An homomorphic LWE based E-voting Scheme

Ilaria Chillotti[1], Nicolas Gama[1,2], Mariya Georgieva[3], and Malika Izabachène[4]

[1] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
[2] Inpher, Switzerland
[3] Gemalto, 6 rue de la Verrerie 92190, Meudon, France
[4] CEA LIST, France

Abstract. In this paper we present a new post-quantum electronic-voting protocol. Our construction is based on LWE fully homomorphic encryption and the protocol is inspired by existing e-voting schemes, in particular Helios. The strengths of our scheme are its simplicity and transparency, since it relies on public homomorphic operations. Furthermore, the use of lattice-based primitives greatly simplifies the proofs of correctness, privacy and verifiability, as no zero-knowledge proof are needed to prove the validity of individual ballots or the correctness of the final election result. The security of our scheme is based on classical SIS/LWE assumptions, which are asymptotically as hard as worst-case lattice problems and relies on the random oracle heuristic. We also propose a new procedure to distribute the decryption task, where each trustee provides an independent proof of correct decryption in the form of a *publicly verifiable ciphertext trapdoor*. In particular, our protocol requires only two trustees, unlike classical proposals using threshold decryption via Shamir's secret sharing.

Keywords: E-vote, post quantum, fully homomorphic encryption, lattice based protocol, LWE

Acknowledgments This work has been supported in part by Région Île-de-France and Fonds Unique Interministériel (FUI) through the CRYPTOCOMP project.

1 Introduction

Electronic-voting aims at providing several elaborated properties. Basically, an e-voting protocol should ensure privacy and verifiability. The first one prevents anyone from retrieving the vote of a particular user, and the second one allows each voter to verify that his vote appears in the bulletin board (individual verifiability) and ensures that the final count of votes corresponds to the votes of legitimate voters (universal verifiability). Also, the scheme is correct when the outcome of the election counts the votes of the honestly generated votes. Among other desirable properties for e-voting schemes, there are strong forms of privacy such as receipt-freeness, coercion-resistance and ballot independence. Defining security properties for electronic based systems has long been debated and the design of secure e-voting protocols achieving all these properties happens to be more intricate than for traditional paper-based systems. Several proposal appeared over the last years and could be categorized in different ways, depending on how the privacy is guaranteed or the tally function is implemented. However, until now, the security of all provably secure protocols still rely on classical assumptions. This means that these proposed schemes could all be compromised if efficient quantum computers arise. Therefore, designing a quantum resistant e-voting scheme is very challenging, and it is a promising approach to comfort people in using e-voting protocols. This paper is a first step towards this goal.

In this paper we present a new e-voting protocol build on post quantum cryptographic primitives : unforgeable lattice-based signatures, LWE-based homomorphic encryption and trapdoors for lattices. The scheme is inspired by existing e-voting protocols, in particular Helios [2], which has already been used for medium-scale elections (and its variant Belenios). However, our scheme differs in two principal ways. The underlying primitive is different: Helios [1] is a remote e-voting protocol based on the additive property of ElGamal (which is broken by Shor's quantum algorithm). Since additive homomorphism lacks some expressiveness, each voter must ensure that the plaintext encrypted in their ballot has a specific shape, suitable for homomorphic additions. For example, if the voter gives one ciphertext per candidate, he must prove that all these ciphertexts encrypt 0, except the one corresponding to the chosen candidate, which encrypts 1. Proving such semantic properties on the plaintext without revealing its content was usually achieved using zero-knowledge proofs. In our protocol, the fully homomorphic encryption based on Ducas and Micciancio [13] bootstrapping allows to efficiently transform full-domain ciphertexts into such ciphertexts with specific semantic. This effectively removes the need of a ZK proof.

Helios uses another zero-knowledge proof in the final phase of the voting protocol, when the trustees decrypt the final result of the votes and must prove that this result is correct without revealing their own secret. In our protocol, this proof is replaced by *publicly verifiable ciphertext trapdoors*, which are produced using techniques borrowed from trapdoor-based lattice signatures, GPV [15], or [21], based on Ajtai's SIS problem.

Interestingly, combining these publicly verifiable ciphertext trapdoors with the inherent randomness of LWE-samples simplifies a lot the proof of a variant of the strongest game-based ballot privacy recently introduced by [5, Def. 7], since all the proposed oracles (except one) essentially follow the protocol, and the simulator, which is usually the most complex part of the game, is simply the identity function.

Our protocol also satisfies correctness and verifiability in the sense of [17]. In order to deter the bulletin board from stuffing itself, we add an additional authority in charge of providing each user with a private and public credential which allows him to sign his vote. This solution was already used in the variant of Helios proposed in [10]. And to compute his vote, the user encrypts his vote expressed as a sequence of 0/1, and signs the ciphertext along its public credential and sends it to the bulletin board. Cortier and Smyth [11, 24] shows that homomorphic based e-voting protocols and in particular Helios could be vulnerable to replay attacks that allow a user to cast a vote related to a previously cast ballot. This type of attacks could possibly incur a bias on the vote of other users and break privacy. Although this attack has a small impact in practice, the model for privacy should capture such attacks. Until now, this attack is prevented by removing ballots which contains a ciphertext that does already appear in a previously cast ballot. This operation is called cipherext weeding. This strategy would not work with fully homomorphic schemes, as bootstrapping operations would allow an attacker to re-randomize duplicated ballots beyond anything one can detect.

