_i = A^{s_i} \bmod p$  and send  $M_i$  to  $G_j$  for any  $j \neq i$
2.  $G_i$ : Compute  $M = \prod_j M_j \bmod p$ ,  $T = \frac{B}{M} \bmod p$ ,  $u_i \xleftarrow{\$} \mathbb{Z}_q^*$ ,  $a_i = g^{u_i} \bmod p$  and  $b_i = A^{u_i} \bmod p$  and send  $(a_i, b_i)$  to  $G_j$  for any  $j \neq i$
3.  $G_i$ : Compute  $a = \prod_j a_j$  and  $b = \prod_j b_j$  and  $c = H(cons_0; cons_1, K, A, B, a, b, M)$
4.  $G_i$ : Send  $v_i = u_i - cs_i$  to  $G_j$  for any  $j \neq i$
5.  $G_i$ : Compute  $a'_j = g^{v_j} K_j^c$  and  $b'_j = A^{v_j} M_j^c$  and verify if  $a_j = a'_j$  and  $b_j = b'_j$  for any  $j \neq i$ . Otherwise, stop.
6. All guardians: Compute  $v = \sum_j v_j$  and send the plaintext  $T$  and the proof  $(c, v)$  to  $V$

7.  $V$ : Compute  $M = \frac{B}{T} \bmod p$ ,  $a = g^v K^c \bmod p$  and  $b = A^v M^c \bmod p$  and verify if  $v \in \mathbb{Z}_q$  and  $c = H(\text{cons}_0; \text{cons}_1, K, A, B, a, b, M)$ . Otherwise, reject.

Required operations per guardian: 3 exp.,  $2(n-1)$  multi-exp

Required operation for verification: 2 multi-exp

## 6.4 Zero-knowledge proofs

---

KnowDlog( $g, K$ )

---

Principals: The prover  $P$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ ,  $K$  and two deployment-dependant constants  $\text{cons}_0$  and  $\text{cons}_1$

Private input ( $P$ ):  $s$  s.t.  $K = g^s \bmod p$

1.  $P$ : computes

- $u \xleftarrow{\$} \mathbb{Z}_q$
- $h = g^u \bmod p$
- $c = H(\text{cons}_0; \text{cons}_1, K, h)$
- $v = u - cs \bmod q$

2.  $P \xrightarrow{c,v} V$

3.  $V$ : Compute  $h = g^v K^c \bmod p$  and then verify if  $c = H(\text{cons}_0; \text{cons}_1, K, h)$ . Otherwise, reject.

Required operations: 1 exp

Required operation for verification: 1 multi-exp

---

EqDlogs( $x, y, X, Y$ )

---

Principals: The prover  $P$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ ,  $x, y, X, Y$  and two deployment-dependant constants  $\text{cons}_0$  and  $\text{cons}_1$

Private input ( $P$ ):  $s$  s.t.  $X = x^s \bmod p$  and  $Y = y^s \bmod p$

1.  $P$ : compute



- $u \xleftarrow{\$} \mathbb{Z}_q$
- $h = x^u \mod p$
- $h' = y^u \mod p$
- $c = H(\text{cons}_0; \text{cons}_1, x, y, X, Y, h, h')$
- $v = u - cs \mod q$

2.  $P \xrightarrow{h, h', c, v} V$

3.  $V$ : Compute  $h = x^v X^c \mod p$  and  $h' = y^v Y^c \mod p$  and verify if  $c = H(\text{cons}_0; \text{cons}_1, x, y, X, Y, h, h')$ . Otherwise, reject.

Required operations: 2 exp

Required operation for verification: 2 multi-exp

---

**Proof that  $(a, b)$  is an encryption of an integer in the range  $0, \dots, L$**

---

Principals: The prover  $P$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $(\alpha, \beta)$ , public key  $K$  and two deployment-dependant constants  $\text{cons}_0$  and  $\text{cons}_1$

Private input ( $P$ ):  $\xi, l$  s.t.  $(\alpha, \beta) = (g^\xi, K^l K^\xi)$

1.  $P$ :

- $u_j \xleftarrow{\$} \mathbb{Z}_q$  for  $j = \{0 \dots, L\}$
- $c_j \xleftarrow{\$} \mathbb{Z}_q$  for  $j = \{0 \dots, L\}, j \neq l$
- $t_j := (u_j + (l - j)c_j) \mod q, (a_j, b_j) := (g^{u_j} \mod p, K^{t_j} \mod p)$  for  $j = \{0 \dots, L\}, j \neq l$ ,
- $(a_l, b_l) := (g^{u_l} \mod p, K^{u_l} \mod p)$
- $c = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1, \dots, a_L, b_L)$
- $c_l := (c - \sum_{j \neq l} c_j) \mod q$
- $v_j = (u_j - c_j \xi) \mod q$
- Send  $(c_j, v_j)$  for  $j = \{0, \dots, L\}$  to  $V$

2.  $V$ :

(a)  $(a_j, b_j) := (g^{v_j} \alpha^{c_j} \mod p, K^{v_j - j c_j} \beta^{c_j} \mod p)$

(b) Verify if  $\sum_j c_j = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1, \dots, a_L, b_L)$ . Otherwise, reject.

