Efficient Receipt-Free Ballot Casting Resistant to Covert Channels

Ben Adida
Harvard University
ben_adida@harvard.edu

C. Andrew Neff
candrewneff@verizon.net

Abstract

Open-audit election protocols have, in recent years, focused on the ballot casting process, where the difficulty lies in proving to the unaided voter that the encrypted ballot faithfully captures her intent. We present MarkPledge2, an evolution of Neff's MarkPledge scheme and the first efficient, covert-channel-resistant, receipt-free ballot casting scheme that can be used by humans without trusted hardware. In comparison to the Moran-Naor proposal at CRYPTO 2006 which shares some of our goals, our scheme produces a significantly shorter ballot, prevents covert channels in the ballot, and opts for statistical soundness rather than everlasting privacy (achieving both seems impossible). The human interface remains the same as MarkPledge, requiring of the voter only very-short-string comparisons. This work solves an "interesting open problem" raised in the original Moran-Naor paper.

Keywords: Voting, Receipt-Freeness, Covert Channels

1 Introduction

Elections require two seemingly contradictory properties: Alice, the voter, should be able to verify that her vote was properly captured into the final tally but should not be able to convincingly reveal this vote to a coercer even if she is willing to be coerced. Classic voting schemes—the ones we use in democratic elections around the world—compromise on verifiability in favor of incoercibility (also called *receipt freeness*): voters trust election officials to implement a secure chain of custody of anonymized ballots, where even a small mistake could surreptitiously corrupt the election result.

Open-audit voting schemes strive to achieve *both* verifiability and receipt-freeness. Typically, these schemes are structured much like public elections of yore: cast ballots are available for all to see on a public bulletin board, and the tabulation process is public. The major difference is that ballots are encrypted through the entire public audit process. Using various proof techniques, the voting machine demonstrates to Alice that her vote was properly encrypted, and the election trustees demonstrate to any observer that the encrypted ballots are properly tallied.

While many contributions have been made to the field of cryptographic voting over the last 25 years, the ballot casting portion has garnered significant interest only recently. Early research presumed that each voter would own a trusted device to perform cryptographic operations on her behalf. Unfortunately, this does not accurately model real-world voters for the foreseeable future. In 2004, Chaum [8] and Neff [5] independently introduced mechanisms that enable an unaided human to cast an encrypted ballot without trusting the voting machine. More recently, Moran and Naor [19] presented a new definitional framework for receipt-freeness and a new ballot-casting scheme, based on Neff's earlier technique, MarkPledge. In a concluding remark, Moran and Naor mention that preventing covert channels in the ballot is "an interesting open problem."

1.1 Our Contributions

We propose a new ballot casting scheme, MarkPledge2, which builds on Neff's MarkPledge scheme [5]. MarkPledge2 generates a much shorter ballot than either MarkPledge or the Moran-Naor scheme. In addition, it solves an important open problem: the voting machine in MarkPledge2 can no longer leak the voter's choice in her encrypted ballot without producing a corrupt ballot and getting caught with high probability. We prove receipt-freeness in the model proposed by Moran and Naor, with notably weaker assumptions regarding the voting device.

Privacy and Soundness. The Moran-Naor scheme provides everlasting privacy and computational soundness, while MarkPledge provides computational privacy and statistical soundness. In other words, an exceptionally powerful adversary could violate voter privacy in MarkPledge or corrupt the election result in the Moran-Naor scheme. The choice between these two approaches is difficult, and there is no obviously superior approach: policy makers will have to decide which security property to prioritize.

In this work, we present our improved ballot size and covert-channel resistance in the statistically sound and computationally private setting of MarkPledge. We leave as an interesting open problem the application of these techniques to the computationally sound and statistically private setting of Moran-Naor, which will likely require more than simply replacing encryptions with commitments.

1.2 Ballot Casting Schemes

In open-audit ballot casting schemes, Alice typically interacts with a voting machine to generate an encryption (or commitment) of her plaintext ballot. Alice cannot generate this encrypted ballot on her own, as she would clearly have an easily transferable receipt of her actions. At the same time, Alice should not trust the voting machine to correctly capture her vote. Thus, typically, the voting machine generates the encrypted ballot and provides a non-transferable proof to Alice that it did so correctly.

In MarkPledge, the Moran-Naor scheme, and our new proposal MarkPledge2, Alice chooses one candidate from a fixed list using as representation an integer index into the candidate array. Once she enters her selection, the voting machine produces a ballot ciphertext (or, in the case of the Moran-Naor scheme, a commitment) and performs a zero-knowledge proof, with Alice providing the challenge, that it did so correctly. Then, to ensure that the protocol is incoercible, the voting machine simulates proofs that this same ballot encodes every other possible candidate. The voting machine includes all proof transcripts, one real and the rest simulated, on Alice's receipt. A third party thus cannot tell which proof was performed correctly and which were simulated.

Human Verifier. The MarkPledge, Moran-Naor, and MarkPledge2 ballot casting schemes ensure that Alice can verify her vote with only minimal computation inside the voting booth: she only checks the *proper order* of the proof messages for her chosen candidate during the exchange with the voting machine. Then, outside the booth, Alice relies on a computer—possibly her own, but more likely one belonging to an organization she trusts, e.g. her political party—to ensure that the proof transcripts are otherwise correct. Importantly, the external computer checks both the real and simulated proofs; in fact it has no way of knowing which is real and which is simulated.

To enable this human verification in practice, the voting machine's first proof message and Alice's challenge must be very short strings, so that Alice can compare and manipulate them. (Icons, or any other memorable representation with relatively small domain, can also be used.) It isn't enough to adapt existing proof protocols by selecting short first and second proof messages: they must remain short and identically distributed at simulation time. The crux of a human-verifiable voting scheme is precisely to provide a proof protocol whose first two messages are naturally short and whose soundness is maximized with respect to the challenge length. In addition, for our covert-channel resistance approach, we need the proof to be simulatable in a particular order—messages 2, 1, then 3—that is not supported by typical 3-round zero-knowledge proofs. We explain this requirement in Section 4.

1.3 An Overview of MarkPledge2

The MarkPledge Structure. The key idea of the MarkPledge2 protocol, like MarkPledge, is to use a special bit-encryption scheme BitEnc that enables a human-friendly proof that a ciphertext encrypts the bit $b = 1$. This special bit encryption schemes function as follows:

1. The plaintext bit b is represented as a message m. There are many message representations possible for each of the two possible values of bit b. The specifics of the message representation approach differ between MarkPledge and MarkPledge2, but in either case, m is composed of at least two plaintexts, each in the domain of an additively homomorphic encryption scheme, e.g. Exponential El Gamal. The message representation depends in part on a security parameter κ.

2. The individual plaintexts that compose m are encrypted using the additively homomorphic scheme, and the bit encryption $\mathsf{BitEnc}(b)$ is then the sequence of the encryptions of the individual plaintexts of m.

3. To prove that $b = 1$, the encryptor produces a commitment string that is related to the message representation m. For a given m, there is exactly one possible commitment string.

4. The verifier produces a challenge string chal of length κ, the security parameter mentioned earlier that impacts the message representation m.

5. The prover reveals a portion of the randomness used to encrypt the individual components of the message representation m. The verifier can check this reveal to assure himself that, indeed, $b = 1$. However, if the prover knows chal in advance, then it is easy for him to produce a message representation m' for $b = 0$ that will match the verifier's expectation given chal. In other words, the protocol transcript can be easily simulated.

6. The protocol lends itself to the full utilization of the bits of chal. In other words, the protocol can be performed by a human who will only need to compare strings of size κ while obtaining confidence of correct bit encryption with probability $2^{-\kappa}$.

In MarkPledge, m is a sequence of pairs $(m_{1,1}, m_{1,2}), \ldots, (m_{\kappa,1}, m_{\kappa,2})$, with each $m_{i,j}$ a single bit, and $\mathsf{c} = \mathsf{BitEnc}(b)$ the corresponding sequence of pairs of Exponential El Gamal encryptions of the $m_{i,j}$. If $b = 1$, then the two bits of each pair are identical: $m_{i,1} = m_{i,2}, i \in [1, \kappa]$. Conversely, if $b = 0$, the bits within each pair are different: $m_{i,1} = 1 - m_{i,2}, i \in [1, \kappa]$. If the pairs are not consistent within m, the plaintext is considered invalid and denoted $\perp$. Note how there are 2^κ valid message representations for $b = 1$, and 2^κ valid message representations for $b = 0$. One can pledge that c encrypts a 1 by stating, for each pair, the bit that each message encodes. For example, a pledge of `11001` indicates that $\mathsf{m} = (1,1),(1,1),(0,0),(0,0),(1,1)$. A challenge chal is then a string of κ bits, where each bit $\mathsf{chal}[i]$ determines whether the prover should reveal the first or second element of pair i. Clearly, if m is an encoding of 1, the pledge string will match the revealed string no matter what the challenge. If m is an encoding of 0, the pledge string will match the revealed string for only one possible challenge value, with probability $2^{-\kappa}$. Figure 1 illustrates this setup. The Moran-Naor scheme is very similar, though it uses commitments instead of ciphertexts.

A Shorter Bit Ciphertext. In MarkPledge2, we develop a bit-encryption mechanism that achieves the same properties with a much shorter message representation m for a given challenge length κ. Specifically, we consider $SO(2, q)$, the special orthogonal group of two-by-two matrices with elements in $\mathbf{Z}_q$, which is isomorphic to the group of rotations represented as vectors in $\mathbf{Z}_q^2$. Subgroups of this group cleanly expose equivalence classes of "1-vectors", "0-vectors", and "test-vectors", such that any 1-vector and 0-vector are bisected by a test-vector. Each vector can be encrypted using two Exponential El Gamal ciphertexts, which lets one homomorphically compute the dot product of an encrypted 1 or 0 vector with a plaintext test vector: any two vectors bisected by the test vector in question will yield the same dot product value. A pledge is thus simply an index into the 1-vector equivalence class; if one pledges a 1-vector when c actually encrypts a 0-vector, the only way to go undetected is if the challenge is the single test vector that bisects the claimed 1-vector and actually encrypted 0-vector. Figure 1 compares the message representation schemes of MarkPledge and MarkPledge2.