In this paper, we use the one-wayness of the bootstrapping to create some "plaintext-awareness" auxiliary information. This auxiliary information is not needed to prove the verifiability of the scheme. This information could be viewed as another encryption of the same ballot, hence it does not leak information on the plaintext vote. The only purpose of this auxiliary information is to guarantee that the ballot has not been copied or crafted from other ballots in the bulletin board as publicly viewed by other users. Thus, the voter sends this info with his ballot, which remains encrypted in the bulletin board until the end of the voting phase. At this point, for the sake of transparency, it could be safely revealed to everyone. In practice, we model this temporarily private channel by giving a public key to the bulletin board, and letting him reveal the private key at the end of the voting phase.

Finally, in order to guarantee privacy even when some of the authorities keys are corrupted, we show that our encryption scheme can be distributed among t trustees. Instead of using a threshold decryption based on Shamir's secret sharing, we rely on a simple concatenated LWE scheme. Each of the trustees carries its own decryption part, and any attempt to cheat is publicly detected. On one hand, we lose the optional ability to reconstruct the result if some trustees attempt a denial of service (which can be prevented anyway by taking the appropriate legal measures). On the other hand, once the public key has been set, we detect any attempt to cheat even if all the trustees collude. And as a bonus, our protocol can be instantiated with only two trustees which operate independently. In comparison, at least three trustees are needed for Shamir's interpolation, and if they all collude, they could produce a valid proof for a false result.

Open Problems: Our definition of an e-voting scheme differs from previous ones in that the bulletin board is carried with an additional secret to decide whether a ballot should be cast or not, but this secret key could be publicly disclosed after the voting phase. We define correctness and verifiability as in [17] and propose an adaptation of the recent definition of privacy [5] to our setting. Due to the constraint on the validation of a ballot before it is cast, the definition of the strong consistency property [5, Def.8] does not adapt properly to our setting, and thus, we leave the definition of proper extensions to strong correctness and strong consistency and privacy models against a malicious bulletin board and/or corrupted registration authority for a future work.

Finally, our proof of privacy relies on an arguably strong assumption, where a properly randomized bootstrapping function is modeled as a random oracle. We require this assumption in order to successfully simulate the Tally in the privacy game, and to a lesser extent, when we use Micciancio and Peikert's trapdoors to sample small solutions for SIS without revealing information on the trapdoor or on the keys. Proving the same result in the standard model is still an open problem.

2 Preliminaries

In the following, we specify the definition and the properties we consider for our e-voting protocol.

2.1 Definition of single pass e-voting schemes

In a single pass e-voting scheme, each user publishes only one message to cast his vote in the bulletin board. A voting scheme is specified by a family of result functions denoted as $\rho : (\mathcal{I} \times \mathbb{V})^* \to \mathcal{R}$ where $\mathbb{V}$ is the set of all possible vote, $\mathcal{I}$ is the set of voter's identifiers, $\mathcal{R}$ specifies the space of possible result. A voting scheme is also associated to a re-vote policy. In our case, we will assume that the last vote is taken into account. The entities are:

- A_1 : the authority that handles the registration of users and updates the public list of legitimate voters.
- BB, the bulletin board; The bulletin board checks the well-formedness of received ballots before they are cast. In our model, we assume that BB uses a secret key to perform a part of this task but the secret could be revealed after the voting phase.
- $\mathcal{T}$: a set of trustee(s) in charge of setting up their own decryption keys, and computing the final tally function.

Let λ denote the security parameter. We denote as ℓ the number of candidates, L an upper bound on the number of voters and t the number of trustees. We denote as $\mathcal{L}_{\mathcal{U}}$ a public list of users set at empty at the beginning. To simplify, we assume an authenticated private channel between the trustees. For our description, we will be given $\mathcal{S} = (\mathsf{KeyGenS}, \mathsf{Sign}, \mathsf{VerifyS})$ an existentially unforgeable scheme and $\mathcal{E} = (\mathsf{KeyGenE}_{\mathsf{BB}}, \mathsf{Enc}_{\mathsf{BB}}, \mathsf{Dec}_{\mathsf{BB}})$ a non-malleable encryption scheme both assumed quantum resistant. The algorithms associated to a single pass e-voting scheme could be defined as follows:

- $(\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_t), \mathsf{params}) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell, L)$: Each trustee chooses its secret key sk_i and publishes a public information pk_i and proves that it knows the corresponding secret w.r.t the published public key.

The bulletin board runs $(\mathsf{pk}_{\mathsf{BB}}, \mathsf{sk}_{\mathsf{BB}}) \leftarrow \mathsf{KeyGenE}_{\mathsf{BB}}(1^\lambda)$, it publishes $\mathsf{pk}_{\mathsf{BB}}$ and keeps $\mathsf{sk}_{\mathsf{BB}}$ private. This step implicitly defines the public pk of the e-voting scheme that includes $\mathsf{pk}_{\mathsf{BB}}$. The parameters params includes the public key pk, the numbers t, ℓ, L, the list $\mathcal{L}_{\mathcal{U}}$ and the set of valid votes $\mathbb{V}$. All these parameters are taken as input to all the following algorithms.