Required operations:  $2L$  exp

Required operation for verification:  $2L$  multi-exp

---

**Proof that  $(a, b) = (g^\xi, K^\xi)$  is an encryption of zero or one**

---

Principals: The prover  $P$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $(\alpha, \beta) = (g^\xi, K^\xi)$ , public key  $K$  and two deployment-dependant constants  $\text{cons}_0$  and  $\text{cons}_1$

Private input ( $P$ ):  $\xi$  s.t.  $(\alpha, \beta) = (g^\xi, K^\xi)$

1.  $P$ :

- $u_0, u_1, c_1 \xleftarrow{\$} \mathbb{Z}_q$
- $(a_0, b_0) = (g^{u_0} \bmod p, K^{u_0} \bmod p), (a_1, b_1) = (g^{u_1} \bmod p, K^{u_1-c_1} \bmod p)$
- $c = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1)$
- $c_0 = c - c_1, v_0 = u_0 - c_0\xi \bmod q, v_1 = u_1 - c_1\xi \bmod q$
- Send  $(c_0, c_1, v_0, v_1)$  to  $V$

2.  $V$

- (a)  $(a_0, b_0) := (g^{v_0}\alpha^{c_0} \bmod p, K^{v_0}\beta^{c_0} \bmod p)$
- (b)  $(a_1, b_1) := (g^{v_1}\alpha^{c_1} \bmod p, K^{v_1-c_1}\beta^{c_1} \bmod p)$
- (c) Verify if  $c_0 + c_1 = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1)$ . Otherwise, reject.

Required operations: 4 exp

Required operation for verification: 4 multi-exp

---

**Proof that  $(\alpha, \beta) = (g^\xi, KK^\xi)$  is an encryption of zero or one**

---

Principals: The prover  $P$  and the verifier  $V$

Public input: ElGamal parameters  $\mathcal{G} = (p, q, g)$ , ciphertext  $(\alpha, \beta) = (g^\xi, KK^\xi)$ , public key  $K$ , two deployment-dependant constants  $\text{cons}_0$  and  $\text{cons}_1$

Private input ( $P$ ):  $\xi$  s.t.  $(\alpha, \beta) = (g^\xi, KK^\xi)$

1.  $P$ :

- $u_0, u_1, c_0 \xleftarrow{\$} \mathbb{Z}_q$
- $(a_0, b_0) = (g^{u_0} \bmod p, K^{u_0+c_0} \bmod p), (a_1, b_1) = (g^{u_1} \bmod p, K^{u_1} \bmod p)$
- $c = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1)$
- $c_1 = c - c_0, v_0 = u_0 - c_0\xi \bmod q, v_1 = u_1 - c_1\xi \bmod q$
- Send  $(c_0, c_1, v_0, v_1)$  to  $V$

2.  $V$

- (a)  $(a_0, b_0) := (g^{v_0}\alpha^{c_0} \bmod p, K^{v_0}\beta^{c_0} \bmod p)$
- (b)  $(a_1, b_1) := (g^{v_1}\alpha^{c_1} \bmod p, K^{v_1-c_1}\beta^{c_1} \bmod p)$
- (c) Verify if  $c_0 + c_1 = H(\text{cons}_0; \text{cons}_1, K, \alpha, \beta, a_0, b_0, a_1, b_1)$ . Otherwise, reject.

Required operations: 4 exp

Required operation for verification: 4 multi-exp

## References

- [ADS20] Andrew W Appel, Richard A DeMillo, and Philip B Stark. Ballot-marking devices cannot ensure the will of the voters. *Election Law Journal: Rules, Politics, and Policy*, 19(3):432–450, 2020.
- [BS12] Jorge H Banuelos and Philip B Stark. Limiting risk by turning manifest phantoms into evil zombies. *arXiv preprint arXiv:1207.3413*, 2012. <https://arxiv.org/abs/1207.3413>.
- [BST19] Josh Benaloh, Philip B Stark, and Vanessa Teague. Vault: Verifiable audits using limited transparency. *Proceedings of E-Vote ID*, 2019. <https://www.stat.berkeley.edu/~stark/Preprints/vault19.pdf>.
- [KBW21] Philip Kortum, Michael D Byrne, and Julie Whitmore. Voter verification of ballot marking device ballots is a two-part question: Can they? mostly, they can. do they? mostly, they don't. *Election Law Journal: Rules, Politics, and Policy*, 20(3):243–253, 2021.
- [Rivo8] Ronald L Rivest. On the notion of ‘software independence’ in voting systems. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3759–3767, 2008. <https://people.csail.mit.edu/rivest/pubs/RW06.pdf>.

- [Sta23] Philip B Stark. Overstatement-net-equivalent risk-limiting audit: Oneaudit. In *Financial Cryptography and Data Security - Voting Workshop*. Springer LNCS 13953, 2023. <https://arxiv.org/abs/2303.03335>.

DRAFT