The Ballot. The MarkPledge2 scheme is effectively MarkPledge augmented with the more efficient bit encryption scheme. If there are M candidates on the ballot, the voting machine produces a sequence of M bit encryptions $\mathsf{c}_1, \ldots, \mathsf{c}_M$. Then, it proves that c_a, where a is Alice's choice, is indeed the bit encryption of 1, by pledging its representation m_a, accepting chal from Alice, and partially revealing m_a accordingly. It can then simulate that the other bit encryptions c_i are also encryptions of 1, using additional challenges provided by Alice. The difference is merely in the timing: for c_a, the voting machine must pledge m_a before it sees Alice's challenge, while for the others it sees Alice's challenge first. The verification of the partial reveals of $\mathsf{m}_1, \ldots, \mathsf{m}_M$ is performed outside the voting booth, by any computer. (Any improper ballots – those that do not have *exactly* one c_i which is a 1 encryption – will always be detected at tabulation time.)

Preventing Covert Channels in the Ballot. We provide a technique for preventing the voting machine from abusing valid ballot ciphertexts, which are displayed for all to see on the bulletin board, to subliminally communicate information about the voter's choice. In particular, our technique addresses one of the main threats described in Karlof et al. 's systems-perspective analysis of cryptographic voting systems [10]. This technique applies to MarkPledge and MarkPledge2, as it does not depend on the specific bit encryption scheme. Like Benaloh and Tuinstra [3], we assume, in addition to the private channel between the voting device and the voter, a one-time one-way private channel

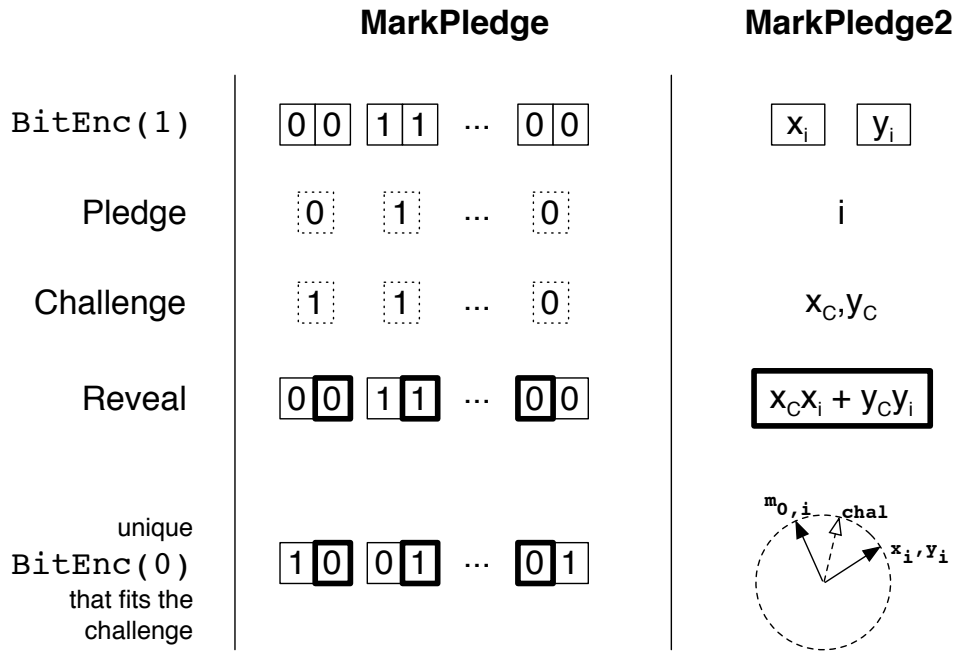

Figure 1: Bit Encryption in MarkPledge and MarkPledge2: (1) messages are first represented as a set of Exponential El Gamal plaintexts: κ pairs for MarkPledge, and just two for MarkPledge2, then each plaintext is encrypted (encryption is represented by a thin box around a plaintext). (2) The prover provides a pledge, which is effectively an indicator of the message representation he chose: the selection of each pair's bit for MarkPledge, and the selected 1-vector for MarkPledge2. (3) The verifier provides a challenge: a κ-bit string for MarkPledge, and an index (up to κ) into the set of test vectors for MarkPledge2. (4) The prover responds by revealing the randomization factor for a portion of the ciphertexts he produced (the thick-boxed elements in the diagram): for MarkPledge, he reveals one element of each pair according to the corresponding bit from the challenge string, and for MarkPledge two the single dot-product of the encrypted 1-vector and test vector, which can be computed homomorphically. (5) Given a pledge string and a challenge, there is only one possible message representation of the 0-bit that would successfully allow the prover to cheat this protocol, thus with probability $2^{-\kappa}$.

from the election trustees to the voting device. This simply models the installation of election-specific data onto the voting machine before the start of election day.

For a given pre-computed ballot with M candidates, all of the bit encryptions $(\mathsf{c}_1, \ldots, \mathsf{c}_M)$ are generated before election day via multi-party computation among the election trustees. The voting machine contains only the ballot identifiers and the corresponding plaintext representations $(\mathsf{m}_1, \ldots, \mathsf{m}_M)$ such that m_{offset} encodes 1 for a single random $offset$, and m_i encodes 0 for $i \neq offset$. The voter's choice is then an index that rotates the ciphertexts to appropriately match the 1 bit-encryption with the selected candidate, a value which obviously depends both on a, the selected candidate and $offset$ the position of the 1-bit in the pre-computed plaintexts. The voting machine's output is highly constrained: if it produces a ballot that passes the trustees' verification, all of its outputs are deterministic functions of the ballot identifier, Alice's candidate choice a, and Alice's challenge. Thus, there is "no room" for the voting machine to encode any information covertly within the ballot.

All partial reveals are now performed by the election trustees with results posted to the bulletin board, which leaves the voting machine with only simple plaintext operations. Simplifying the voting machine equipment is advantageous to the operations and economics of elections, so this change is an important practical benefit as well.

1.4 Previous & Related Work

Humans and Basic Cryptography. In 1994, Naor and Shamir introduced visual cryptography [21], a secret-sharing method that uses visual superposition as a kind of human XOR. A number of improvements were proposed to improve contrast and usability [12, 13]. In 1997, Naor and Pinkas introduced visual authentication [20], a technique to help humans authenticate computers using visual cryptography methods. Chaum used visual cryptography to build one of the first encrypted voter receipt techniques in 2004 [8, 9].

Humans and Interactive Proofs. The literature is rich with protocols that allow humans to securely authenticate to computer systems. Hopper and Blum [16] were the first to use the Learning Parity with Noise (LPN) problem as a basis for human authentication, based on Blum et. al.'s earlier LPN-related work [4]. Interestingly, the low-computational-power constraint now applies to certain computing environments, like RFIDs, which led Juels and Weis to define an LPN-based authentication system for RFIDs [17]. A plethora of works deal with a human proving to a computer that he is, indeed, human. A large number of such protocols, often called CAPTCHAs (Completely Automated Public Turing Test to Tell Computers and Humans Apart), use hard Artificial Intelligence problems to exploit tasks that only humans can do efficiently, e.g. reading garbled text. These were first formalized by von Ahn et. al. [28, 29]. By contrast, ballot casting schemes like MarkPledge2 are human interactive proofs where a human being is *the verifier*.

Receipt-Free Elections. Benaloh and Tuinstra first introduced and implemented receipt-free voting [3]. Enhancements and variations were quickly proposed [27, 23, 24], under the assumption of a private channel between the administrator and voter. Canetti and Gennaro showed how to accomplish incoercible multiparty computation without this assumption [7]. Moran and Naor [19] extended Canetti and Gennaro's model. The voting literature is plentiful: a large number of receipt-free schemes have been proposed [11, 15, 2], all of which assume a computationally capable voter. Some schemes even achieve optimal ballot secrecy [18, 19]. Our proposal, like MarkPledge and the Naor-Moran scheme, provides receipt-freeness without fully trusted hardware. All three schemes require a private channel between the voting machine and the voter.

1.5 Organization

In Section 2, we review notation and number theory. In Section 3, we define the model, soundness and receipt-freeness. We present MarkPledge2 in Section 4 and covert-channel resistance in Section 5.

2 Preliminaries

2.1 Notation and Terminology

We denote PPT and PT^* the class of probabilistic polynomial-time and non-uniform polynomial time Turing machines. All algorithms and parties are denoted using caligraphic letters, e.g. $\mathcal{A}$. All vectors are denoted using bold font, e.g. $\boldsymbol{u}$ or $\boldsymbol{v}$. A protocol run between parties $\mathcal{A}$ and $\mathcal{B}$ with inputs a and b respectively and output out is denoted $out = \langle \mathcal{A}(a), \mathcal{B}(b) \rangle$. When we use the term "efficient," we mean "efficient from a practical standpoint," a far stricter constraint than polynomial time.

2.2 Number Theory of $SO(2, q)$.

Background. Recall that an orthogonal matrix is a square matrix whose transpose is its inverse. $O(n, F)$ is then the group of $n \times n$ orthogonal matrices with matrix elements in field F, and matrix multiplication the group operation. $O(n, q)$ with q a prime refers to the group of $n \times n$ orthogonal matrices with matrix elements in GF_q. Because a matrix's transpose is its inverse, all have determinant 1 or -1. $SO(n, q)$, the *special orthogonal group*, is thus defined as the subgroup of $O(n, q)$ of matrices with determinant 1.

Geometric Interpretation. In this work, we consider specifically $SO(2, q)$, whose elements are of the form:

$$\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$$

with $\alpha^2 + \beta^2 = 1$. $SO(2, q)$ is a cyclic group with order a multiple of 4, which we thus denote 4Λ. Elements of $SO(2, q)$ can be represented as 2-vectors by associating each matrix with its first row. Thus, we consider elements $\boldsymbol{u}$ of the form (α, β). We interpret elements in $SO(2, q)$ as rotations, and the group's operation as rotation composition. Considering vector operations in $\mathbf{Z}_q^2$ with generator $\boldsymbol{\gamma}$, multiplicative notation in $SO(2, q)$, and $\cdot$ the vector dot-product of the 2-vector representations of elements of $SO(2, q)$, for all integer values of i and k:

$$\left(\boldsymbol{\gamma}^{i+2k} - \boldsymbol{\gamma}^{i}\right) \cdot \boldsymbol{\gamma}^{i+k} = 0$$

Given the geometric interpretation, this is to be expected, as $\boldsymbol{\gamma}^{i+k}$ bisects $\boldsymbol{\gamma}^{i}$ and $\boldsymbol{\gamma}^{i+2k}$.