- $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{Register}(1^\lambda, \mathsf{id})$: on input a security parameter and a user identity, it provides the secret part of the user credential usk and its public part upk. It updates the public list $\mathcal{L}_{\mathcal{U}}$ with upk.
- $b \leftarrow \mathsf{Vote}(\mathsf{pk}, \mathsf{usk}, \mathsf{upk}, v)$: It takes as input a secret credential and public credential that possibly inclides id and a vote $v \in \mathbb{V}$. It outputs a ballot b which consists in a content message that includes upk, an encryption c of v, an auxiliary information aux encrypted using the key $\mathsf{pk}_{\mathsf{BB}}$ and a version number num plus a signature of this content message under the secret usk.
- $\mathsf{ProcessBB}(\mathsf{BB}, b, \mathsf{sk}_{\mathsf{BB}})$ As long as the bulletin board is open, when the bulletin board manager receives a ballot b: its parses it as $(\mathsf{aux}, \mathsf{upk}, c, \mathsf{num}, \sigma)$, verifies that $\mathsf{upk} \in \mathcal{L}_{\mathcal{U}}$ and uses upk to verify the signature of the ballot. Then he decrypts aux using $\mathsf{sk}_{\mathsf{BB}}$. And it performs a validity check on b and upk and finally verifies the revote policy with the version number. If b passes all these checks, it is added in BB, otherwise BB remains unchanged.
- $(\Pi_1, \ldots, \Pi_t) \leftarrow \mathsf{Tally}(\mathsf{BB}, \mathsf{sk}_1, \ldots, \mathsf{sk}_t)$: Once the voting phase is closed and the public bulletin board is published together with $\mathsf{sk}_{\mathsf{BB}}$, each trustee $T \in \mathcal{T}$ takes as input the public bulletin board BB, and its own secret key to produce a partial proof Π_i, which is publicly disclosed.
- $\mathsf{VerifyTally}(\mathsf{BB}, (\Pi_1, \ldots, \Pi_t))$: (public) takes as input t partial proofs associated to a given bulletin board BB and verifies that each individual proof Π_i is correct, and uses all of them to decrypt. It outputs a final result r and $\perp$ in case of failure.

Correctness In this paper, we only address correctness in the case where the bulletin board is supposed to be honest. In particular, it is not allowed to stuff itself or suppress valid ballots cast by honest users. Correctness for an e-voting scheme states that, if users follow the protocol, then the tally leads to the result of the election on the submitted votes. Considering an honest execution as follows: Assume $(\mathbf{sk}, \mathsf{params} = (\mathsf{pk}, \ldots)) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell, L)$ and $p = \#\mathcal{V} = \#\mathcal{I}$, where $\mathcal{V}$ is the set of valid votes whose users' identifiers lie in $\mathcal{I} = \{\mathsf{id}_1, \ldots, \mathsf{id}_p\}$. Denote as BB_i the set of the first i valid ballots cast corresponding to valid votes in $\mathcal{V}$. Then $\mathsf{ProcessBB}(\mathsf{BB}_{i-1}, b_i, \mathsf{sk}_{\mathsf{BB}})$ adds $b_i \leftarrow \mathsf{Vote}(\mathsf{pk}, \mathsf{usk}, \mathsf{upk}, v_i)$ in BB_{i-1} for all $i \leq p$ and some $\mathsf{id} \in \mathcal{I}$ s.t. $(\mathsf{usk}, \mathsf{upk}) \leftarrow \mathsf{Register}(1^\lambda, \mathsf{id})$. Also $(\Pi_1, \ldots, \Pi_t) \leftarrow \mathsf{Tally}(\mathsf{BB}_p, \mathsf{sk})$ where $\mathsf{VerifyTally}(\mathsf{BB}_p, (\Pi_1, \ldots, \Pi_t)) = r$ and $r = \rho(v_1, \ldots, v_p)$.

2.2 Security model

Privacy. Several models for privacy have been introduced over the last years. In this paper, we will use a simulation-based definition inspired from the definition recently proposed in [5]. The challenger maintains two bulletin boards BB_0 and BB_1. It randomly chooses $\beta \in \{0, 1\}$ and the adversary will be given access to

BB_β. The adversary can corrupt a subset of the trustees and the adversary can vote for candidate of his choice and cast ballots. The tally is computed on the real board in both worlds. At the end, it should not be able to tell the difference. The procedures and oracles given as access to the adversary in the definitional game are defined as follows:

