Practical Quantum-Safe Voting from Lattices, Extended

Ian Black[1], Emma McFall[2], Juliet Whidden[3], Bryant Xie[4], and Ryann Cartor*[5]

This work was partially supported by the National Science Foundation under Grant DMS-1547399.

[1] Boston College, blackia@bc.edu
[2] Brown University, emma_mcfall@brown.edu
[3] Vassar College, jwhidden@vassar.edu
[4] University of Arkansas, bjxie@uark.edu
*[5] Clemson University, rcartor@clemson.edu, 864-656-5239

Abstract

E-voting offers significant potential savings in time and money compared to current voting systems. Unfortunately, many current e-voting schemes are susceptible to quantum attacks. In this paper, we expand upon EVOLVE, an existing lattice-based quantum-secure election scheme introduced by Pino et al. We are able to make these expansions by extending the dimensions of the voter's ballot and creating additional proofs, allowing for applicability to realistic election schemes. Thus, we present our system of schemes, called EVOLVED (Electronic Voting from Lattices with Verification and Extended Dimensions). We present schemes for numerous different types of elections including Single-Choice Voting, Borda Count, and Instant Runoff.

Key Words: E-Voting, Post-Quantum Cryptography, Lattice-Based Cryptography

1 Introduction

The importance of a secure e-voting scheme in the context of modern technology cannot be exaggerated; however, few extensive e-voting schemes exist (especially those of lattice-based construction) and fewer still have practical applications. Existing voting systems often utilize primitive pen and paper voting, and those that do utilize e-voting are often discovered to posses numerous security risks. Vote security is an ongoing project in many countries, and e-voting still needs significant development for wide-scale usage.

Most current e-voting schemes rely on the hardness of factoring discrete logarithms, which will lack security in a post-quantum world. The e-voting developed in [3] and extended in this paper are based on difficult Lattice problems. Lattice-based cryptography is currently a promising field for post-quantum cryptographic schemes, and a likely candidate for the security of widespread e-voting in the future.

In [3], researchers developed a lattice-based e-voting scheme for binary elections called EVOLVE (Electronic Voting from Lattices with Verification) theorized to be quantum-safe and efficient enough for practical use. In this paper we introduce extensions of EVOLVE that allow the cryptographic tools to be used to a greater number of election applications.

2 Preliminaries

2.1 Notation

We will denote matrices with bold uppercase letters, and vectors with bold lowercase letters. We let $\mathcal{R} := \mathbb{Z}[x]/\langle x^n + 1\rangle$, and $\mathcal{R}_q := \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. Values in modulus q will be represented as values between $-\lfloor \frac{q}{2} \rfloor$

and $\lfloor \frac{q}{2} \rfloor$. We will write $d \leftarrow D$ to denote that an element d was sampled from a set D. Similarly, $d \xleftarrow{\$} D$ denotes that d is sampled uniformly at random from D.

Throughout this paper, we will represent a voter's ballot as a message m. The number of candidates in an election will be denoted as k, and the total tally of an election will be t. In each scheme, w represents the number of candidates that a voter is allowed to vote for ($w \leq k$) in a given ballot. In addition, N_V and N_A represents the number of voters and authorities, respectively, in a given election.

2.2 Hard Lattice Problems

We define the following hard problems based on lattices. These problems are assumed to be difficult for classical and quantum computers. We will define the search and decision type of the Learning with Errors (LWE problem), along with the Short Integer Solution (SIS) problem.

Definition 1 (Decision LWE)**.** *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ with standard deviation σ, and $\mathbf{s}$ be a secret vector in $\mathbb{Z}_q^n$. Choose $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, $e \in \mathbb{Z}_q$ according to χ, and return $(\mathbf{a}, b) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Given a number of samples $(\mathbf{a}, b)$, and samples $(\mathbf{a}, u)$, where $u \in \mathbb{Z}_q$ is chosen uniformly at random, the distributions are computationally indistinguishable.*

Definition 2 (Search LWE)**.** *Let n, q be positive integers, χ be a probability distribution on $\mathbb{Z}$ with standard deviation σ, and $\mathbf{s}$ be a secret vector in $\mathbb{Z}_q^n$. Choose $\mathbf{a} \in \mathbb{Z}_q^n$ uniformly at random, $e \in \mathbb{Z}_q$ according to χ, and return $(\mathbf{a}, b) = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$. Given a number of samples $(\mathbf{a}, b)$, try to recover $\mathbf{s}$.*

Definition 3 (SIS)**.** *Given $\mathbf{A} \leftarrow \mathbb{Z}_q^n$, find $\mathbf{z} \in \mathbb{Z}_q^n$, $0 < ||\mathbf{z}|| \leq \beta$, such that $\mathbf{Az} = 0$.*

Ring-LWE (R-LWE), and the related Modulo LWE (M-LWE), are both variations of the LWE problem in which the group $\mathbb{Z}_q^n$ is replaced with the ring $\mathbb{Z}_q[x]/\langle x^n + 1 \rangle$. Our scheme uses the M-LWE problem, which is said to have the same hardness as the LWE problem [5]. The short integer solution problem also has ring and modulo variants with the same level of hardness.

2.3 Outline of Cryptographic Voting Scheme

Each cryptographic voting scheme will be comprised of two groups of people: voters and authorities. Voters will cast a ballot, and authorities will tally the results. To cast a ballot, voters will take a valid vote of 0 or 1 and break their vote into partial components before encryption. The number of partial components will be equal to N_A, the total number of authorities present. Each authority will then receive their respective encrypted partial components from each voter in the total voting pool. Each authority will add all of the partial sums received from each voter into one encrypted value. Lastly, all the authorities will share their one encrypted sum with each other, add them all together, and decrypt to create a final total tally. Unless every authority were to collude to discover a particular voter's response, each voter's vote would remain secret.

3 Algorithms and Analysis from Practical Quantum-Safe Voting from Lattices, [3]

In this section, we outline the algorithms and proofs presented in [3]. We start by considering an election where each voter can submit either a 0 or a 1 as their vote.

3.1 Cryptographic Primitives

3.1.1 Commitment Scheme

The goal of a commitment scheme is to allow an individual to commit to a specific message m using a commitment $\mathbf{c}$, while keeping the message m hidden. There are three stages to a commitment scheme: key generation, a committing stage (where the individual commits to their value), and an opening stage (where

the message is revealed). A commitment scheme should satisfy the properties of correctness (opening the commitment $\mathbf{c}$ will give you the message m), hiding (a commitment $\mathbf{c}$ will hide the value of the message m), and binding (the commitment $\mathbf{c}$ can only be opened to the message m). We let $d \in \mathbb{N}$, $\sigma \in \mathbb{R}$, and $B_r \in \mathbb{R}^+$ be a positive bound. In this scheme m is the vote, $\mathbf{r}$ is a randomness vector, and $\mathbf{c}$ is the commitment.