Lemma 1 *Given $\boldsymbol{t} \in SO(2,q)$, $\alpha_0 \in \mathbf{Z}_q$, there are exactly two elements $\boldsymbol{u}, \boldsymbol{v} \in SO(2,q)$, such that $\boldsymbol{u}\cdot\boldsymbol{t} = \boldsymbol{v}\cdot\boldsymbol{t} = \alpha_0$.*

The details, including geometric interpretation, order, and lemma proof, can be found in Appendix D.

3 Model

Our model is similar to Moran and Naor's. We assume a typical, precinct-based, supervised voting setting, with one election race with M candidates (and no write-ins). As in most election schemes, we assume the existence of an *authenticated bulletin board* where any authorized party can post messages, and any observer can read the posted messages in synchronized order. We assume only one voting machine. We note that this is only meant to simplify exposition, whereas the Moran-Naor scheme actually requires this constraint because of its tabulation phase.

In this paper, we focus on the ballot casting stage: we assume the existence of *a robust universally-verifiable mixnet* [25, 27, 22] that shuffles and decrypts cast ballots available on the bulletin board, then proves it did so correctly. We also assume the existence of at least one honest verifier (e.g. Jimmy Carter and his election watchers). We do not use the Universal Composability model [6], because our focus is on efficient techniques. Instead, we prove security using a standalone definition of soundness. We do, however, consider receipt freeness in the model of Moran and Naor.

Parties. We identify the following parties:

- **Voters:** there are N voters $\mathcal{V}_1, \ldots, \mathcal{V}_N$, each with a chosen candidate index $a \in [1, M]$.
- **Voting Machine:** the voting machine $\mathcal{M}$ is a computer inside the voting booth with a touch-screen input and a printer. The voting machine can post to the bulletin board at the end of election day.
- **Trustees:** there are K trustees denoted $\mathcal{T}_1, \ldots, \mathcal{T}_K$. Each can post to the bulletin board.
- **Verifiers:** there are many verifiers, as anyone in the public can be a verifier. A verifier can read the bulletin board and complain if he detects an error. As these verifiers are effectively *helpers* where the voter is concerned, we denote any honest helper $\mathcal{H}$.

Human Capabilities. We assume a voter has the ability to type, read, and compare κ-bit strings, where κ is, in practice, at most 20. These strings may be represented alphanumerically or using some other clever technique, e.g. icons. We also assume a voter can provide a short random input of this same size κ.

Physical Commitment. As in MarkPledge and the Moran-Naor scheme, we assume that the voting machine can perform a physical commitment where Alice, the voter, receives no information about the value of the commitment. This can be achieved using a printer with a partial shield, so that Alice can tell that lines of text have been printed, but not *what* has been printed. This is crucial to ensure that Alice cannot base her random challenge on the value of the commitment.

Soundness. For soundness, we assume every trustee, every voter other than Alice, and the voting machine may all be adversarial. Still, if Alice's ballot is encrypted incorrectly, we want either Alice or a Helper to catch the problem with reasonable probability. More formally, given an honest voter $\mathcal{V}$ selecting candidate a and an honest helper $\mathcal{H}$, for any corrupt set of trustees $\mathcal{T}_1^*, \ldots, \mathcal{T}_K^* \in \mathrm{PT}^*$, any corrupt voting machine $\mathcal{M}^* \in \mathrm{PT}^*$:

$$Pr\Big[(sk, pk) = \langle \mathcal{T}_1^*, \ldots, \mathcal{T}_K^* \rangle; (\mathsf{receipt}, b_{\mathcal{V}}) = \langle \mathcal{M}^*(pk), \mathcal{V}(a) \rangle ; \\ b_{\mathcal{H}} = \mathcal{H}(pk, \mathsf{receipt}) : \\ \mathsf{Dec}_{sk}(\mathsf{receipt}) \neq a \wedge b_{\mathcal{V}} = 0 \wedge b_{\mathcal{H}} = 0\Big] \leq \frac{1}{2}$$

taken over the coin flips of the voter $\mathcal{V}$. The Helper $\mathcal{H}$ is expected to check that the system's public key pk is "good". In a real protocol, the secret key sk does not exist in a single location: decryption can only occur using a distributed protocol among the trustees. For human-verifiable schemes, one should strive to reduce the soundness error to far less than $1/2$. In MarkPledge2, as in MarkPledge and the Moran-Naor scheme, we achieve the best possible soundness given a human's ability to compare κ-bit strings: $2^{-\kappa}$.

Receipt-Freeness. For receipt-freeness, we assume at least one trustee is honest (which is already required for the robust mixnet's privacy). We assume the voting machine has no communication channel with the outside world other than the ballots and receipts it produces. This last assumption implies some enforcement of practical constraints: the voting machine must be simple, shielded, un-networked, and easily resettable at the end of election day.

We use the Moran-Naor model, which extends the Canetti-Gennaro incoercibility model. The real-world protocol is compared to an ideal world, and the protocol is declared receipt-free if any coercion outcome in the real world can be recreated indistinguishably in the ideal world model with the same coercion attempts by the adversary and coercion response by the voters. We review this model in detail in Appendix A.

4 The MarkPledge2 Scheme

We seek a plaintext representation for a bit encryption scheme that allows for partial reveals of the plaintext, down to exactly one possible representation of 0 and one possible representation of 1. Recall that $SO(2, q)$ is a group of order 4Λ, where group elements are vectors of norm 1 representing rotations, and the group operation is effectively rotation composition. We partition the group into representations of 1, representations of 0, and test elements, such that any pair of a 1-vector and a 0-vector is bisected by a test vector on either side. The dot product of either the 1 or 0 vector with the test vector will be the same. Note how a quarter of the elements are 1-vectors, another quarter are 0-vectors, and the remaining half are test vectors. Each test vector $\boldsymbol{t}$ can be paired up with its $\mathbf{Z}_q^2$ inverse $-\boldsymbol{t}$, another test vector, so that in the end there are Λ 1-vectors, Λ 0-vectors, and Λ test vectors.

Then, a bit encryption of a 0 or a 1 is the pair of Exponential El Gamal encryptions of a corresponding vector's coordinates. A pledge is a 1-vector, a challenge is a test vector, and a partial reveal is the dot-product of the plaintext 1-vector and the challenge vector, an encryption of which can be computed homomorphically from the encrypted 1-vector and the plaintext challenge vector. Thus, given a test vector and a dot-product value, as per Lemma 1, there are exactly two possible values for the encrypted vector, one of which is a 1-vector, and the other a 0-vector. To simulate the reveal for an encrypted 0-vector given a challenge, one uses as a pledge the unique 1-vector whose dot-product with the given test vector is the same as that with the encrypted 0-vector.

4.1 Bit Encryption

Consider El Gamal in the q-order subgroup of $\mathbf{Z}_p^*$, where $q|(p-1)$. We will, in particular, make use of Exponential El Gamal, like the original MarkPledge, where $\mathsf{Enc}_{pk}(m; r) = (g^r, g^m y^r)$, with x the secret key, and $y = g^x \bmod p$ the public key. We can use Exponential El Gamal efficiently because we will never need to recover a long plaintext m: we will only compare decryptions and check for ciphertexts that decrypt to $m = 0$.

Defining Classes within $\mathbf{SO(2, q)}$. Given q, the order of the El Gamal subgroup, let Γ be a subgroup of $SO(2, q)$ of order 4λ with $\boldsymbol{\gamma}$ a generator. Note that, like the primes p and q, the parameter λ is just another election parameter. It can be any integral divisor of $|SO(2, q)|/4$.[1] Ideally, λ is selected such that a human can easily compare strings taken from a set of size λ. For simplicity, we assume that $\lambda = 2^\kappa$, where κ is the bit length of a human-verifiable string, though a carefully selected alphabet can accommodate any λ. We define:

- $\mathsf{Zero} = \{\, \boldsymbol{\zeta}_i : \boldsymbol{\zeta}_i = \boldsymbol{\gamma}^{4i+1} \,; i \in [0, \lambda[\}$,
- $\mathsf{One} = \{\, \boldsymbol{\vartheta}_i : \boldsymbol{\vartheta}_i = \boldsymbol{\gamma}^{4i-1} \,; i \in [0, \lambda[\}$,
- $\mathsf{Test} = \{\, \boldsymbol{\tau}_i : \boldsymbol{\tau}_i = \boldsymbol{\gamma}^{2i} \,; i \in [0, \lambda[\}$.

As the names imply, we consider any element of Zero a 0-plaintext, and any element of One a 1-plaintext. Test is the challenge domain: challenges from the verifier will be selected from this class.

Relating the Classes. We denote the difference vector $\boldsymbol{\delta}_{k\,c} = \boldsymbol{\vartheta}_k - \boldsymbol{\zeta}_{c-k}$, using standard vector subtraction in $\mathbf{Z}_q^2$, and we note that, in the geometric interpretation, $\boldsymbol{\tau}_c = \boldsymbol{\gamma}^{2c}$ effectively bisects $\boldsymbol{\vartheta}_k = \boldsymbol{\gamma}^{4k-1}$ and $\boldsymbol{\zeta}_{c-k} = \boldsymbol{\gamma}^{4c-4k+1} = \boldsymbol{\gamma}^{2c+(2c-(4k-1))}$. The geometric interpretation is found in Figure 2.
Thus, for a given element $\boldsymbol{t} \in \mathsf{Test}$ and a given $\boldsymbol{u} \in \mathsf{Zero}$, there is exactly one element $\boldsymbol{v} \in \mathsf{One}$ such that $\boldsymbol{u} \cdot \boldsymbol{t} = \boldsymbol{v} \cdot \boldsymbol{t}$ (and no other element of Zero with the same dot product). This follows from Lemma 1 and the careful choice of indices for the classes One, Zero, and Test.