- $\mathsf{Init}(1^\lambda, t, \ell, L)$: This procedure is run at the beginning interactively by the challenger and the adversary. The lists $\mathcal{L}_\mathcal{U}$ (published), $\mathcal{L}'_\mathcal{U}$ (kept by the challenger) of registered users and $\mathcal{L}_{\mathcal{CU}}$ of corrupted users are initialized at empty. The adversary might corrupt a subset ($< t$) of trustees when running the Setup algorithm and deviate from the algorithm specification. At the end, the non-corrupted secret keys are derived as well as the BB's secret key $\mathsf{sk}_{\mathsf{BB}}$ and the public parameters $(t, \ell, L, \mathcal{L}_\mathcal{U}, \mathbb{V}, \mathsf{pk})$ are published.
- $\mathsf{ORegister}(\mathsf{id})$: it checks whether an entry $(\mathsf{id}, *)$ appears in $\mathcal{L}'_\mathcal{U}$. If yes, it aborts, otherwise it runs the algorithm $\mathsf{Register}(1^\lambda, \mathsf{id})$. It updates $\mathcal{L}_\mathcal{U}$ and $\mathcal{L}'_\mathcal{U}$ with $\mathsf{upk}_{\mathsf{id}}$ and $(\mathsf{id}, \mathsf{upk}_{\mathsf{id}})$ respectively. It outputs $\mathsf{upk}_{\mathsf{id}}$.
- $\mathsf{OCorruptU}(\mathsf{id})$: it checks whether $(\mathsf{id}, *)$ appears in $\mathcal{L}_{\mathcal{CU}}$. If yes, it returns the corresponding $\mathsf{usk}_{\mathsf{id}}$. Otherwise it checks whether id has been registered using $\mathcal{L}'_\mathcal{U}$. If not, it calls $\mathsf{ORegister}$ on input id. It outputs $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$ and updates $\mathcal{L}_{\mathcal{CU}}$ with $(\mathsf{id}, \mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$.
- $\mathsf{OVote}(\mathsf{id}, \mathsf{v}_0, \mathsf{v}_1)$: if some entry $(\mathsf{id}, \mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$ does not exist in $\mathcal{L}'_\mathcal{U}$ or $v_0, v_1 \notin \mathbb{V}$, it halts. Else, it updates $\mathsf{BB}_i \leftarrow \mathsf{BB}_i \cup \{\mathsf{Vote}(\mathsf{pk}, \mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}, v_i)\}$ for $i = 0, 1$. Here the adversary has access to the public view of BB and thus to the associated ballot b.
- $\mathsf{OCast}(\mathsf{id}, b)$: if $\mathsf{upk}_{\mathsf{id}} \notin \mathcal{L}_\mathcal{U}$, it halts. Otherwise it parses b and checks its validity w.r.t $\mathsf{upk}_{\mathsf{id}}$ and the auxiliary information inside the submitted ballot using $\mathsf{sk}_{\mathsf{BB}}$. If b passes the checks, it adds b to BB_i for $i = 0, 1$.
- $\mathsf{OTally}()$: This procedure is run only once when the voting phase is closed. It runs $(\Pi_1, \ldots, \Pi_t) \leftarrow \mathsf{Tally}(\mathsf{BB}_0, \mathsf{sk}_1, \ldots, \mathsf{sk}_t)$ s.t. $r = \mathsf{VerifyTally}(\mathsf{BB}_0, (\Pi_1, \ldots, \Pi_t))$. For $\beta = 0$, it returns $(\Pi_1, \ldots, \Pi_t)$. For $\beta = 1$, it returns $(\Pi'_1, \ldots, \Pi'_t) \leftarrow \mathsf{SimTally}(\Pi_1, \ldots, \Pi_t, \text{info})$ s.t. $r' = \mathsf{VerifyTally}(\mathsf{BB}_1, (\Pi'_1, \ldots, \Pi'_t))$, where info includes auxiliary information known by the simulator $\mathsf{SimTally}$, and thus not the trustee's private keys. If $r \neq r'$, it halts.

And we define the experiment $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathsf{Vote}}(\lambda)$ in Figure 1.

Definition 2.1. We say that a voting protocol Vote has the *ballot privacy* property if there exists an efficient simulator $\mathsf{SimTally}$ such that, for any PPT[5] adversary $\mathcal{A}$, it holds that

$$\left| \Pr\left[\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathsf{Vote}}(\lambda) = \beta\right] - \tfrac{1}{2} \right| \text{ is negligible in } \lambda.$$

[5] Probabilistic Polynomial Timing

```
Experiment  $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathsf{Vote}}(\lambda)$
$(\mathbf{pk}, \mathbf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, t, \ell, L)$
$\beta' \leftarrow \mathcal{A}^{\mathsf{ORegister},\mathsf{OCorruptU},\mathsf{OVote},\mathsf{OCast},\mathsf{OTally}}(\mathsf{BB}_\beta)$
```

```
Experiment  $\mathsf{Exp}^{\mathsf{ver}}_{\mathcal{A},\mathsf{Vote}}(\lambda)$
$(\mathsf{params}, \mathsf{sk}_{\mathsf{BB}}, \mathsf{BB}, (\Pi_1, \ldots, \Pi_t), (\Pi'_1, \ldots, \Pi'_t)) \leftarrow \mathcal{A}^{\mathsf{ORegister},\mathsf{OCorruptU}}$
let $r = \mathsf{VerifyTally}(\mathsf{BB}, (\Pi_1, \ldots, \Pi_t))$
let $r' = \mathsf{VerifyTally}(\mathsf{BB}, (\Pi'_1, \ldots, \Pi'_t))$
if $r \neq \perp$ and $r' \neq \perp$ and $r \neq r'$ return 1, else return 0
```

Fig. 1. The experiments $\mathsf{Exp}^{\mathsf{bpriv},\beta}_{\mathcal{A},\mathsf{Vote}}(\lambda)$ and $\mathsf{Exp}^{\mathsf{ver}}_{\mathcal{A},\mathsf{Vote}}(\lambda)$

Verifiability. We say that a voting protocol Vote is verifiable if it ensures that the tally verification algorithm does not accept two different results for the same view of the public bulletin board. This condition has to be verified even if in the presence of malicious adversary corrupting all users except $\mathcal{A}_1$. The adversary still have access to the ORegister, OCorrupt oracles.

We define the experiment $\mathsf{Exp}^{\mathsf{ver}}_{\mathcal{A},\mathsf{Vote}}(\lambda)$ in Figure 1.

Definition 2.2. We say that a voting protocol Vote is *verifiable* if for any PPT adversary $\mathcal{A}$, it holds that $Succ^{\mathsf{ver}}(\mathcal{A}) = \Pr\left[\mathsf{Exp}^{\mathsf{ver}}_{\mathcal{A},\mathsf{Vote}}(\lambda) = 1\right]$ is negligible in λ.

3 (Cryptographic) Building blocks

Our scheme is built on the following post-quantum building blocks: Existentially unforgeable Signatures, Non-malleable Encryption, LWE-based Homomorphic Encryption, Trapdoors for lattices.