	Keygen(1^λ)
Step 1.	$\mathbf{A}' \xleftarrow{\$} \mathcal{R}_q^{d\times(d+1)}$
Step 2.	$\mathbf{A} = \left[\begin{array}{c\|c} \mathbf{A}' & \mathbf{I}_d \end{array}\right] \in \mathcal{R}_q^{d\times(2d+1)}$
Step 3.	$\mathbf{B} \xleftarrow{\$} \mathcal{R}_q^{1\times(2d+1)}$
Step 4.	Output $\mathbf{C} := \left[\begin{array}{c} \mathbf{A} \\ \mathbf{B} \end{array}\right] \in \mathcal{R}_q^{(d+1)\times(2d+1)}$

Figure 1

	Commit $(m \in \{0,1\})$
Step 1.	$\mathbf{r} \leftarrow \mathcal{D}_\sigma^{n(2d+1)}$
Step 2.	Output $\mathbf{c} = \mathbf{Com}(m;\mathbf{r}) := \mathbf{Cr} + \left[\begin{array}{c} \mathbf{0} \\ m \end{array}\right] \in \mathcal{R}_q^{d+1}$

Figure 2

	Open$(\mathbf{c} \in \mathcal{R}_q^{d+1}, \mathbf{r} \in \mathcal{R}_q^{2d+1})$
Step 1.	If there exists $m' \in \mathcal{R}_q$ such that $\mathbf{c} - \mathbf{Cr} = \left[\begin{array}{c} \mathbf{0} \\ m' \end{array}\right]$ and $\|\mathbf{r}\| \leq B_r$, output m'.
Step 2.	Else, output $\perp$ (error)

Figure 3

Section 3.1 of [3] proves that this commitment scheme is computationally hiding, binding, and correct.

Proof of Binding Property A scheme is defined as being computationally binding if it is sufficiently secure such that a commitment cannot be opened to two different messages. It should be computationally hard for an adversary to generate information capable of opening two unique messages. It is shown in [3] that the binding property is equivalent to the M-SIS problem.

Proof of Hiding Property A scheme is defined as being computationally hiding if a commitment successfully hides the hidden message. An adversary should not, without significant difficulty, be able to distinguish between real commitments and useless commitments of the same distribution. An adversary capable of distinguishing valid commitments in a uniform distribution would be capable of solving the LWE problem, which is known to be computationally hard. The proof of hiding was originally shown in [3] and shows that the hiding property is equivalent to M-LWE.

Proof of Correctness A scheme is defined as having correctness if when given a message m, performing the opening algorithm on the commitment of m returns m with overwhelming probability. The correctness of this commitment scheme is guaranteed by Lemma 2.4 in [3].

3.1.2 OR Proofs and Amortized Proof

Other important components of the voting scheme are the OR proofs and Amortized proofs. The goal of these algorithms is to prove that a commitment $\mathbf{c}$ opens to an allowed value m without revealing the actual value of m. If m must be in $\{0,1\}$ to be valid, its validity can be proven by showing that "either $m = 1$ or $m = 0$" is true. Secrecy is maintained by not showing which statement is true, but instead proving that the union of statements is true. The proof should satisfy the properties of correctness, zero knowledge, and soundness (refer to [3] for a demonstration of each). Amortized proofs can be visualized as OR proofs performed in batches. Because the OR proofs are approximate, when showing a proof of knowledge of $\mathbf{r}$ such that $\mathbf{c} = \mathbf{Cr} + \begin{bmatrix} \mathbf{0} \\ m \end{bmatrix}$, one is actually showing that $f\mathbf{c} = \mathbf{Cr} + \begin{bmatrix} \mathbf{0} \\ fm \end{bmatrix}$ for some polynomial f that fits certain parameters (detailed in [3]). By setting f equal to a constant, such as 2, one can prove in zero knowledge that they have information about a certain randomness $\mathbf{r}$ for a sum of commitments. Authorities use this method to show that all the votes they received are valid in a single amortized OR proof.

3.2 The Scheme

Here we present the general scheme presented in [3]. The first step is to generate public parameters n, q, d, σ and public key $\mathbf{C} \leftarrow \mathbf{KeyGen}(1^\lambda)$, $\mathbf{C} \in \mathcal{R}_q^{(d+1)\times(2d+1)}$. We consider ($\mathbf{KeyGen}(1^\lambda)$, $\mathbf{Enc}$, $\mathbf{Dec}$) to be a CCA-Secure public key encryption scheme which authorities use to obtain the shares of the randomness of each voter. The key pair $(pk_j, sk_j) \leftarrow \mathbf{KeyGen}(1^\lambda)$ is chosen for each authority and the jth authority is given sk_j. Each pk_j is published, while only Authority j knows sk_j.

3.2.1 Casting a Vote

$\mathbf{Vote}_i$ Voter i splits their vote m into N_A parts, $\{v_i^{(1)}, \cdots, v_i^{(N_A)}\}$, where $v_i^{(j)} \xleftarrow{\$} \mathbb{Z}_q$ for $j \in \{1, \cdots, N_A - 1\}$, and $v_i^{N_A}$ is chosen so that $v_i = \sum_{j=1}^{N_A} v_i^{(j)}$. Voter i chooses a random vector $\mathbf{r}_i^{(j)} \leftarrow \mathcal{D}_\sigma^{n(2d+1)}$ for each authority j and uses the commitment key described above to compute $\mathbf{c}_i^{(j)} := \mathbf{Com}(v_i^{(j)}; \mathbf{r}_i^{(j)})$ for each authority j. We denote Voter i's total randomness vector as $\mathbf{r}_i := \sum_1^{N_A} \mathbf{r}_i^{(j)}$ and Voter i's total commitment vector $\mathbf{c}_i := \sum_1^{N_A} \mathbf{c}_i^{(j)}$.

Voter i proves that their message is a valid vote using the OR proof $\pi_i^V = \Pi_{OR}(\mathbf{c}_i, \mathbf{r}_i)$. Voter i also encrypts each randomness vector sk_j using authority j's respective public key s.t. $\mathbf{e}_i^{(j)} = \mathbf{Enc}(\mathbf{r}_i^{(j)}, pk_j)$ Voter i signs and posts $b_i = (id_i, \pi_i^V, (\mathbf{c}_i^{(j)}, \mathbf{e}_i^{(j)})_{j\in[N_A]})$ to the bulletin board where id_i is the voter ID.