[1] That is, an integral divisor of $(q+1)/4$ or $(q-1)/4$ depending on whether $q \equiv 3$ or $q \equiv 1 \bmod 4$ respectively. See Appendix D.

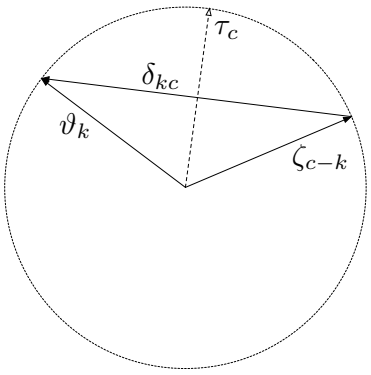

Figure 2: The geometric interpretation of the interplay between elements of One, Zero, and Test. Given $\vartheta_k \in$ One and $\tau_c \in$ Test, there is exactly one element of Zero, ζ_{c-k}, such that the difference vector $\delta_{k\,c}$ is orthogonal to τ_c.

Vector Encryption. We can perform component-wise encryption of vectors in $\mathbf{Z}_q^2$. Using Exponential El-Gamal, we gain the ability to perform homomorphic vector addition of two encrypted vectors, and homomorphic vector dot-product of one encrypted vector with a plaintext vector. We abuse the single-value notation to represent vector encryption:

$$\mathsf{Enc}_{pk}(\boldsymbol{s};\boldsymbol{r}) = \Big(\mathsf{Enc}_{pk}(\boldsymbol{s}[0];\boldsymbol{r}[0]), \mathsf{Enc}_{pk}(\boldsymbol{s}[1];\boldsymbol{r}[1])\Big)$$

Note that, with vector components in exponential El Gamal ciphertext, homomorphic vector addition is trivial using component-wise homomorphic multiplication.

In addition, given $\boldsymbol{Q} = \mathsf{Enc}_{pk}(\boldsymbol{s};\boldsymbol{r})$, we can homomorphically compute an encryption of the dot product $\boldsymbol{s}\cdot\boldsymbol{t}$ where $\boldsymbol{t}$ is a plaintext element of $\mathbf{Z}_q^2$. Here, multiplicative notation denotes homomorphic multiplication (component-wise exponentiation of ciphertexts), while additive notation denotes homomorphic addition (component-wise multiplication of ciphertexts):

$$\boldsymbol{Q}\cdot\boldsymbol{t} = (\boldsymbol{Q}[0]\boldsymbol{t}[0] + \boldsymbol{Q}[1]\boldsymbol{t}[1]) = \mathsf{Enc}_{pk}(\boldsymbol{s}\cdot\boldsymbol{t};\boldsymbol{r}\cdot\boldsymbol{t})$$

The result is a single ciphertext, which is to be expected when performing a vector dot-product. Notice also how the randomization value in the resulting ciphertext is also the result of a dot-product, given that the homomorphic operation actually performs an *exponentiation* of the El Gamal ciphertexts.

Bit Encryption. Let b be the bit we wish to encrypt. We define an $SO(2,q)$-based method of bit encryption. First, pick a random vector $\boldsymbol{\gamma}^{4r'+2b+1}$ using randomness $r' \in \mathbf{Z}_\lambda$. This vector will be in Zero if $b = 0$, and in One if $b = 1$. Then, perform encryption of this vector using randomness $\boldsymbol{r} \in \mathbf{Z}_q^2$:

$$\mathsf{BitEnc}_{pk}(b,(\boldsymbol{r},r')) = \mathsf{Enc}_{pk}(\boldsymbol{\gamma}^{4r'+2b+1};\boldsymbol{r})$$

In other words, to perform a bit encryption, we first select a message representation of the bit b as an element of $SO(2,q)$, which is a 2-vector with components in $\mathbf{Z}_q^2$, then we perform the element-wise Exponential El-Gamal encryption of the vector's coordinates. The result is a vector of two ciphertexts.

4.2 The MarkPledge2 Protocol

The MarkPledge2 protocol presents the exact same interface as MarkPledge [5], except BitEnc is now performed using the bit-encryption mechanism just described, and the proof algebra is adjusted accordingly.

MarkPledge2 Ballot Casting Protocol

1. Alice enters the booth, where she interacts with the voting machine to select candidate $a \in [1, M]$.
2. The voting machine prepares

(a) $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_M$, randomization vectors in $\mathbf{Z}_q^2$

(b) $k_a \xleftarrow{R} [1, \lambda[$, a random index into One and the associated element $\boldsymbol{\vartheta}_{k_a}$. $\boldsymbol{\vartheta}_{k_a}$ is effectively a randomly selected "1-vector" which will be used when producing the bit encryption of b_a,

(c) $(l_i)_{i \in [1,M], i \neq a}$, indexes into Zero, and the associated elements $(\boldsymbol{\zeta}_{l_i})$,

(d) $\mathsf{c}_a = \boldsymbol{Q}_a = \mathsf{Enc}_{pk}(\boldsymbol{\vartheta}_{k_a}; \boldsymbol{r}_a)$, for the chosen index a,

(e) $\mathsf{c}_i = \boldsymbol{Q}_i = \mathsf{Enc}_{pk}(\boldsymbol{\zeta}_{l_i}; \boldsymbol{r}_i)$, for all $i \in [1, M], i \neq a$,

then physically commits on the receipt to the values $(\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_M)$. These values, which are the bit encryptions of a 1 for the selected option and a 0 for all others, are committed using the printer shield so that Alice knows they have been committed but cannot see their values.

3. For each candidate index $i \neq a$, Alice is prompted for a short challenge chal_i. Each such challenge is an index into the class Test, thus corresponding to a specific test vector $\boldsymbol{\tau}_{\mathsf{chal}_i}$.

4. The voting machine computes $k_i = \mathsf{chal}_i - l_i$, for $i \neq a$. These are the indexes of elements of One, one for each candidate where the voting machine actually encrypted an element of Zero.

   The voting machine then physically commits to (i.e. prints under the shield) the values $(k_1, \ldots, k_M)$. Note that k_a is being committed to before Alice has entered a challenge for the corresponding bit encryption of her selected option a.

5. Alice enters chal_a, which corresponds to $\boldsymbol{\tau}_{\mathsf{chal}_a}$.

   At this point, the previously physically committed data can be revealed.

6. The voting machine then computes

   (a) $l_a = \mathsf{chal}_a - k_a$, an element of Zero for the chosen index a. This is the corresponding element of Zero for the one selected option a, now that the prover knows chal_a.

   (b) $(\boldsymbol{\delta}_{k_1\, \mathsf{chal}_1}, \ldots, \boldsymbol{\delta}_{k_M\, \mathsf{chal}_M})$, the difference vectors between $\boldsymbol{\zeta}_{l_i}$ and $\boldsymbol{\vartheta}_{k_i}$. Note that $\boldsymbol{\delta}_{k_i\, \mathsf{chal}_i}$ is orthogonal to $\boldsymbol{\tau}_{\mathsf{chal}_i}$ (as per Lemma 1).

   (c) $\rho_i = \boldsymbol{r}_i \cdot \boldsymbol{\tau}_{\mathsf{chal}_i}$, for all $i \in [1, M]$.
   This is effectively the revealed random factors of the encrypted dot products.

   and prints to the receipt $(\rho_1, \ldots, \rho_M)$ and $(\mathsf{chal}_1, \ldots, \mathsf{chal}_M)$, which now contains:

$$\mathsf{receipt} = \Big(\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_M, k_1, \ldots, k_M, \rho_1, \ldots, \rho_M, \mathsf{chal}_1, \ldots, \mathsf{chal}_M\Big)$$

7. Alice accepts the receipt if she confirms that her challenges $(\mathsf{chal}_1, \ldots, \mathsf{chal}_M)$ are properly printed on the receipt next to their designated candidate.

8. Once outside the booth, Alice further verifies her receipt with the help of a program:

   (a) homomorphically compute $\boldsymbol{Q}'_i = \boldsymbol{Q}_i - \boldsymbol{\zeta}_{\mathsf{chal}_i - k_i}$, for all $i \in [1, M]$.
   Thus, $\boldsymbol{Q}'_i$ is either the null vector (if $i \neq a$), or the difference vector $\boldsymbol{\delta}_{k_i\, \mathsf{chal}_i}$ (if $i = a$). In either case, $\boldsymbol{Q}'_i$ is orthogonal to $\boldsymbol{\tau}_{\mathsf{chal}_i}$. Because $\boldsymbol{\zeta}_{\mathsf{chal}_i - k_i}$ is a plaintext vector, this homomorphic subtraction can be performed so that $\boldsymbol{Q}'_i$ keeps the same randomization as $\boldsymbol{Q}_i$, which is $\boldsymbol{r}_i$.

   (b) homomorphically compute $\boldsymbol{Q}'_i \cdot \boldsymbol{\tau}_{\mathsf{chal}_i}$ and check that it is equal to $\mathsf{Enc}_{pk}(0; \rho_i)$.
   This verifies that $\boldsymbol{Q}'_i$ and $\boldsymbol{\tau}_{\mathsf{chal}_i}$ are orthogonal. Thus, $\boldsymbol{Q}_i$ must be the encryption of $\boldsymbol{\vartheta}_{k_i}$ or $\boldsymbol{\zeta}_{l_i}$.