3.1 Signatures

The signature is used by the voter to sign the ballot. The security of the signature scheme in our protocol should be based on post-quantum assumptions. In our scheme, we rely on the hardness of finding short vectors in lattice. One example of lattice-based signature was proposed in [12] inspired by Lyubashevsky's scheme [19].

3.2 Scale-invariant LWE encryption.

Our protocol strongly relies on the Learning With Errors problem, first introduced by Regev in [23], and improved to obtain ring variants [20] and homomorphic encryption [8, 7, 16, 4, 13].

To ease the presentation, we use a normal form notation for LWE which captures its inherent scale-invariant property by working directly in the unit torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, not only for the right member like in Regev's original description [23], but also for the left member (i.e. no modulus q or other technical rounding). The LWE secret is decomposed as bits. This separates the main hardness parameters (i.e. entropy of secret and error rate) from implementation or optimization technicalities (which takes the form of some unspecified, and not so important discretization group). Furthermore, this representation is easy to obtain from any classical representation, and will allow us to study homomorphic protocols by reasoning directly on the (hidden) plaintext and the continuous noise.

Definition 3.1 (LWE scale-invariant normal form). Let $\alpha \in \mathbb{R}^+$ be a noise parameter, $(s_1, ..., s_n)$ be a uniformly distributed binary secret in $\mathbb{B}^n$, and $G \subseteq \mathbb{T}^n$ be a sufficiently dense[6] finite discretization group. We note $\mathsf{LWE}(\boldsymbol{s}, \alpha, G)$ the following scale-invariant Learning with errors instance. A random LWE Sample from $\mathsf{LWE}(\boldsymbol{s}, \alpha, G)$ of a message $\mu \in \mathbb{T}$ is mathematically defined as an element $(\boldsymbol{a}, b) \in G \times \mathbb{T}$ where: the left term $\boldsymbol{a} = (a_1, ..., a_n) \in G \subset \mathbb{T}^n$ is (indistinguishable from) a uniform sample of G and the right term b is equal to $\sum_{i=1}^{n} s_i a_i + \mu + e \in \mathbb{T}$ where e is statistically close from a zero-centered continuous Gaussian sample of $\mathbb{T}$ of parameter α.

Definition 3.2 (Phase). We define the *phase* of a LWE sample $(\boldsymbol{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ as $\varphi_{\boldsymbol{s}}((\boldsymbol{a}, b)) = b - \sum_{i=1}^{n} s_i a_i \in \mathbb{T}$.

As a straightforward example, a classical sample $(\boldsymbol{a}, b) \in \mathbb{Z}_q^n$ of integer LWE with binary secret and binary noise, denoted as binLWE(n,q,1.4) in [6] or [13] corresponds to the normal form sample $(\frac{\boldsymbol{a}}{q}, \frac{b}{q}) \in \mathbb{T}^{n+1}$. In this case, the left member is uniformly distributed over the discretized group $G = (\frac{1}{q}\mathbb{Z}/\mathbb{Z})^n$, and the error rate α is $\approx 1/q$. If the secret is not binary, classical binary-decomposition methods (see for instance the $\mathsf{BitDecomp}$ and $\mathsf{PowersOfTwo}$ methods from [7, sec. 3.2]) can quickly put the sample into normal form.

Security. The security of LWE therefore relies on the two other parameters: the number n of bits or entropy in the secret, and the Gaussian error parameter α. Figure 3 in the appendix summarizes the practical secure choices for (n, α), according to standard lattice reduction estimates (see [9]). In particular, for α equal to 2^{-10}, 2^{-30} or 2^{-50}, LWE is 128-bit secure as soon as $n \geq 300$, 800 and 1500 respectively, if no better attack than the lattice embedding exists. Furthermore, for any α and $n = \Omega(\log(1/\alpha))$, LWE asymptotically benefits from the worst-case to average case reduction (see [3, 23, 15, 22] or [14] depending on the shape of G).

The choice of the discretization group G controls the efficiency, but not the security of LWE. Indeed, as a simple reformulation of the Modulus-dimension

[6] Technically speaking, the smoothing parameter of the real lattice $G + \mathbb{Z}^n$ must be smaller than $\alpha/\sqrt{2n}$, as implied by [14] or [6].

reduction from [6, Cor. 3.2,3.3, Thm 4.1] or the Group switching from [14, Lem. 6.3, Cor 6.4], groups $G \subset \mathbb{T}^n$ can be swapped as long as they are sufficiently dense to not interfere with the result of the phase (i.e. decryption) function from Def 3.1. Since this function is $1/\sqrt{n}$-lipschitzian from $\mathbb{T}^n \to \mathbb{T}$, and has a precision α, it means that $\#G$ can always be chosen as small as $\log_2(\#G) = \tilde{O}(n \log_2(\alpha))$, which will be assumed in the remaining of the paper.

3.3 LWE symmetric encryption

In Def. 3.1, we define LWE samples with continuous messages $\mu \in \mathbb{T}$. The meaning of this message is natural for freshly generated LWE samples, but is less obvious when a sample is obtained as a combination of other samples. In all cases, the message μ and resp. the noise parameter α of a LWE sample c can mathematically (and not computationally) be defined as the center and resp. the Gaussian parameter of the distribution of its phase $\varphi_{\boldsymbol{s}}(c)$. Here, the probability space consists in re-sampling all Gaussian error terms of all fresh LWE samples, and in resampling all random choices that were made in decomposition or bootstrapping algorithms.