Testing the Ballot Each voter will publish their ballot $(id_i, \pi_i^V, \mathbf{c}_i^{(1)}, \mathbf{e}_i^{(1)}, \cdots, \mathbf{c}_i^{(N_A)}, \mathbf{e}_i^{(N_A)}) := b_i$ to the bulletin board. In order to verify the ballot, an authority would first check that b_i was signed by the appropriate voter ID id_i and then verify the OR proof π_i^V. Authority j would then decrypt each randomness vector using their private key to get $\mathbf{r}_i^{(j)} := \mathbf{Dec}(\mathbf{e}_i^{(j)}, sk_j)$. Finally the authority will verify that $\mathbf{r}_i^{(j)}$ is an acceptable randomness vector, i.e. that it is less than or equal to the parameter $2\sqrt{n(2d+1)}\sigma$.

3.2.2 Tallying

Tally the votes (authorities) For each b_i, authority j use the secret key sk_j to decrypt $\mathbf{e}_i^{(j)}$ to recover randomness $\mathbf{r}_i^{(j)} := \mathbf{Dec}(\mathbf{e}_i^{(j)}, sk_j)$. Authority j then proves that $\mathbf{r}_i^{(j)}$ is a valid opening of $\mathbf{c}_i^{(j)}$ for each voter i. These proofs are amortized to increase efficiency. $\pi^{A,(j)} = (\pi_i^A, ...\pi_{N_v}^A) = \prod_{AMO,A} = (\mathbf{a}_i^{(j)}, \cdots, \mathbf{a}_{N_V}^{(j)}, \mathbf{r}_i^{(j)}, \cdots \mathbf{r}_{N_V}^{(j)})$. Authority j then computes $\mathbf{r}^{(j)} = \sum_{i=1}^{N_V} \mathbf{r}_i^{(j)}$ and will sign and publish $\pi^{A,(j)}, \mathbf{r}^{(j)}$.

Tally the votes (anyone) Anyone can then compute $\mathbf{c}^{(j)} := \sum_{i=1}^{N_V} \mathbf{c}_i^{(j)}$ for each authority j. Using these partial commitments $\mathbf{c}_i^{(j)}$ we can remove the randomness $\mathbf{r}^{(j)}$ to obtain the partial tallies $t^{(j)}$ as follows:

$$\begin{bmatrix} \mathbf{0} \\ t^{(j)} \end{bmatrix} = \mathbf{c}^{(j)} - \mathbf{C}\mathbf{r}^{(j)}.$$

We can then compute and publish the final tally

$$t = \sum_{j=1}^{N_A} t^{(j)}.$$

Verify (anyone) This voting scheme satisfies universal verifiability, meaning anyone can check that all honest votes were counted correctly. This proof can be found in Section 1.3 of [3].

3.3 Lattice-Based Voting Schemes with Multiple Candidates

In Appendix B of [3], the authors expanded their scheme from a simple yes-no vote into multiple simultaneous yes-no elections. They achieved this by changing the votes from being $m \in \{0,1\}$ to $\mathbf{m} \in \{0,1\}^k$, changing the dimensions of the public key $\mathbf{C}$ to $(d+k) \times (2d+k)$ with $\mathbf{A} \in \mathcal{R}_q^{d \times (2d+k)}$ and $\mathbf{B} \in \mathcal{R}_q^{k \times (2d+k)}$, and letting each $\mathbf{r}_i^{(j)}$ be a vector of length $2d+k$.

Importantly, the authors of [3] proved that the scheme and all of their described algorithms and proofs are still secure in this case. However, these changes alone creates a scheme for which voters have no limit on the number of candidates they choose to vote for, which is rarely the case in practice. In order to transition to a multi-candidate election with a restriction on the number of votes each Voter can cast, we add a Hamming Weight Proof. This will allow for restrictions on the allowed number of nonzero votes cast by each voter.

4 EVOLVED: Hamming Weight Proof

The Hamming Weight Proof allows us to extend the algorithms of EVOLVE to the extension EVOLVED. The Hamming Weight Proof is designed to reveal the weight (or number of nonzero entries) of the vote vector $\mathbf{m}$ while maintaining voter privacy. The point of this proof is to be able to guarantee that a voter i votes for exactly w candidates.

We consider a vote of the form $\mathbf{m}_i = [m_{i,1}, \ldots, m_{i,k}] \in \{0,1\}^k$, where k is the number of candidates. Each candidate will be assigned a placement $z \in \{1, \ldots, k\}$. Placing a 1 in the z^{th} coordinate of $\mathbf{m}_i$ denotes a vote for candidate z. To perform the Hamming Weight proof, voter i splits each element of their vote into N_A components, denoted $\mathbf{v}_i^{(j)}$, as previously described. In addition to the information previously published by the authorities, Authority j will now also publish $\tau_i^{(j)}$ for each Voter i where $\tau_i^{(j)} = \sum_{z=1}^{k} v_{i,z}^{(j)}$. We can then see that $\tau_i = \sum_{j=1}^{N_A} \tau_i^{(j)}$ represents the number of votes cast by voter i. This allows us to require a specific number of votes (i.e., nonzero entries in $\mathbf{m}$) that voter i can cast. We denote w as the number of allowed votes.

Figure 4 illustrates an example of the Hamming Weight Proof when $w = 1$. For this illustration we consider the plaintext of the partial votes sent to each authority, instead of the commitment and the encrypted randomness vector.

We do not currently have the same level of universal verifiability of the partial sums as we do with the total tally of votes because we cannot publish $\mathbf{r}_i$ in the same way that we publish $\mathbf{r}^{(j)}$ since that would betray each voter's ballot. But, we can publicly verify that the total number of votes are correct.

	$j=1$	$j=2$	$j=3$	$\sum_{j=1}^{N_V}$
$v_i^{(j)}$	$\begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -4 \\ 0 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 \\ -1 \\ -2 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
$\tau_i^{(j)}$	5	-2	-2	1

Figure 4: Because the authorities' partial sums add up to 1 ($5-2-2=1$), we can confirm the voter voted exactly once.

Consider the following notation. Given the public matrix $\mathbf{B}$,

$$\mathbf{B} = \begin{bmatrix} B_{1,1} & \cdots & B_{1,2d+k} \\ B_{2,1} & \cdots & B_{2,2d+k} \\ \vdots & \vdots & \vdots \\ B_{k,1} & \cdots & \cdots B_{k,2d+k} \end{bmatrix},$$

and the vote vector $\mathbf{v}_i^{(j)}$, we denote $\widehat{B}_j = \sum_{i=1}^{k} B_{i,j}$ and $\widehat{v}_i^{(j)} = \sum_{z=1}^{k} v_{i,z}^{(j)}$. Let $\widehat{\mathbf{C}} := \begin{bmatrix} \mathbf{A} \\ \widehat{\mathbf{B}} \end{bmatrix} \in \mathcal{R}_q^{(d+1)\times(2d+k)}$ where $\widehat{\mathbf{B}} = \begin{bmatrix} \widehat{B}_1 & \cdots & \widehat{B}_{2d+k} \end{bmatrix}$.