Ballot Canonicalization. It is important to notice that even though encrypted ballots will be passed through a mixnet, ballots will eventually be decrypted: any uniqueness of the underlying plaintext could be used to identify the voter and violate her ballot secrecy. Thus, before ballots enter the mixnet, we propose to canonicalize them. All encrypted 1-vectors should be transformed into the canonical 1-vector $\boldsymbol{\gamma}^{-1}$, and all encrypted 0-vectors into the canonical 0-vector $\boldsymbol{\gamma}$. Given that the bit encryptions are partially revealed to the public, each ciphertext $\boldsymbol{Q}_i$ is publicly known to be the encryption of either the 0-vector $\boldsymbol{\zeta}_{l_i}$ or the 1-vector $\boldsymbol{\vartheta}_{k_i}$. It is then trivial to publicly determine the plaintext linear transformation that transforms $\boldsymbol{\zeta}_{l_i}$ into $\boldsymbol{\gamma}$ and $\boldsymbol{\vartheta}_{k_i}$ into $\boldsymbol{\gamma}^{-1}$, and to publicly apply this transformation homomorphically to $\boldsymbol{Q}_i$. The mixnet output will then reveal nothing more than the specific candidate choice, with all randomness, and thus all uniqueness, removed.

4.3 Soundness and Receipt-Freeness

Intuitively, MarkPledge2 is sound because the voting machine can only cheat if it guesses the voter's challenge test vector ahead of time. It has a $2^{-\kappa}$ chance of doing so correctly. We prove this in detail in Appendix B. Intuitively, MarkPledge2 is receipt-free because Alice cannot take any action that depends on private information she sees in the booth *and* that affects her vote. We prove this in detail in Appendix C.

We note, however, that a corrupt voting machine could leak Alice's vote by cleverly selecting the randomization factors that affect the ciphertexts in the ballot and thus on the bulletin board. Thus, for now, we must assume that the voting machine is honest for receipt-freeness. Next, we show how to reduce this requirement: we only require that the voting machine have no *other* communication channel than the ballots.

4.4 Practical Optimization

It may be too much of a burden for Alice to generate and remember a different challenge for each candidate. Just like MarkPledge [1], MarkPledge2 can be adapted so that Alice need only enter a single challenge that can be used for the real proof and all of the simulated proofs. In this case, the voting machine commits to its real pledge string on screen before Alice enters her challenge, and Alice verifies only two values: the pledge string for her candidate, and the single challenge string she entered. This tweak presents two major benefits: Alice can now ignore all data regarding candidates she rejected, and the special shielded printer is no longer necessary. One complication arises: Alice's challenge cannot depend on the pledge string, as this would clearly make the protocol coercible. Thus, when this optimization is used, Alice must commit to her challenge ahead of time, e.g. when she signs in to vote.

5 Resistance to Covert Channels

In the currently described scheme, as well as in MarkPledge and the Moran-Naor scheme, the voting machine can easily select ciphertexts so as to reveal information about a, Alice's chosen candidate, e.g. using the last few bits of the ciphertexts. We now propose a method to prevent this. We ensure that, in order to discern information about the voter's choice from the receipt data, an adversary must have cooperation from at least a threshold number of Trustees. We present this solution in the context of MarkPledge2, though it is immediately applicable to MarkPledge. It is *not* immediately applicable to the Moran-Naor scheme.

Intuitively, the solution is quite simple: only the first two steps of the three-round proof of bit encryption need to be performed inside the voting booth. The last step, partial decryption, can be performed once Alice has left the booth, by the election trustees rather than the voting machine. Thus, we limit the voting machine's power by pre-generating all of the ciphertexts, leaving the voting machine to deal only with the plaintexts. With all ciphertexts pre-generated, the voting machine no longer has the "wiggle room" to embed any covert information inside the receipt. Our proposal also greatly reduces the complexity of the voting machine itself, which is of great interest to voting officials who seek simple and inexpensive equipment.

5.1 Before the Election

The trustees generate the election public key pk using threshold El Gamal [26], such that each $\mathcal{T}_i$ has a secret key share sk_i. Consider, then, the trivial encryption of a ballot:

$$(\mathsf{c}_1^{(0)}, \ldots, \mathsf{c}_M^{(0)}) = (\mathsf{BitEnc}_{pk}(1, (\mathbf{0}, 0)), \mathsf{BitEnc}_{pk}(0, (\mathbf{0}, 0)), \ldots, \mathsf{BitEnc}_{pk}(0, (\mathbf{0}, 0)))$$

In turn, each trustee $\mathcal{T}_i$ then re-encrypts and shuffles these bit encryptions *for each ballot*. Re-encryption includes modifying both the El Gamal randomization factor and the specific plaintext representation. We abuse notation here to indicate component-wise multiplication:

$$\mathsf{BitReenc}_{pk}(\mathsf{c}, \boldsymbol{r}, r') = \mathsf{c} \times \mathsf{Enc}_{pk}(\boldsymbol{\gamma}^{4r'}; \boldsymbol{r})$$

Thus, for $k \in [1, N\beta]$ where N is the number of voters and β is a cut-and-choose factor, and for $l \in [1, M]$ where M is the number of candidates, $\mathcal{T}_i$ generates random values $\boldsymbol{r}_{k,l} \in \mathbf{Z}_q^2, r'_{k,l} \in \mathbf{Z}_\lambda$ and outputs:

$$\mathsf{c}_{k,l}^{(i)} = \mathsf{BitReenc}_{pk}(\mathsf{c}_{k,\pi_k(l)}^{(i-1)}, (\boldsymbol{r}_{k,l}, r'_{k,l}))$$

These values are then posted to the bulletin board for the next trustee to reencrypt and shuffle. To parallelize the process, the ballots can be split into batches, then passed from one trustee to the next in circular order.

To verify that all trustees performed their re-encryption tasks correctly, we perform a cut-and-choose on the final set of ballots, opening up $(\beta - 1)/\beta$ of them, leaving N unrevealed ballots. If any problematic ballot is encountered, the mistake is traced by asking each trustee to reveal how it shuffled and reencrypted that faulty ballot. The guilty parties are removed and the process is restarted. A small-factor β cut-and-choose is sufficient because any incorrect ballot that makes it through the cut-and-choose will be detected at tabulation time. At the end of this setup phase, all ballot ciphertexts to be used are posted on the bulletin board, indexed by unique ballot identifier.

5.2 The Voting Machine

Each trustee $\mathcal{T}_i$ communicates its permutation and reencryptions factors, $\{\pi_k, r'_{k,1}, \ldots, r'_{k,M}\}_{k \in [1,N]}$, to the voting machine via its private one-time one-way channel. Thus, given ballot index k, the voting machine knows the plaintexts of the ciphertexts $(\mathsf{c}_{k,1}^{(K)}, \ldots, \mathsf{c}_{k,M}^{(K)})$, including the *exact* single 1-vector and $(M - 1)$ 0-vectors. When Alice enters the voting booth, her ballot card (usually a smart card inserted into the machine) indicates a ballot identifier. The voting machine performs the MarkPledge2 protocol as expected, using the vector plaintexts to compute the simulated pledge strings, and prints on the receipt a circular offset, modulo M, which indicates how to rotate the sequence $(\mathsf{c}_{k,1}^{(K)}, \ldots, \mathsf{c}_{k,M}^{(K)})$ to capture Alice's chosen candidate. The voting machine *does not* perform the partial reveal step: it can't. It also need not print the full ciphertexts, only the pledge strings, voter challenges, and ballot identifier. Note how the voting machine *only needs plaintexts*. This process is diagrammed in Figure 3.

5.3 After Ballot Casting

Once Alice casts her ballot, each trustee homomorphically computes the difference vectors $(\boldsymbol{Q}'_1, \ldots, \boldsymbol{Q}'_M)$, then the dot-products $(\boldsymbol{Q}'_1 \cdot \boldsymbol{\tau}_{\mathsf{chal}_1}, \ldots, \boldsymbol{Q}'_M \cdot \boldsymbol{\tau}_{\mathsf{chal}_M})$, as per Step 8 of the protocol. Then, instead of revealing the random factors $(\rho_1, \ldots, \rho_M)$ which would be difficult to perform without revealing how the trustees shuffled the bit encryptions, the trustees perform a joint verifiable decryption [26, 14] of this dot-product, proving that the plaintext is 0.

We note that the proof protocol must lend itself to simulation where messages are generated in order 2, 1, then 3, rather than the typical 2, 3, 1 order of special-sound proofs. This requirement exists because step 3 is only performed by election trustees after Alice has left the voting booth, and it is another important property of the proof protocol presented here (as well as that of MarkPledge and Moran-Naor).

5.4 Soundness and Receipt-Freeness

From the voter's point of view, the only difference between this variation and the previous protocol description is in the partial reveal step, which is now performed by threshold verifiable decryption rather than simply revealing the randomization values. Assuming the soundness of this proof is at least as high as the soundness of our human verification (which is trivial to accomplish), the protocol's overall soundness remains the same. Receipt-freeness is also unchanged: from the point of view of the voter and potential coercer, nothing has changed.

We have, however, achieved an interesting additional improvement with this resistance to covert channels. Because the voting machines are now highly constrained and can only output valid ballots using pre-generated ciphertexts, we