Given a security parameter λ, the noise amplitude $\bar{\alpha}$ is the smallest distance such that $|\mu - \varphi_{\boldsymbol{s}}(c)| \leq \bar{\alpha}$ with probability $\geq 1 - 2^{-\lambda}$. It typically means $\bar{\alpha} = \alpha \cdot \sqrt{\lambda/\pi}$. For a fixed security parameter, amplitude and parameters are just proportional one to each other. Bootstrapping and decryption operations are in general easier to present in terms of amplitude rather than parameter, because it better depicts the actual noise that we get.

To algorithmically extract the message from a LWE sample c like in usual decryption algorithms, we need an external information on the message: usually, μ belongs to a discrete message space $\mathcal{M}$ of packing radius $\geq \bar{\alpha}$. In this case, the message of c is computed by rounding its phase $\varphi_{\boldsymbol{s}}(c)$ to the closest point in $\mathcal{M}$. We note this $\mathsf{LWEDecrypt}_{\mathcal{M},\boldsymbol{s}}(c)$. And in this context, we will also write $\mathsf{LWESymEncrypt}_{\boldsymbol{s},\alpha,G}(\mu)$ the operation which consists in generating a random $\mathsf{LWE}(\boldsymbol{s}, \alpha, G)$ sample of $\mu \in \mathcal{M}$.

3.4 Homomorphism

LWE samples satisfy a straightforward linear homomorphism property, which follows from continuous Gaussian convolution:

Proposition 3.3 (Linear Homomorphism). Let $\boldsymbol{c_1}, \ldots, \boldsymbol{c_p}$ be p independent LWE samples of messages $\mu_1, \ldots, \mu_p \in \mathbb{T}$ and noise parameters $\alpha_1, \ldots, \alpha_p$, and let $x_1, \ldots, x_p \in \mathbb{Z}$ be p integer coefficients. Then the sample $\boldsymbol{c} = \sum_{i=1}^{p} x_i \boldsymbol{c_i}$ is a valid encryption of the message $\mu = \sum_{i=1}^{p} x_i \mu_i$ with square noise parameter $\alpha^2 \leq \sum_{i=1}^{p} x_i^2 \alpha_i^2$.

If non-linear operations are needed, one may use the following theorem, which can be viewed as an abstracted (and slightly generalized) version of the Bootstrapping theorem of Ducas and Micciancio [13].

Theorem 3.4 (General Bootstrapping Theorem). *Let λ denote a security parameter. Let $\mathcal{E} = \mathsf{LWE}(\boldsymbol{s}, \beta, G)$ and $\mathcal{E}' = \mathsf{LWE}(\boldsymbol{s'}, \alpha, G')$ be two instances of LWE with respective n and n'-bit secrets $\boldsymbol{s} \in \mathbb{B}^n$ and $\boldsymbol{s'} \in \mathbb{B}^{n'}$. We note $\bar{\alpha}$ and $\bar{\beta}$ their respective noise amplitudes. Let $\mathcal{M} \subseteq \mathbb{T}$ be an input message space at distance d from $\{-\frac{1}{4}, \frac{1}{4}\}$, and N be a power of two at least of the order of $\sqrt{(\lambda n)/(d^2 - \bar{\beta}^2)}$. If (n, β) and $(n', \alpha/\lambda\sqrt{n(N + n\log(\beta))})$ are both λ-bit secure LWE parameters, then, there exists a bootstrapping key $BK_{[(\boldsymbol{s},\bar{\beta})\rightarrow(\boldsymbol{s'},\bar{\alpha})]}$ and a polynomial bootstrapping algorithm which takes as input the bootstrapping key, a sample from $\mathcal{E}$ and two points $(\mu'_0, \mu'_1) \in \mathbb{T}^2$ of our choice, and simulates the following algorithm without knowing the secrets:*

$$\mathsf{Bootstrap}_{BK}(c, \mu'_1, \mu'_0) =$$

$$\begin{cases} \mathsf{LWESymEncrypt}_{\boldsymbol{s'},\alpha,G'}(\mu'_1) & \textbf{\textit{if}}\ d(\varphi_{\boldsymbol{s}}(c), \frac{1}{2}) < d(\varphi_{\boldsymbol{s}}(c), 0) \\ \mathsf{LWESymEncrypt}_{\boldsymbol{s'},\alpha,G'}(\mu'_0) & \textbf{\textit{otherwise}}. \end{cases}$$

Historically, the first bootstrapping notions were just designed to suppress the input noise of a ciphertext, and optionally to switch its encryption key. But from a plaintext point of view, bootstrapping was just the identity function. In contrast, the bootstrapping function from Theorem 3.4, and which is implicitly used by [13], evaluates the comparator operator (or mux) between its three arguments.

[13] provides a concrete example of $\mathrm{BK}_{[(\boldsymbol{s},\frac{1}{4}-\varepsilon)\rightarrow(\boldsymbol{s},\frac{1}{16})]}$ bootstrapping key for any 500-bit key $\boldsymbol{s}$, which has 128-bit security and whose bootstrapping algorithm runs in about 700 sequential ms, and which we intend to reuse. They use this key to fully-homomorphically simulate NAND gates between LWE samples of noise amplitude $\bar{\alpha} = \frac{1}{16}$ and message space $\mathcal{M} = \{0, \frac{1}{4}\}$, just by doing $\mathrm{HomNAND}(\boldsymbol{c_1}, \boldsymbol{c_2}) = \mathsf{Bootstrap}_{\mathrm{BK}}\left((\boldsymbol{0}, \frac{5}{8})\text{-}\boldsymbol{c_1}\text{-}\boldsymbol{c_2}, \frac{1}{4}, 0\right)$. However, for the final step of our protocol, we will also need to bootstrap to a much lower noise amplitude, which is not covered in [13], although their construction works with minor adjustments.