Given the definition of the commitment vector $\mathbf{c}_i^{(j)}$, we define the corresponding $\widehat{\mathbf{c}}_i^{(j)}$ vector as follows:

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,d+k} & 1 & 0 & \cdots & 0 \\ A_{2,1} & \cdots & A_{2,d+k} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{d,1} & \cdots & A_{d,d+k} & 0 & 0 & \cdots & 1 \\ B_{1,1} & \cdots & B_{1,d+k} & \cdots & \cdots & \cdots & B_{1,2d+k} \\ B_{2,1} & \cdots & B_{2,d+k} & \cdots & \cdots & \cdots & B_{2,2d+k} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ B_{k,1} & \cdots & B_{k,d+k} & \cdots & \cdots & \cdots & B_{k,2d+k} \end{bmatrix} \begin{bmatrix} r_{i,1}^{(j)} \\ \vdots \\ r_{i,d+k}^{(j)} \\ \vdots \\ r_{i,2d+k}^{(j)} \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ v_{i,1}^{(j)} \\ \vdots \\ v_{i,k}^{(j)} \end{bmatrix} = \begin{bmatrix} r_{i,1}^{(j)} A_{1,1} + \cdots + r_{i,d+k}^{(j)} A_{1,d+k} + r_{i,d+k+1}^{(j)} \\ r_{i,1}^{(j)} A_{2,1} + \cdots + r_{i,d+k}^{(j)} A_{2,d+k} + r_{i,d+k+2}^{(j)} \\ \vdots \\ r_{i,1}^{(j)} A_{d,1} + \cdots + r_{i,d+k}^{(j)} A_{d,d+k} + r_{i,2d+k}^{(j)} \\ r_{i,1}^{(j)} B_{1,1} + \cdots + r_{1,2d+k}^{(j)} B_{1,2d+k} + v_{i,1}^{(j)} \\ \vdots \\ r_{i,1}^{(j)} B_{k,1} + \cdots + r_{2d+k}^{(j)} B_{k,2d+k} + v_{i,k}^{(j)} \end{bmatrix} = \mathbf{c}_i^{(j)}$$

$$\begin{bmatrix} A_{1,1} & \cdots & A_{1,d+k} & 1 & 0 & \cdots & 0 \\ A_{2,1} & \cdots & A_{2,d+k} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{d,1} & \cdots & A_{d,d+k} & 0 & 0 & \cdots & 1 \\ \widehat{B}_1 & \cdots & \widehat{B}_{d+k} & \cdots & \cdots & \cdots & \widehat{B}_{2d+k} \end{bmatrix} \begin{bmatrix} r_{i,1}^{(j)} \\ \vdots \\ r_{i,d+k}^{(j)} \\ \vdots \\ r_{i,2d+k}^{(j)} \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \widehat{v}_i^{(j)} \end{bmatrix} = \begin{bmatrix} r_{i,1}^{(j)} A_{1,1} + \cdots + r_{i,d+k}^{(j)} A_{1,d+k} + r_{i,d+k+1}^{(j)} \\ r_{i,1}^{(j)} A_{2,1} + \cdots + r_{i,d+k}^{(j)} A_{2,d+k} + r_{i,d+k+2}^{(j)} \\ \vdots \\ r_{i,1}^{(j)} A_{d,1} + \cdots + r_{i,d+k}^{(j)} A_{d,d+k} + r_{i,2d+k}^{(j)} \\ r_{i,1}^{(j)} B_{1,1} + \cdots + r_{1,2d+k}^{(j)} B_{1,2d+k} + v_{i,1}^{(j)} \\ \vdots \\ r_{i,1}^{(j)} \widehat{B}_{k,1} + \cdots + r_{2d+k}^{(j)} \widehat{B}_{k,2d+k} + v_{i,k}^{(j)} \end{bmatrix} = \widehat{\mathbf{c}}_i^{(j)}$$

Note that

$$\widehat{\mathbf{c}}_i^{(j)} = \begin{bmatrix} c_{i,1}^{(j)} \\ \vdots \\ c_{i,2d}^{(j)} \\ \sum_{\ell=2d+1}^{k} c_{i,\ell}^{(j)} \end{bmatrix}.$$

We can verify $\tau^{(j)} = \sum_{i=1}^{N_V} \tau_i^{(j)}$ for each authority j by computing

$$\widehat{\mathbf{C}}\mathbf{r}^{(j)} - \widehat{\mathbf{c}}^{(j)} = \begin{bmatrix} \mathbf{0} \\ \tau^{(j)} \end{bmatrix}.$$

We can also verify that the total number of votes are correct by verifying

$$\widehat{\mathbf{C}}\mathbf{r} - \widehat{\mathbf{c}} = \begin{bmatrix} \mathbf{0} \\ N_V \cdot w \end{bmatrix}.$$

5 Using EVOLVED for Non-Ranked Elections

As the name suggests, EVOLVED is a modification of the EVOLVE scheme that allows us to apply the original scheme to multiple election types. In order for EVOLVE to become EVOLVED, we must extend the secret message (0 or 1) into some other data structure (such as vector or matrix) of a set size determined by the type of election with each component being a 0 or 1. We must also use the Hamming Weight Proof to verify that a voter did not over-vote by marking too many entries with 1.

The first umbrella of elections we consider are "non-ranked" elections, meaning voters may vote for multiple candidates without ranking the candidate choices. These elections consist of k candidates, and the voter can vote for w of these candidates.

5.1 EVOLVED scheme for single-choice, multiple-candidate

When EVOLVED is used in a single-choice, multiple-candidate election, each voter's ballot $\mathbf{m}_i \in \{0,1\}^k$ and $w = 1$. A nonzero entry in the zth coordinate of $\mathbf{m}_i$ represents a vote for candidate z. Two proofs are needed in order to verify each ballot is valid under our voting scheme: Multiple OR proofs, which proves that every element in $\mathbf{m}_i$ is in $\{0,1\}$, and the Hamming weight proof. The Hamming Weight proof is used to determine whether there is only one nonzero entry in $\mathbf{m}_i$. The commitment process here is the same described in Appendix B of [3]. Recall that in this case the public commitment key is $\mathbf{C} \in \mathcal{R}_q^{(d+k)\times(2d+k)}$ with $\mathbf{A} \in \mathcal{R}_q^{d\times(2d+k)}$ and $\mathbf{B} \in \mathcal{R}_q^{k\times(2d+k)}$. In addition to the information previously included in an Authority's post, each Authority will also post $\{\tau_1^{(j)}, \ldots, \tau_{N_V}^{(j)}\}$.