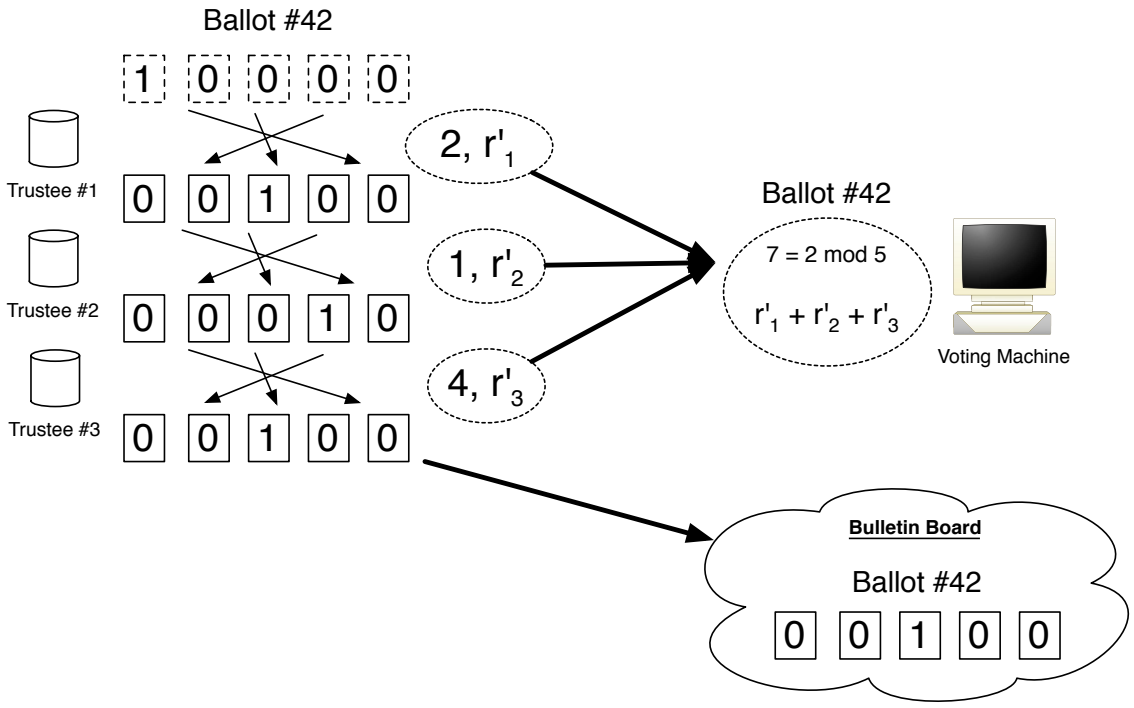

Figure 3: Trustees Generate and Deliver Ballots: Rather than have the voting machine generate ciphertexts on the fly, all ciphertexts are pre-generated by the trustees by sequential, provable shuffling of the ciphertexts within each ballot. This shuffling randomizes the outer encryption shell, but also the message representation within the encryptions, using homomorphic vector operations. (In this diagram, the boxes are bit-encryptions, not just the underlying El Gamal encryptions.) At the start of election day, the trustees deliver to the voting machine just the message representation re-randomization and cyclic shift each performed for each ballot. The voting machine thus knows which ciphertext is the bit-encryption of 1, and the underlying message representation of the plaintexts of the bit encryptions for each ballot. However, the voting machine does *not* know the El Gamal randomization factors. Given the voter's candidate choice, the voting machine can print the offset needed to shift the 1-bit into the right position and can provide the appropriate pledge string for this 1-bit, as well as the simulated pledge strings for the 0-bits, all using only basic operations in $\mathbf{Z}_q^2$, i.e. only plaintext operations in a finite group. Decryption is distributed amongst the trustees after the ballot pledge and reveal strings are posted on the bulletin board.

can show that this system provides receipt-freeness even if the voting machine is malicious: a voting machine that deviates from the protocol and tries to leak information to violate receipt-freeness would be immediately caught, because the only ballot it can produce is a deterministic function of the ballot identifier and the voter's intent. This stands in contrast to the Moran-Naor receipt-freeness proof, which assumes an honest voting machine. We leave to future work the detailed discussion of how to set up the voting process to detect and recover from malicious voting machines. We simply note that receipt-freeness comes more naturally once we have covert-channel resistance.

6 Conclusion & Future Work

We've improved on Neff's MarkPledge and the Moran-Naor ballot casting schemes. Our approach continues to be usable by unaided humans with exactly the same interface as before, but the ballot representation is much shorter, and we show how to ensure that the receipt cannot become a covert channel. By the same token, we greatly simplify the voting machine itself, which is promising for real-world deployment, where the individual cost of such machines is crucial.

It will be interesting to see if our techniques can be adapted to the everlasting privacy setting. Though this would involve computational rather than statistical soundness, it is important to provide multiple technical options, so that policy makers can choose the compromise that is most sensible for their needs.

References

[1] Ben Adida and C. Andrew Neff. Ballot Casting Assurance. In *EVT '06, Proceedings of the First Usenix/ACCURATE Electronic Voting Technology Workshop, August 1st 2006, Vancouver, BC,Canada.*, 2006. Available online at `http://www.usenix.org/events/evt06/tech/`.

[2] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.

[3] Josh Cohen Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *STOC*, pages 544–553, 1994.

[4] Avrim Blum, Merrick Furst, Michael Kearns, and Richard J. Lipton. Cryptographic Primitives Based on Hard Learning Problems. In *Advances in Cryptology – CRYPTO'93*, volume 773, pages 278–291, 1994.

[5] C. Andrew Neff. Practical High Certainty Intent Verification for Encrypted Votes. `http://votehere.net/vhti/documentation/vsv-2.0.3638.pdf`.

[6] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA, Proceedings*, pages 136–145. IEEE Computer Society, 2001. (Full version at Cryptology ePrint Archive, Report 2000/067, `http://eprint.iacr.org`, October, 2001.).

[7] Ran Canetti and Rosario Gennaro. Incoercible multiparty computation (extended abstract). In *FOCS*, pages 504–513, 1996.

[8] David Chaum. Secret-Ballot Receipts: True Voter-Verifiable Elections. *IEEE Security and Privacy*, 02(1):38–47, 2004.

[9] David Chaum, Peter Y. A. Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[10] Chris Karlof and Naveen Sastry and David Wagner. Cryptographic Voting Protocols: A Systems Perspective. In *Fourteenth USENIX Security Symposium (USENIX Security 2005)*, pages 33–50, August 2005.

[11] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Proceedings of EUROCRYPT 1997*. Springer-Verlag, LNCS 1233, 1997.

[12] Stefan Droste. New results on visual cryptography. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 1996.

[13] Quang Viet Duong and Kaoru Kurosawa. Almost ideal contrast visual cryptography with reversing. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 353–365. Springer, 2004.

[14] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 1999.

[15] Martin Hirt and Kazue Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, pages 539–556, 2000.

[16] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In Colin Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66. Springer, 2001.

[17] Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology – CRYPTO 2005*, volume 3621, pages 293–308. Springer, August 2005.

[18] Aggelos Kiayias and Moti Yung. Self-tallying elections and perfect ballot secrecy. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 141–158. Springer, 2002.

[19] Tal Moran and Moni Naor. Receipt-free universally-verifiable voting with everlasting privacy. In Cynthia Dwork (Editor), editor, *CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 373–392. Springer-Verlag, August 2006.

[20] Moni Naor and Benny Pinkas. Visual authentication and identification. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 1997.

[21] Moni Naor and Adi Shamir. Visual cryptography. In *EUROCRYPT*, pages 1–12, 1994.

[22] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[23] Valtteri Niemi and Ari Renvall. How to prevent buying of votes in computer elections. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT*, volume 917 of *Lecture Notes in Computer Science*, pages 164–170. Springer, 1994.

[24] Tatsuaki Okamoto. Receipt-free electronic voting schemes for large scale elections. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 25–35. Springer, 1997.

[25] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient anonymous channel and all/nothing election scheme. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 1994.

[26] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer, 1991.

[27] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995.

[28] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311. Springer, 2003.

[29] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

A The Receipt-Freeness Model

We use the Moran-Naor model. We restate it in full for completeness, but we claim no novelty here.

A.1 The Ideal World

We consider N parties $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_N$ with inputs $x_1, x_2, \ldots, x_N$. A trusted third party collects all inputs, computes $f(x_1, x_2, \ldots, x_N)$, and broadcasts the result. The ideal adversary is denoted $\mathcal{I}$. Each party $\mathcal{V}_i$ also has a coercion-response input c_i which indicates whether the party will comply with a coercion attempt ($c_i = 1$), or will continue to vote as she chooses and pretend to be coerced ($c_i = 0$). The adversary $\mathcal{I}$ can choose to:

- **corrupt a subset $\boldsymbol{V}^*$ of the parties:** $\mathcal{I}$ learns the inputs $(x_i)_{i \in \boldsymbol{V}^*}$, takes control of $\{\mathcal{V}_i\}_{i \in \boldsymbol{V}^*}$, and replaces the inputs $(x_i)_{i \in \boldsymbol{V}^*}$ with $(x'_i)_{i \in \boldsymbol{V}^*}$.
- **coerce a subset $\boldsymbol{V}^c$ of the parties:** $\mathcal{I}$ sends to each of $\{\mathcal{V}_i\}_{i \in \boldsymbol{V}^c}$ a new input x'_i. If $c_i = 1$, then $\mathcal{V}_i$ sends its real input x_i to $\mathcal{I}$ and uses the prescribed x'_i as its new input. If $c_i = 0$, then $\mathcal{V}_i$ ignores x'_i and sends a randomly generated fake input x^c_i to $\mathcal{I}$. One exception exists: if $\mathcal{I}$ submits input $\perp$, which means forced abstention, $\mathcal{V}_i$ uses $\perp$ regardless of its coercion-resistance bit c_i.

$\mathcal{I}$ can choose to adaptively corrupt or coerce parties one at a time, based on what it learns regarding previously coerced parties' original input. Once $\mathcal{I}$ signals to the trusted third party that its setup is complete, the trusted third party collects the inputs, including corrupt inputs $(x'_i)_{i \in \boldsymbol{V}^*}$ and successfully coerced inputs (though the trusted third party doesn't know which are honest, corrupt, or coerced), then computes the corresponding output and broadcasts it. The view of the ideal adversary includes its own random coins, the corrupted parties' original inputs $(x_i)_{i \in \boldsymbol{V}^*}$, the coerced parties' supposed original inputs (some of which may be fake), and the output of the trusted third party. Note that $\mathcal{I}$ does not see the coercion-response bits $c_1, \ldots, c_N$, though it may indirectly learn some of them from the rest of its view.

A.2 The Real World

We consider N parties $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_N$ with inputs $x_1, x_2, \ldots, x_N$ and coercion-resistant inputs $c_1, c_2, \ldots, c_N$. The real-world adversary is denoted $\mathcal{A}$. All parties are probabilistic polynomial time interactive Turing machines, with pairwise communication tapes, synchronous and atomic message exchanges, and no erasures.