Theorem 3.4 implies that the output of the bootstrapping function is indistinguishable from a fresh LWE sample of μ'_0 or μ'_1. In fact, it even seems to behave like a good collision-resistant one-way function, especially if we re-randomize the input sample by adding a random combination of the public key. However, for verifiability purposes, one may also wish to control the randomness to reproduce some computations. To simplify the analysis, we will therefore model bootstrapping as a random oracle:

Assumption 3.5 (Bootstrapping as a random oracle). In the conditions of Theorem 3.4, the $\mathsf{Bootstrap}$ function is assimilated to a random oracle which returns a fresh LWE sample of μ'_b where $b = 1$ iff. $\mathrm{dist}(\varphi_{\boldsymbol{s}}(c), \frac{1}{2}) < \mathrm{dist}(\varphi_{\boldsymbol{s}}(c), 0)$. In particular, the left term $\boldsymbol{a'}$ is always indistinguishable from uniform over G'.

3.5 Publicly verifiable decryption for LWE

In the previous section, the LWE normal form with secret $\boldsymbol{s} \in \mathbb{B}^n$ has been presented in a symmetric key manner. To allow public encryption, one usually publishes a polynomial number $m = \Omega(n \log(1/\alpha))$ of random LWE samples of the message 0 with noise parameter α. This is the public key $\mathsf{pk} \in (G \times \mathbb{T})^m$. The public key can equivalently be written as a $m \times (n{+}1)$ matrix $\mathsf{pk} = [M|\boldsymbol{y}]$ of $\mathbb{T}$ where $\boldsymbol{y} = M\boldsymbol{s}^t + \mathsf{error}$. Public encryption of a message $\mu \in \mathbb{T}$ can then achieved by summing a random subset (of rows) of the public key, and adding the trivial ciphertext $(0, \mu)$ to the result. We call this operation $\mathsf{LWEPubEncrypt}_{\mathsf{pk}}(\mu)$.

In the protocol we will present, this allows a voter to encrypt his vote. Then the BB can publicly use the bootstrapping theorem to homomorphically evaluate whatever circuits produces the (encrypted) final result. And in the end, some trustee must decrypt this result using the secret key. If decrypting a LWE ciphertext on a discrete message space is easy, proving to everyone that the decryption is correct without revealing anything on the LWE secret key requires some more work.

To do so, we adopt a strategy which is borrowed from Lattice-based signatures like GPV [15]. To allow a public decryption of $\boldsymbol{c} \in G \times \mathbb{T}$, we reveal a small integer combination $(x_1, \ldots, x_m)$ of the public key pk which could have been used to encrypt $\boldsymbol{c}$, as in the following definition:

Definition 3.6 (Publicly Verifiable Ciphertext Trapdoor). Let $\mathsf{LWE}(\boldsymbol{s}, \alpha, G)$ be a LWE instance, and $\mathsf{pk} = [M|\boldsymbol{y}] \in (G \times \mathbb{T})^m$ a public key, and $\mathcal{M}$ a discrete message space of packing radius $\geq d$. Let $\boldsymbol{c} = (\boldsymbol{a}, b)$ be a sample with noise amplitude $\leq \bar{\delta}$ and $\beta = \sqrt{(d^2 - \bar{\delta}^2)/\bar{\alpha}^2}$, we say that $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{Z}^m$ is a ciphertext trapdoor of $\boldsymbol{c}$ if $\|\boldsymbol{x}\| \leq \beta$ and if $\boldsymbol{x} \cdot M = \boldsymbol{a}$ in G.

Anyone who knows the public key can verify the correctness of the ciphertext trapdoor. Furthermore, since the difference $\boldsymbol{c} - \boldsymbol{x} \cdot \mathsf{pk}$ is a trivial ciphertext $(\boldsymbol{0}, b')$ of phase b' of the same message $\mu \in \mathcal{M}$ with noise amplitude $< d$, this reveals the message in the same time.

Of course, finding a small combination of random group elements which is close to some target is related to the subset sum, or the SIS family of problems, which are hard in average. Luckily, the framework proposed in [21], and which we briefly summarize in the next paragraph, introduces an efficient trapdoor solution.

Definition 3.7 (Master Trapdoor as in def. 5.2 of [21]). Let $\mathsf{LWE}(\boldsymbol{s}, \alpha, G)$ be a λ-bit secure instance of LWE. A Gadget $\mathsf{Gad} \in G^{m'}$ is some publicly known superincreasing generating family of G, such that any element $a \in G$ can be decomposed as a small (or binary) linear combination of Gad. Let A be a uniformly distributed family in $G^{m-m'}$, and let R be a $m' \times (m\text{-}m')$ integer matrix with (small) subGaussian entries. We define the matrix $M = \begin{bmatrix} A \\ \hline A' \end{bmatrix} \in G^m$ where $A' = \mathsf{Gad} - R \cdot A$. We call R a *master trapdoor*, and its corresponding public key is $\mathsf{pk} \in (G \times \mathbb{T})^m$, whose i-th row is $\mathsf{pk}_i = (M_i | M_i \cdot \boldsymbol{s} + e_i)$ for some Gaussian noise e_i of parameter α. The master trapdoor verifies the condition $\mathsf{Gad} = [R|\, Id_{m-m'}] \cdot M$ and the parameters $m, m', m - m' = O(\log_2(\#G))$.