The tallying process is similar to that of a binary voting system, just with k tallies. Each authority receives commitments $\mathbf{c}_{i,1}^{(j)}, \ldots, \mathbf{c}_{i,k}^{(j)}$ and the corresponding encrypted randomness values from each voter instead of just one $\mathbf{c}_i^{(j)}$. Authority $\mathcal{A}^{(j)}$ first decrypts the associated randomness vector to each commitment using their private key. Authority $\mathcal{A}^{(j)}$ can then compute and post $\mathbf{r}^{(j)} = \sum_{i=1}^{N_V} \mathbf{r}_i^{(j)}$ and compute $\mathbf{t}^{(j)}$ as

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{t}^{(j)} \end{bmatrix} = \mathbf{c}^{(j)} - \mathbf{C}\mathbf{r}^{(j)}.$$

The final tally $\mathbf{t} = \sum_{j=1}^{N_A} \mathbf{t}^{(j)} = [t_1, \ldots, t_k]^\top$ is a vector of length k. Candidate z is the winner when $t_z = \max(t_1, \ldots, t_k)$. Anyone can compute $\begin{bmatrix} \mathbf{0} \\ \mathbf{t} \end{bmatrix} = \mathbf{c}^{(j)} - \mathbf{C}\mathbf{r}$ and verify that $\begin{bmatrix} \mathbf{0} \\ N_V \end{bmatrix} = \hat{\mathbf{c}} - \mathbf{C}\hat{\mathbf{r}}$.

5.2 EVOLVED scheme for multiple-choice, multiple-candidate: Block Voting

Block Voting, also known as plurality-at-large voting, is a voting system in which first-past-the-post elections are adopted to allow multiple candidates to be elected from the same district. Block Voting allows voters to vote for up to w candidates. We want to be able to set up the ballot in such a way that a voter could vote for any number w', where $0 \leq w' \leq w$, candidates without revealing how many votes they actually placed.

In a Block Voting election the vote $\mathbf{m}_i$ will be an element of $\mathbb{F}_2^{k+w}$. The first k positions correspond to the k candidates, and the next w positions are "reject slots." A voter votes for any number w' candidates, then fills in any $w - w'$ of the reject slots. This ensures that the Hamming weight for every voter's vote is w while the actual the number of candidates that any voter approves of is kept private.

The voting process is the same as described in Section 5.1, just adapted to contain the appropriate dimensions. When tallying, only the first k positions of a ballot are considered. The remaining "reject slots" are only used for verification and do not affect the tally at all. These k positions are tallied as described in Section 5.1. The number of total votes computed by the Hamming Weight Proof should equal wN_V.

6 Using EVOLVED for Ranked Voting Elections: Borda Count

A Borda Count election system allows voters to rank the candidates in order of preference. Different weights are assigned to each rankings, and the candidate with the highest number of weighted points is declared the winner. We consider the case with k candidates and each voter is allowed to rank up to w of them. We denote each vote as a $(w+1) \times (k+1)$ matrix, which in this section we will denote as $\mathbf{M}_i$. Each column represents a candidate and each row represents a desired positioning or ranking (1st, 2nd, 3rd, etc.). In addition, one "dummy row/ranking" is added to the bottom and one "dummy column/candidate" is added on the right. A voter fills out a ballot $\mathbf{M}_i$ by placing a 1 in each row/column pair that is desired, and a 0 in all others. If a voter decides to undervote, i.e. to choose to rank fewer than w candidates, they would place a 1 in the dummy column under every rank for which they didn't choose a candidate. Additionally, a voter would place a 1 in the dummy row under every candidate that the voter did not rank (see Appendix A for example). If desired, a slightly more efficient scheme is possible by removing dummy rows/columns. In that case, undervoting would be prohibited and by necessity $k = w$.

The Borda Count Scheme is very similar to the scheme outlined in Section 5.2, the main difference being the size of the matrices and the vote. A vote matrix, $\mathbf{M}_i$, described above, creates the vote vector $\mathbf{m}_i$ used for the commitment algorithm by concatenating the columns of $\mathbf{M}_i$ into one large column vector. Once authorities receive the partial vote vectors $\mathbf{v}_i^{(j)} \in \mathbb{Z}_q^{(w+1)(k+1)}$, the authorities will "un-concatenate" the vector to get the $(w+1) \times (k+1)$ matrix $\mathbf{V}_i^{(j)}$.

An OR proof is computed on each element and a Hamming weight proof is run on each row and column, except the dummy row and column, to ensure that every voter only ranked each candidate once and only chose one candidate for each ranking. This proof is not done on row $w+1$ or column $k+1$ in an effort to maintain voter privacy. The Hamming weights of either of these would reveal whether a voter undervoted, and if so, how many candidates that voter ranked.

Once these verification proofs are completed we can compute the tally matrix $\mathbf{T}$ by computing the tally vector $\mathbf{t}$ as described previously, and then writing the vector as a matrix as described above. Let $\mathbf{t}_n$ denote the n^{th} row vector of $\mathbf{T}$,

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 \\ \vdots \\ \mathbf{t}_{w+1} \end{bmatrix}.$$

We compute the final number of points each candidate received by computing the row vector

$$\bar{\mathbf{t}} = \sum_{n=1}^{w} (w - n + 1)\mathbf{t}_n.$$

7 Using EVOLVED for Ranked Voting Elections: Instant Runoff

Instant Runoff is a type of ranked-choice voting in which a candidate will only be declared the winner if he or she has a majority (i.e. more than half) of the votes. In Instant Runoff, voters have one transferable vote, which often changes round-to-round. After every voter has cast their ranked vote, the authorities tally up first-choice votes and determine whether a candidate has a majority of all first place votes cast. If a candidate has a majority, they are declared the winner. If not, the authorities remove the candidate with the lowest tally from all of the ballots. The first place votes originally cast for the eliminated candidate are redistributed to those voters' second-place choices (or in future rounds, to the voter's highest ranked choice that remains in the election). This process repeats until one candidate has a majority of the first place votes.

Adopting Instant Runoff to secure elections is difficult due to the large number of possible voting combinations, $k!$ possible rankings, which creates challenges related to time and size constraints. Furthermore, Instant Runoff requires that the authorities be able to connect the different rankings between candidates on a ballot. This threatens to harm voter security.

Our scheme utilizes a single voter ballot as a tree of votes $\mathcal{M}_i$ (see Figure 5). The top level in the tree consists of a single vector representing the initial vote, comparing all candidates. Each subsequent level acts as the possible combinations of vote rankings with each possible pattern of candidate removal. Therefore, the subsequent vectors are opened only if the authorities need to verify and tally the necessary combination, and discarded if the pattern of candidate elimination leaves them irrelevant.