Again as in Moran-Naor, each $\mathcal{V}_i$ has a private communication channel with the adversary $\mathcal{A}$ and a special coercion-status register ("honest","corrupt","coerced") that $\mathcal{A}$ can write once. Each party $\mathcal{V}_i$ effectively contains three ITMs with joint tapes, one for each of the honest, coerced-but-resisting, coerced-willfully-or-corrupt behaviors. A specific protocol under consideration defines the honest and coerced-but-resisting behaviors, while the coerced-willfully-or-corrupt behavior is defined generically here:

- **corrupt**: when $\mathcal{A}$ corrupts $\mathcal{V}_i$, $\mathcal{V}_i$ becomes a puppet for $\mathcal{A}$ using the private channel they share. Thus, $\mathcal{A}$ can send any message it wants in $\mathcal{V}_i$'s place, and can request any past view or portion of $\mathcal{V}_i$'s tape.
- **coerced willfully**: when $\mathcal{A}$ sets the coercion flag for $\mathcal{V}_i$ and $\mathcal{V}_i$'s coercion-resistance bit is set to comply, $\mathcal{V}_i$ sends x_i to $\mathcal{A}$ and executes the puppet strategy, as if it were corrupt.

$\mathcal{A}$ can corrupt or coerce any party at any time, adaptively. $\mathcal{A}$'s view of the protocol consists of its random coins, the complete views of the corrupted parties, and the communication between $\mathcal{A}$ and the coerced parties.

A.3 Receipt Freeness

Definition 1 *A protocol is receipt-free if, for every real adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{I}$, such that, for every input vector $x_1, x_2, \ldots, x_N$ and any coercion-response vector $c_1, c_2, \ldots, c_N$:*

1. *$\mathcal{I}$'s output in the ideal world is indistinguishable from $\mathcal{A}$'s output in the real world with the same input and coercion-resistance vectors, with distributions taken over the random coins of $\mathcal{I}$, $\mathcal{A}$, and the parties $\mathcal{V}_i$.*
2. *$\mathcal{I}$ corrupts (or coerces) exactly the same parties as $\mathcal{A}$ (respectively).*

B Proofs of Soundness and Receipt-Freeness

We now provide the proofs of soundness and receipt-freeness that did not fit in the main body of the paper. Recall our definition of soundness, with malicious trustees $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots, \mathcal{T}_K^*$, malicious voting machine $\mathcal{M}^*$, and an honest voter $\mathcal{V}$:

$$Pr\Big[(sk, pk) = \langle \mathcal{T}_1^*, \ldots, \mathcal{T}_K^* \rangle; (\mathsf{receipt}, b_{\mathcal{V}}) = \langle \mathcal{M}^*(pk), \mathcal{V}(a) \rangle : \\ b_{\mathcal{H}} = \mathcal{H}(pk, \mathsf{receipt}) : \\ \mathsf{Dec}_{sk}(\mathsf{receipt}) \neq a \wedge b_{\mathcal{V}} = 0 \wedge b_{\mathcal{H}} = 0\Big] \leq \frac{1}{2}$$

In MarkPledge2, as in MarkPledge, we assume that all ballots produced by an honest or malicious voting machine are at least valid: they contain exactly one bit encryption of 1 and $M-1$ bit encryptions of 0. We can safely make this assumption because ballots are eventually decrypted, and any "bad ballot" can simply be traced back to the original voting machine that produced it. Our proof of soundness focuses on the voting machine producing a ballot that *matches the voter's intent*.

Theorem 1 *Assuming a verifiable threshold encryption scheme, a voting machine that only produces valid ballots, a voter with the ability to read, enter, and compare κ-bit strings, MarkPledge2 is a sound ballot casting algorithm with soundness $1 - 2^{-\kappa}$.*

Proof. We prove this by contradiction. Assume a bad voting machine $\mathcal{M}^*$ that manages to produce a ballot for a candidate other than the voter's selected option a, without either the Voter $\mathcal{V}$ or the Helper $\mathcal{H}$ noticing. To achieve this, $\mathcal{M}^*$ might fool $\mathcal{V}$ or $\mathcal{H}$ (or both, but one is enough).

1. **Fooling the Helper**: $\mathcal{M}^*$ can try to fool the Helper $\mathcal{H}$ by incorrectly revealing the pledge strings. Before the distributed generation of the ballot, the reveal is performed via the randomization factors, which cannot be faked with the El Gamal cryptosystem (or any other cryptosystem with unique decryption). When the distributed ballot generation improvement is used, the proof of decryption using threshold El Gamal has soundness error negligible in the cryptosystem security parameter: soundness is much higher than $1 - 2^{-\kappa}$, since κ is smaller than any reasonable cryptographic security parameter. Thus, the Helper $\mathcal{H}$ is fooled with negligible probability.
2. **Fooling the Voter**: Because we assume a well-formed ballot, and that $\mathcal{V}$ checks every step of the protocol correctly—in particular that the ciphertexts $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_M$ and the pledges $k_1, \ldots, k_M$ are printed on the receipt before the challenges are issued— $\mathcal{M}^*$ can only fool the voter by encrypting a 0-vector as $\boldsymbol{Q}_a$, so that its 1-vector is used for a different candidate $a', a' \neq a$.

   Assuming this happens, and since pledges are revealed correctly when the Helper is not fooled, there is only one possible challenge that will let $\mathcal{M}^*$ succeed: the test vector that bisects the pledged 1-vector and encrypted 0-vector. Assuming the voter selects her challenges randomly and enters the one for her chosen candidate only after the pledge strings are printed, $\mathcal{M}^*$ has a $2^{-\kappa}$ chance of receiving the right challenge that will let it cheat.

Thus, the easiest way for $\mathcal{M}^*$ to cheat is to cheat the human verifier, and the best chance it has to do so is $2^{-\kappa}$. Soundness is thus $1 - 2^{-\kappa}$.

□

C Receipt Freeness

Theorem 2 *Assuming an ideal mixnet functionality and semantic security of the El Gamal cryptosystem, MarkPledge2 is receipt-free.*

Proof. To prove receipt freeness, we first describe the voter's coercion-resistance strategy, which details how a voter can double-cross a coercer indistinguishably. We then construct an ideal-model adversary that can output something indistinguishable from any real-world advesary, not knowing which parties will choose the coercion-resistance strategy. The rest of this section provides all relevant details. □

C.1 Coercion-Resistance Strategy

The voter $\mathcal{V}_i$'s only interactions with the voting machine are:

1. the selection of a candidate, before which the machine has produced no confidential information.
2. the generation and entry of a challenge, after the voting machine has committed to its pledge strings in a physical way, which means the voter cannot see these ciphertexts.

Thus, $\mathcal{V}_i$'s coercion-resistance strategy is very simple: she behaves exactly as a puppet would, except that she continues to use her preferred candidate x_i in her interactions with the voting machine, responding to the adversary with the responses a puppet would give. Specifically:

- if the adversary instructs $\mathcal{V}_i$ to use a specific set of challenges, $\mathcal{V}_i$ complies. Note that these challenges cannot be based on either the ciphertexts $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_M$, or the pledge string k_a, which are committed to physically such that the voter must enter her challenges before she sees them. (We actually do not need to physically commit to the ciphertexts, only to the chosen-candidate pledge string, as a challenge based on the ciphertexts does not result in coercibility. However, for simplicity's sake, we assume everything before the voter enters her challenges is physically committed.)

- if the adversary instructs $\mathcal{V}_i$ to vote for a specific candidate x_i', $\mathcal{V}_i$ ignores this command when dealing with the voting machine, but acts as if she had followed it when interacting with the adversary. Since $\mathcal{V}_i$'s only actions are the entry of her candidate choice x_i and the entry of the challenges, this is trivial even for a human to fake.

C.2 Proof of Receipt Freeness

To abstract out the operation of the robust mixnet, we consider a hybrid situation for our real-world setup, where the voting machine broadcasts all encrypted ballots to all parties including the adversary, then submits all encrypted ballots to a decryption mixnet functionality, which shuffles, decrypts, and broadcasts the plaintext results to all parties, including the adversary. (The broadcast of encrypted ballots is meant to mimick the effects of a bulletin board.) The broadcast of all encrypted ballots and the shuffling does not begin until the adversary $\mathcal{A}$ allows it to. Though we do not care to consider the mixnet's inner workings, it remains important to consider the output of this mixnet as part of the adversary's view, in case such an output provides the adversary with some insight that makes the protocol coercible. We also assume a public-key setting where both $\mathcal{I}$ and $\mathcal{A}$ have access to pk, but not sk.

We construct an ideal adversary $\mathcal{I}$. $\mathcal{I}$ uses the real-world adversarial strategy $\mathcal{A}$ as a black box, emulating to it the rest of the protocol. ($\mathcal{I}$ also emulates the abstract mixnet functionality.) The key intuition is that, when $\mathcal{I}$ emulates the rest of the protocol, it knows which parties $\mathcal{A}$ tries to coerce, but it doesn't know which will choose the coercion-resistance response. Thus, we show how $\mathcal{I}$ can forward coercion-and-corruption messages appropriately to the ideal-model parties and fill in the real-world portions of the protocol. Eventually, $\mathcal{I}$ emulates the mixnet functionality so as to correct for any mistake it made in assuming that all coerced voters would become puppets. This "false mixing" is an acceptable emulation because we assume an ideal mixnet functionality.

Emulating the Corrupt and Coerced Parties. When $\mathcal{A}$ sends the command to coerce or corrupt a party $\mathcal{V}_i$, $\mathcal{I}$ instantiates a puppet strategy $\mathcal{V}_i^c$ to talk to $\mathcal{A}$, for which it will emulate the voting machine $\mathcal{M}$. $\mathcal{I}$ creates this puppet strategy as a complete ITM with a random secret input x_i and a consistent history. Notably, $\mathcal{I}$ does *not* immediately send a coercion/corruption message to the corresponding ideal party. Instead, it waits until $\mathcal{A}$ sends the actual candidate touch-screen selection to the puppet according to our specific MarkPledge2 protocol. At this point, $\mathcal{I}$ sends the appropriate "corrupt" or "coerce" message to the corresponding ideal-world party along with the new input x_i' prescribed by $\mathcal{A}$ in its order to the puppet. Note how, in the case of a corrupt party, $\mathcal{I}$'s choice to adopt the puppet strategy is exactly what the ideal party will do, but that it may not be so in the case of a *coerced* party.