7.1 The Scheme

An Instant Runoff scheme requires some sort of ordering of candidates to determine a ranked ballot, which is an obstacle under voting schemes attempting to protect vote privacy. Our scheme achieves private rankings by employing a tree of possible elimination combinations for k candidates. Voter i will choose their rankings and then will create the corresponding tree, denoted $\mathcal{M}_i$, as illustrated in Figure 5. We then define a vote vector $\mathbf{m}_i$ by the concatenating many vectors $\widetilde{\mathbf{m}}_{r,i}$. Each vector $\widetilde{\mathbf{m}}_{r,i}$ is derived from round r of $\mathcal{M}_i$. Figure 5 illustrates how to go from a voter's ranking, to the tree $\mathcal{M}_i$, to vectors $\widetilde{\mathbf{m}}_{r,i}$, and finally end up with a vote vector $\mathbf{m}_i$

Voter time is greatly increased by sending many vectors, however, we substantially minimize authorities' time by only opening the vectors that must be used (as candidates are eliminated). Because the initial scheme is more demanding on authority time, we prefer to place the burden on voter time. This is further explained in Section 8.

7.1.1 Casting a Vote

As described above, a voter's ballot can by represented as a tree $\mathcal{M}_i$ of different vectors representing the voter's first choice candidate, given any possible arrangement of candidates still in the running. The actual ballot $\mathbf{m}_i$ is a concatenation of vectors that create the tree $\mathcal{M}_i$. This tree consists of levels or rounds of elimination, $r \in \{0, 1, 2, ..., k-2\}$. Each level r contains $\binom{k}{k-r}$ vectors, each in $\{0,1\}^{k-r}$. The tree will continue until round $r = k-2$. Once only 2 candidates are remaining, one must be a majority.

Similarly to the above schemes, concatenated vectors forming the ballots will be split into vectors $\mathbf{v}_i^{(j)}$ where $\mathbf{m}_i = \sum_{j=1}^{N_A} \mathbf{v}_i^{(j)}$ and the commitment of $\mathbf{v}_i^{(j)}$ is sent to Authority j.

7.1.2 Verification and Tallying

The tally vector will have

$$N := \sum_{n=0}^{k-2} (k-n) \binom{k}{k-n}$$

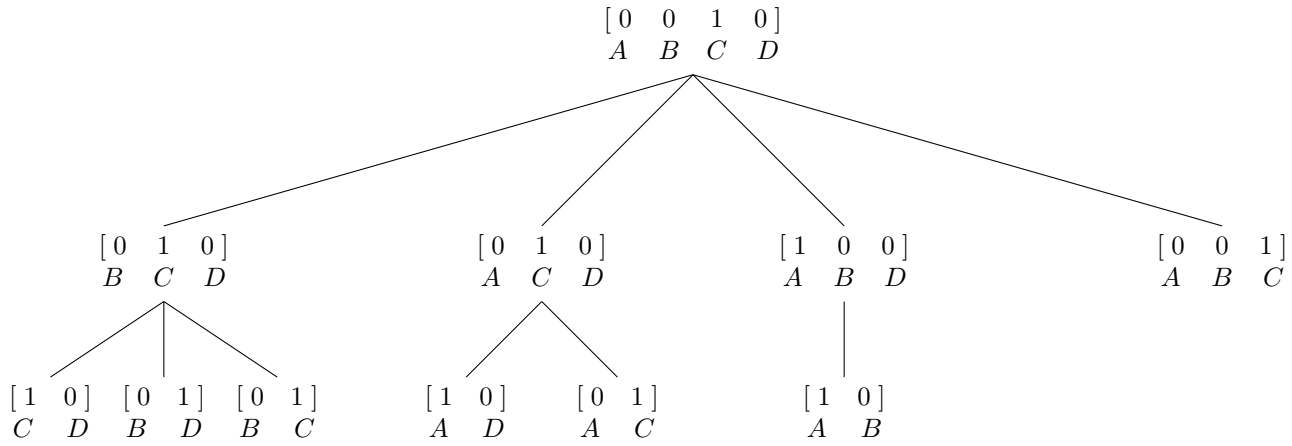
elements, i.e. $\mathbf{t} = [t_1, \ldots, t_N]^\top$. To determine the winner from the tally vector $\mathbf{t}$, we will use a sliding window technique. We first consider the round with all of the candidates, which we will denote as round 0. We determine the winner by viewing $[t_1, \ldots, t_k]^\top$. If the winner from round 0 has more than half of the votes, they are declared the winner and we are done. Otherwise, we eliminate the candidate with the lowest number of votes and move to round 1.

If we let $N_\alpha := \sum_{n=0}^{\alpha} (k-n)\binom{k}{k-n}$, then generally speaking in round r we consider windows of size $k-r$ within the set $[t_{N_r}, \ldots, t_{N_{r+1}}]$. The specific subset of size $k-r$ within $[t_{N_r}, \ldots, t_{N_{r+1}}]$ is dependent upon which r candidates have been eliminated. For example, if candidate z was eliminated after round 0, then in round 1 we will consider the tally from elements $[t_{zk-(z-2)}, \ldots, t_{(z+1)k-z}]$.

Voters' ballots should only have elements in $\{0,1\}$, and in each vector node of the tree only one vote is allowed. Therefore, the Hamming weight of each node on the tree should equal exactly 1, and the hamming weight of $\widetilde{\mathbf{m}}_{r,i} = \binom{k}{k-r}$. Voters will set up OR proofs and Hamming Weight proofs for every portion of

Figure 5: We consider an example with candidates A, B, C, and D, which voter i has ranked as C, A, D, B. Each vector in the tree holds the voter's top choice of the remaining candidates.Note that we do not need to repeat the same rankings under multiple branches. For example, we write $[AB]$ under $[ABD]$ but not again under $[ABC]$.

Voter i's Ranking: C, A, D, B
Voter i's tree, $\mathcal{M}_i$:



$\widetilde{\mathbf{m}}_{r,i}$ vectors:

$$\widetilde{\mathbf{m}}_{0,i} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^\top$$
$$\widetilde{\mathbf{m}}_{1,i} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top$$
$$\widetilde{\mathbf{m}}_{2,i} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}^\top$$

Final vote vector:

$$\mathbf{m}_i = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}^\top$$

$\mathbf{m}_i$ that represents a node in $\mathcal{M}$, but authorities will only amortize and run the OR and Hamming weight proofs of the elements of $\mathbf{v}_i^{(j)}$ used in the tallying process. By only running proofs on the sub-vectors that are actually utilized, the authorities' time is significantly reduced.

8 Efficiency and Security

8.1 Effeciency

8.1.1 Non-Ranked Elections

The efficiency of the multi-candidate scheme is comparable to that of the binary scheme in [3], but instead of a single OR proof, k OR proofs would be necessary (one for each candidate). The scheme in [3] had a voting time of 8.5ms per voter and a post-election verification time of 0.15s per voter. As a result, this scheme is easily scalable to any realistic number of candidates.

8.1.2 Borda Count Elections

We can compare our scheme to the efficiency of [3]. The major change in efficiency is that our scheme needs $\ell = (w+1)(k+1)$ OR-proofs, whereas the scheme in [3] only requires one. As a result, much of our efficiency is determined by changes in ℓ. Assuming w is constant, the time and space demands of our scheme would increase linearly with k. Should w increase with k, the demands of our scheme would increase quadratically with k.

We use the implementation speeds presented in [3] to estimate how fast our scheme would be in a mock election. The authors of [3] construct an election with 11,000 voters and three authorities. We will keep these parameters and expand the election to five candidates in which the top three candidates are ranked. Our scheme would take about 23 times as long as the binary election for a total running time of approximately 15.72 hours.

This is a significant increase in the running time. However, it should be noted that the voting step still requires less than one fifth of a second per voter. Consequently, almost all of the increased time can be attributed to the verification and tallying steps. These steps occur after the election has happened and are thus less time sensitive. In addition, authorities are more likely to have access to powerful hardware, speeding up the verification and tallying processes.

An important note about this scheme is that efficiency greatly depends on whether w remains constant or increases linearly with k. If w remains constant, size and time increase roughly linearly with k. If w increases with k, size and time increase quadradically with k. This is a significant difference, as shown in Figure 6. If $w = 3$ and remains constant, the scheme is fairly practical even up to $k = 15$. The biggest limitation is that the election would take 43 hours to tally, but since this is only done by authorities after all ballots are in, it is probably reasonable to allow two days before results are announced. If w increases linearly and holds its maximum value of $w = k$, both size and time increase much more quickly. With $k = 15$ candidates, tallying takes over a week, which is impractical. In this case, the practical limit of k is probably around or under $k \leq 10$ (tallying takes about 3.5 days).

Figure 6: Efficiency of Borda Count with 11,000 voters

w and k values	$w = 3$ $k = 5$	$w = 3$ $k = 10$	$w = 3$ $k = 15$	$w = 5$ $k = 5$	$w = 10$ $k = 10$	$w = 15$ $k = 15$
Voter ballot size (MB)	0.46	0.86	1.26	0.70	2.40	5.10
Ballot size (GB)	5.06	9.46	13.8	7.70	26.40	56.10
Voter time (seconds)	0.20	0.37	0.54	0.298	1.02	2.17
Total time (hours)	15.72	29.38	43.05	23.92	82.00	174.25

These values are based off of EVOLVE efficiency, scaled based on the number of OR proofs needed for each set of parameters.

8.1.3 Instant Runoff Elections

The voter's time and size requirements increase with respect to the number of candidates k using the following formula: $\sum_{i=2}^{k} \binom{k}{i}$. Although this is a high rate of growth, casting a ballot in a single binary election is extremely fast (taking only 8.5ms)[3]. Thus our scheme remains relatively fast for as many as seven or eight candidates.

The authorities' time requirements grow quadratically based on the equation $\sum_{i=1}^{k} i = \frac{n(n+1)}{2}$. This is because the authority is only required to open and tally some of the ballots. The above equation is based on the assumption that no candidate will receive a majority until the very last round. In reality, a candidate will frequently win outright or after a few rounds, significantly reducing the time constraints on the authorities.

The authority's space constraint is relatively high. Because the authorities must store the ballots of every voter, their space requirements are proportional to voters. That is Space Requirements$_A$ = Space Requirement$_V$ $*$ N_V.

As shown in Figure 7, our scheme remains viable with as many as eight candidates.

Although the system still has considerable issues with scalability, it is an improvement on many current Instant Runoff schemes, which represent every possible voting option as a candidate. Although these schemes can work immediately in existing voting systems, such a scheme requires $k!$ different voting options [1]. This inefficiency is acceptable in elliptical curve cryptography, where keys sizes are tiny. However, in lattice-based cryptography such a scheme would become impractical very quickly. For seven candidates, our scheme is over 10 times as fast for voters[1]. Our scheme provides even more advantages to authorities. Whereas their scheme increases authority time factorially with respect to the number of candidates, our scheme increases authority time quadratically.

Altogether, our scheme presents considerable savings compared to current lattice-based election schemes. However, more research is necessary to develop a scheme capable of handling large election with high numbers of candidates.

Figure 7: This shows the efficiency of the scheme based on various numbers of candidates assuming 11,000 voters. The numbers in this graph are based off of the efficiency presented in [3], scaled to our demands.

	$k = 7$	$k = 8$
Voter ballot size (MB)	8.4	19.8
Total ballot size (GB)	92.4	217.0
Voter time (seconds)	3.6	8.4
Total time (hours)	19	25

8.2 Concrete Security

8.2.1 Concrete Security of Hiding Property

All of the above schemes depend on the security of the LWE problem in order to ensure the security of the hiding property. That is, the LWE problem is responsible for providing concrete security ensuring that the submitted vote commitments are hidden and secure.

We use the estimator in [4] to give the bits of security for each scheme. Because the assumption for this scheme is M-LWE, our dimension n will be the length of our secret vector multiplied by the ring dimension. We have decided to keep the ring dimension of 256 from the original EVOLVE scheme. Additionally, we retain the same module size of 7 as in [3]. The dimension of the secret vector is always the length of $\mathbf{m} + 7$. The parameters that we used to create our estimates found in the below figures are as follows:

- $d = 7 + \mathbf{m}$; d is the dimension of the secret vector (which varies among schemes)
- $n = 256 * d$; n is the LWE dimension
- $\sigma = 1$; σ is the standard deviation
- $q = 2^{31} - 2^7 - 2^5 + 1$; q is the congruence modulo
- $\alpha = \sqrt{2\pi} * \sigma / \mathrm{RR}(q)$
- LWE secret distribution is $(0, 1)$
- β is block size used in BKZ algorithm

[1] Our scheme requires only 420 entries vs. 5040 entries

In [4], the estimator gives the security for different BKZ cost models. For the sake of simplicity, we have only included three cost models for each scheme, as listed in the first row of the tables. However, one could use the code in [4] and our parameters to estimate the security for the other BKZ cost models.

	LWE dimension (n)	0.265β	0.368β	$0.000784\beta^2 + 0.366\beta - 0.9 + \log(8d)$
Single-candidate	2048	172.2	239.2	585.1
Multi-candidate, single-choice	4352	439.9	610.9	2785.8
Block Voting	5120	534.8	742.6	3949.8
Borda Count	13056	1595.8	2216.1	30648.1
Instant Runoff ($k = 7$)	32512	4468.2	6204.8	229148.5

Figure 8: Sample security values (log base 2 of ring operation values) for primal attacks, $k = 10$ & $w = 3$ where relevant.

	LWE dimension (n)	0.265β	0.368β	$0.000784\beta^2 + 0.366\beta - 0.9 + \log(8d)$
Single-candidate	2048	188.7	257.2	597.7
Multi-candidate, single-choice	4352	474.6	659.1	2794.7
Block Voting	5120	573.2	780.2	4233.7
Borda Count	13056	1899.3	2596.2	33355.8
Instant Runoff ($k = 7$)	32512	5288.9	7190.7	261064.6

Figure 9: Sample security values (log base 2 of ring operation values) for dual attacks, $k = 10$ & $w = 3$ where relevant.