Emulating the Untouched Parties. The adversary, real or ideal, sees nothing regarding the untouched parties—those that it neither corrupts nor coerces—until their encrypted votes are broadcast just before shuffling. For these parties, $\mathcal{I}$ picks random inputs x_i, prepares a correctly distributed ballot ciphertext and receipt for such an input, and stores them for future use. No messages are exchanged with $\mathcal{A}$ regarding these untouched parties.

Emulating the Voting Machine. $\mathcal{I}$ emulates the voting machine $\mathcal{M}$ for the benefit of the corrupt and coerced puppet parties. It does so exactly as a normal voting machine would, using the election public key pk. When $\mathcal{A}$ orders the voting machine to broadcast all ciphertext votes, $\mathcal{I}$ does so using the ciphertexts it generated interactively with the puppets of the corrupt and coerced parties (who are themselves talking to the real-world adversary $\mathcal{A}$), and the ciphertexts it generated on its own for the untouched parties.

Emulating the Mixnet. When $\mathcal{I}$ receives the order to shuffle from $\mathcal{A}$, it sends the signal to the trusted third party to perform its voting tabulation function, and obtains the outputs: plaintext ballots. It then emulates for $\mathcal{A}$ the mixnet functionality, using these exact outputs as the claimed result of the shuffling. In the end, $\mathcal{I}$ outputs whatever $\mathcal{A}$ outputs.

Receipt-Freeness. Note how $\mathcal{I}$ and $\mathcal{A}$ corrupt and coerce exactly the same parties in the real and ideal worlds. By a standard hybrid argument, we can see that, if the outputs of $\mathcal{I}$ and $\mathcal{A}$ are distinguishable, then there exists two variants of $\mathcal{I}$ with distinguishable outputs that only differ by one party's response to a coercion attempt. (We can safely ignore the differences regarding the untouched parties, since those depend entirely on the semantic security of the cryptosystem.) The party's actions in the MarkPledge2 protocol include only candidate selection and challenge input. The challenges are exactly the same, thanks to the coercion-resistance strategy, because the party has no private information to report to the adversary before entering her challenge. Thus the only difference is the candidate selection,

and thus the only difference in the output is the non-revealed portion of the ciphertexts. If these are distinguishable, then the semantic security of the underlying El Gamal cryptosystem has been broken.

D Structure of $SO(2,q)$

D.1 Order of $SO(2,q)$

Let $R = GL(2,q)$ be the full matrix ring generated by $SO(2,q)$, so $R = \left\{ \begin{pmatrix} a & b \\ -b & a \end{pmatrix} : a,b \in \mathbf{Z}_q \right\}$. One easily verifies that $R \equiv \mathbf{Z}_q[X]/(X^2+1)$ under $(X) \leftrightarrow \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$.

Lemma 2 *If q is prime, and $q \equiv 3 (\mathrm{mod}\ 4)$, then $SO(2,q)$ is cyclic of order $q+1$.*

Proof. X^2+1 is an irreducible element of $\mathbf{Z}_q[X]$, hence R is a field and R^* is cyclic. Since $\det$ is multiplicative[2], $SO(2,q)$ is a multiplicative subgroup of R^*, *hence is cyclic*.

To deduce its order, note that the map $\alpha(z) = z^{q-1}$ defines a multiplicative homomorphism, $\alpha : R^* \to R^*$. Since $\det$ is multiplicative, and since $a^{q-1} = 1$ for all $a \in \mathbf{Z}_q^*$, $\alpha(R^*) \subset SO(2,q)$. On the other hand, there are only two elements of $\mathbf{Z}_q$ whose determinant (norm) is $1 : \{1,-1\}$. Since $q-1 = 4k+2$, $-1 \in \alpha(R^*)$, which implies $SO(2,q) \subset \alpha(R^*)$. Thus we conclude:

1. $SO(2,q) = \{ z^{q-1} : z \in R^* \}$.
2. $|SO(2,q)| = |R^*|/|\ker(\alpha)| = (q^2-1)/(q-1) = q+1$.
3. If G is a generator of the multiplicative group R^*, then $g = G^{q-1}$ is a generator of $SO(2,q)$.

□

Lemma 3 *If q is prime, and $q \equiv 1 (\mathrm{mod}\ 4)$, then $SO(2,q)$ is cyclic of order $q-1$.*

Proof.

We can find $i \in \mathbf{Z}_q$ such that $i^2 = -1$. Define matrices A and B in R by

$$A = \begin{pmatrix} 1/2 & i/2 \\ -i/2 & 1/2 \end{pmatrix} \qquad B = \begin{pmatrix} 1/2 & -i/2 \\ i/2 & 1/2 \end{pmatrix} \tag{1}$$

Since $A^2 = A$, $B^2 = B$ and $AB = 0$, each of the following statements can be verified mechanically.

1. Every element of R can be expressed uniquely as $aA + bB$ where $a,b \in \mathbf{Z}_q$.
2. Under this correspondence, R^* consists of exactly those elements with *both* $a \neq 0$ and $b \neq 0$.
3. Further, since $AB = 0$, this correspondence defines a *multiplicative* homomorphism from R^* to $\mathbf{Z}_q^* \times \mathbf{Z}_q^*$. It follows from 2 that this is an *isomorphism*.
4. Under this coorespondence, $SO(2,q)$ consists of exactly those elements of R^* for which $ab = 1$.
5. Hence $SO(2,q)$ is naturally isomorphic to $\mathbf{Z}_q^*$ via either of the two coordinate projections (i.e. $z \to a$ or $z \to b$). Since $\mathbf{Z}_q^*$ is cyclic of order $q-1$, so is $SO(2,q)$.

□

[2]Note that $\det$ is identical to $N_{R|Z_q}$, the *norm* of R over $\mathbf{Z}_q$ as an algebraic extension.

D.2 Geometric Interpretation

One can interpret each of these elements in $SO(2,q)$ as rotations. Consider $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in SO(2,q)$ such that $\boldsymbol{w} = \boldsymbol{u} \otimes \boldsymbol{v}$. Then consider the vector dot product: $\boldsymbol{u} \cdot \boldsymbol{w}$. First, we recall $\boldsymbol{w}$:

$$\boldsymbol{w} = (u_0 v_0 - u_1 v_1, u_0 v_1 + u_1 v_0)$$

then, we perform the dot product:

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{w} &= u_0^2 v_0 - u_0 u_1 v_1 + u_0 u_1 v_1 + u_1^2 v_0 \\ &= (u_0^2 + u_1^2) v_0 \\ &= v_0 \end{aligned}$$

In other words, the dot product between two vector elements is the first coordinate of the ratio between the two vectors, where by ratio we mean to use the inverse group operation. This property matches the geometric interpretation of rotation transformations, where the dot product yields the effective cosine of the angle between the two vectors (as they are both of norm 1.) Note also that the inverse of $\boldsymbol{c}$ retains the same first coordinate v_0, since, in $SO(2,q)$, the inverse of a matrix is its transpose. This is also consistent with the rotation interpretation: whichever direction one rotates, the resulting dot product between the start and end vectors should be the same given a fixed angle of rotation.

The geometric interpretation can be particularly useful when combined with the dot-product operation. In particular, consider $\boldsymbol{\gamma}$ a generator element of $SO(2,q)$. Vector addition and dot-product are performed in $\mathbf{Z}_q^2$, while exponentiation denotes operations in $SO(2,q)$. We know from above that:

$$\boldsymbol{\gamma}^i \cdot \boldsymbol{\gamma}^j = \boldsymbol{\gamma}^{j-i}[0]$$

Then, if we have

$$\boldsymbol{\gamma}^i \cdot \boldsymbol{\gamma}^j = \boldsymbol{\gamma}^i \cdot \boldsymbol{\gamma}^k$$

we conclude that:

$$j - i = \pm(k - i) \bmod 4\Lambda$$

This leads us to the property mentioned in the main body of the paper. Consider vector addition and dot-product in $\mathbf{Z}_q^2$, $\otimes$ the $SO(2,q)$ operation, and exponentiation the composed $SO(2,q)$ operation:

$$\begin{aligned} (\boldsymbol{\gamma}^{i+2k} - \boldsymbol{\gamma}^i) \cdot \boldsymbol{\gamma}^{i+k} &= (\boldsymbol{\gamma}^{i+2k} \cdot \boldsymbol{\gamma}^{i+k}) - (\boldsymbol{g^i} \cdot \boldsymbol{g^{i+k}}) \\ &= \boldsymbol{\gamma}^{-k}[0] - \boldsymbol{\gamma}^k[0] \\ &= 0 \end{aligned}$$

This then leads to Lemma 1 from the main body of the paper, which we restate here.

Lemma 1 *Given $\boldsymbol{t} \in SO(2,q)$, and $\alpha_0 \in \mathbf{Z}_q$, there are exactly two elements $\boldsymbol{u}, \boldsymbol{v} \in SO(2,q)$, such that:*

$$\boldsymbol{u} \cdot \boldsymbol{t} = \boldsymbol{v} \cdot \boldsymbol{t} = \alpha_0$$

Proof. Recall that the vector-representation of elements in $SO(2,q)$ is $(\alpha, \beta) \in \mathbf{Z}_q^2$, where $\alpha^2 + \beta^2 = 1 \bmod 4\Lambda$. Denote $\boldsymbol{\delta} \in SO(2,q)$ such that $\boldsymbol{\delta} = \alpha_0$. There is, of course, exactly one such element.

Then, by prior reasoning regarding vector dot product:

$$\boldsymbol{u} \cdot \boldsymbol{t} = \alpha_0 \Longrightarrow \boldsymbol{u} = \boldsymbol{\delta}^{\pm 1} \otimes \boldsymbol{t}$$

Denote $\boldsymbol{\gamma}$ a generator of $SO(2,q)$, $c \in \mathbf{Z}_{4\Lambda}^*$ such that $\boldsymbol{t} = \boldsymbol{\gamma}^c$, and $d \in \mathbf{Z}_{4\Lambda}^*$ such that $\boldsymbol{\delta} = \boldsymbol{\gamma}^d$. The vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are thus $\boldsymbol{\gamma}^{(c+d)}$ and $\boldsymbol{\gamma}^{(c-d)}$, and no other elements in $SO(2,q)$ can have the same dot product α_0 with $\boldsymbol{t}$.

□