8.2.2 The SIS Assumption

In addition to the security of the hiding property of the commitment scheme, we must also address the security of the binding property. As shown in Section 3.1.1, the security of the binding property is based on the hardness of the Short Integer Solution (SIS) problem.

The Decision-LWE problem is reducible to the search SIS problem. If we can find a short vector $\mathbf{c}$ then we can distinguish between samples of the form $(\mathbf{a}, \langle \mathbf{a}, \mathbf{c} \rangle + e)$ and uniformly random (a, b), thus solving Decision-LWE . Dual attacks solve the Decision-LWE by reducing the problem to SIS problem, as explained in [2], so the SIS security can be estimated by our LWE estimator for dual attacks table, as shown in Figure 9.

9 Conclusion

In this paper we introduce a family of extensions of EVOLVE, which we call EVOLVED (Electronic Voting from Lattices with Verification ExtendeD). EVOLVED includes schemes for a single-choice multi-candidate election (with possible extensions to Approval Voting and Block Voting), Borda Count voting, and Instant Runoff voting. Each of these schemes is based on the scheme and proofs in [3], but we develop additional proofs and designs in order to allow the lattice based construction to be both applicable to reasonable voting expectations and remain secure. Specifically, we develop a so-called Hamming Weight Proof and alter the dimensions of a ballot.

Our biggest success in this paper was in utilizing these developments to discover working schemes for both Borda Count and Instant Runoff voting elections. However, while these schemes allow for more complexity and variety in election systems, they also decrease efficiency in both storage and time. Therefore, we expect these schemes to be stepping stones towards practical implementations, rather than usable schemes themselves.

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

[1] R. Aditya, C. Boyd, E. Dawson, and K. Viswanathan. Secure e-voting for preferential elections. In R. Traunmüller, editor, *Electronic Government, Second International Conference, EGOV 2003, Prague, Czech Republic, September 1-5, 2003, Proceedings*, volume 2739 of *Lecture Notes in Computer Science*, pages 246–249. Springer, 2003.

[2] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the {LWE, NTRU} schemes! In D. Catalano and R. D. Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2018.

[3] R. del Pino, V. Lyubashevsky, G. Neven, and G. Seiler. Practical quantum-safe voting from lattices. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1565–1581. ACM, 2017.

[4] R. P. Martin R. Albrecht and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9:169–203, 2015.

[5] O. Regev. The learning with errors problem (invited survey). In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, page 191–204, USA, 2010. IEEE Computer Society.

Appendices

A Toy Example of Borda Count

In this toy example, we demonstrate our Hamming weight proof. The OR-proofs are not demonstrated here out of simplicity. Additionally, we did not include encrypted commitments, but rather assumed the authorities were able to use their decryption keys to access the partial votes. We will consider 3 candidates ($k = 3$) where voters can rank their top 2 choices ($w = 2$) and let $q = 2$.

A.1 Casting Votes

- $\mathbf{M}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ Voter 1 ranks candidate B first and candidate A second.
- $\mathbf{M}_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$ Voter 2 ranks candidate C first and chose not to rank A or B.

Note, these votes are private and not shared with authorities directly. Voters will split these votes amongst the authorities and commit to the partial votes and post their commitments $\mathbf{c}_i^{(j)}$, encrypted randomness $\mathbf{e}_i^{(j)}$, and sum of each row and column of $\mathbf{M}$.

A.2 Verifying Votes

The authorities will receive the following information from each voter:

	Authority 1:	**Authority 2:**
Voter 1:	$\begin{bmatrix} -1 & 0 & -1 & 1 \\ 1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & -1 \\ 0 & -1 & 1 & -1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$
Voter 2:	$\begin{bmatrix} -1 & 1 & 0 & 1 \\ 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 \end{bmatrix}$

The verification process then continues as follows. Each authority begins by adding each row and column of each vote they receive to verify the Hamming weight proof. The partial sums of each row and column (excluding dummies) are shared among the authorities. The authorities then sum the partial tallies for each row and column, all of which should sum to 1.

	Authority 1:	**Authority 2:**
Voter 1:	Row $1 = -1+0+-1+1 = -1$ Row $2 = 1+1+-1+1 = 2$ Column $1 = -1+1+-1 = -1$ Column $2 = 0+1+-1 = 0$ Column $3 = -1+-1+1 = -1$	Row $1 = 1+1+1+-1 = 2$ Row $2 = 0+-1+1+-1 = -1$ Column $1 = 1+0+1 = 2$ Column $2 = 1+-1+1 = 1$ Column $3 = 1+1+0 = 2$
Voter 2:	Row $1 = -1+1+0+1 = 1$ Row $2 = 0+-1+0+0 = -1$ Column $1 = -1+0+0 = -1$ Column $2 = 1+-1+1 = 1$ Column $3 = 0+0+1 = 1$	Row $1 = 1+-1+1+-1 = 0$ Row $2 = 0+1+0+1 = 2$ Column $1 = 1+0+1 = 2$ Column $2 = -1+1+0 = 0$ Column $3 = 1+0+-1 = 0$

In this example, we see that after Authorities 1 and 2 post their column and row totals, each voter can be individually verified. First we consider Voter 1. The Row 1 partial sums for Authorities 1 and 2 are -1 and 2, respectively. Added together, this verifies that Voter 1 only ranked one candidate as their first choice. In an iterative process, the authorities verify all the columns and rows for V_1, proving V_1 submitted a correct ballot. Authorities do this for each voter, validating all ballots. In this example, all ballots are correct. Note that even though Voter 2 undervoted in this example, the dummy candidate and dummy column prevent this knowledge from being shared among the authorities

A.3 Tallying Votes

Each authority must start by tallying their partial votes. The entries corresponding to either a dummy row or column are not tallied- their purpose is to assist in verification.

For authority #1 we have $v^{(1)} = \begin{bmatrix} (-1)+(-1) & 0+1 & (-1)+0 \\ 1+0 & 1+(-1) & (-1)+0 \end{bmatrix} = \begin{bmatrix} -2 & 1 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

For authority #2 we have $v^{(2)} = \begin{bmatrix} 1+1 & 1+(-1) & 1+1 \\ 0+0 & 1+(-1) & 1+0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$

Combining the two tallies gives us our final vote tally: $v = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

At this point, we must weight the 1st and 2nd choice votes to receive the total number of points for each candidate. We will use a weight of 2 points for 1st choice and 1 point for 2nd choice. It is easy to see that Candidate A receives 1 point while Candidates B & C each receive 2 points. The end result is a tie.