Theorem 3.8 (Adapted from Thm. 5.1 of [21]). *Let $\boldsymbol{c} \in G \times \mathbb{T}$ be a $\mathsf{LWE}(\boldsymbol{s}, \delta, G)$ sample on a message space of packing radius $> 2\bar{\delta}$, R a master trapdoor of parameter γ and pk an associated public key with noise $< \bar{\delta}/\bar{\gamma}\log(\#G)^{1.5}$. Given $\boldsymbol{c}$ and R, one may efficiently compute a ciphertext trapdoor $\boldsymbol{x}$ for $\boldsymbol{c}$ of norm $O(\beta \log(\#G)^{1.5})$. This trapdoor can decrypt $\boldsymbol{c}$, as in Def. 3.6. Furthermore, the distribution of the ciphertext trapdoors of $\boldsymbol{c}$ is statistically close to some discrete Gaussian distribution on $\mathbb{Z}^m$, of parameter $O(\beta \log(\#G)^{0.5})$, and thus, does not reveal any information about R.*

Like square roots oracles for RSA moduli, ciphertext trapdoors are trivially vulnerable to chosen ciphertext attacks, so they should only be invoked on the output of some good hash function, or some random oracle. It is the case for all provable instantiations of trapdoor-based lattice signatures like GPV [15] or [21], and in this paper, we will use the output of the bootstrapping algorithm, which can also be viewed as a random oracle by Assumption 3.5.

3.6 Concatenated LWE, with distributed decryption

To prevent a single authority from decrypting individual ballots, or to guaranty privacy in the long term, even if all but one trustee leak their private key, we need to split the LWE secret key among multiple trustees. We do not propose a threshold decryption like Shamir's secret sharing scheme, but instead, a simple concatenation of LWE systems where all the trustees must do their part of the decryption, and any cheater is publicly detected. This requirement seems sufficient for an e-voting scheme, and has the additional benefit of being achievable with only two trustees.

Let $\mathsf{LWE}(\boldsymbol{s_i}, \alpha, G_i)$ for $i \in [1, t]$ be λ-bit secure instances of LWE, and $\mathsf{pk}_i = [M_i | \boldsymbol{y_i}] \in (G \times \mathbb{T})^m$ be the corresponding public keys with associated master trapdoors R_i. We call concatenated LWE the LWE instance whose private key is $\boldsymbol{s} = (\boldsymbol{s_1}| \dots |\boldsymbol{s_t})$, discretization group is $G = G_1 \times \cdots \times G_t$, and public key is

$$\mathsf{pk} = \left[\begin{array}{c|c|c|c} M_1 & 0 & 0 & \boldsymbol{y_1} \\ \hline 0 & \ddots & 0 & \vdots \\ \hline 0 & 0 & M_t & \boldsymbol{y_t} \end{array}\right] \tag{1}$$

To decrypt such LWE ciphertext (with publicly verifiable decryption) $\boldsymbol{c} = (\boldsymbol{a_1}| \dots |\boldsymbol{a_t}, b) \in G \times \mathbb{T}$, each of the t trustee independently use his master trapdoor R_i to provide a ciphertext trapdoor Π_i of $(\boldsymbol{a_i}, 0)$, and like in the previous section, the concatenated ciphertext trapdoor $\Pi = (\Pi_1 | \cdots | \Pi_t)$ is a ciphertext-trapdoor for $\boldsymbol{c}$.

Finally, note that even if all trustees leak their private keys except $\boldsymbol{s_1}$,R_1 (we take 1 for simplicity), then decrypting $\boldsymbol{c}$ rewrites in decrypting the $LWE(\boldsymbol{s_1}, \alpha', G_1)$ ciphertext $(\boldsymbol{a_1}, b')$ where $b' = b - \sum_{i=2}^{t} \boldsymbol{a_i} \cdot \boldsymbol{s_i}$. This is by definition still λ-bit secure. In other words, even in case of collusions between the trustees, the whole scheme remains secure as long as one trustee is honest.

4 Detailed Description of our E-voting Protocol

4.1 Setup phase

The bulletin board manager generates a pair of keys $(\mathsf{pk}_{\mathsf{BB}},\mathsf{sk}_{\mathsf{BB}})=\mathsf{KeyGenE}_{\mathsf{BB}}(1^\lambda)$ and publishes $\mathsf{pk}_{\mathsf{BB}}$.

The trustees setup the concatenated LWE scheme presented in section 3.6: each trustee generates its own separate LWE secret key $\boldsymbol{s_i} \in \mathbb{B}^n$, its own master trapdoor R_i, and a corresponding public key $\mathsf{pk}_i \in (G \times \mathbb{T})^m$.

Thus, the secret key sk_i of each trustee consists in R_i and $\boldsymbol{s_i}$. Without revealing any information on s_i, they must provide a proof that the public key pk_i is indeed composed of $\mathsf{LWE}(\boldsymbol{s_i}, \alpha)$ samples of 0, because it is a requirement for the correctness of the decryption with ciphertext trapdoors. To do so, the trustees may for instance use the NIZK proof defined in [18, Section 2.2]. Once the existence of s_i is established, the trustees do not necessarily need to prove that they know the secrets s_i or R_i, although, the simple fact that they can output valid ciphertext trapdoors prove it anyways, by standard LWE-to-SIS or decision-to-search reduction arguments.

The main public key pk is the tensor product defined in equation (1).

The main secret key $\boldsymbol{s} = (\boldsymbol{s_1}, \ldots, \boldsymbol{s_t})$ must be secure for low noise rates, like $O(1/L^{1.5})$, which means that the number of secret bits is larger than usual. To perform homomorphic operations efficiently, the trustees define two other secret keys $\boldsymbol{s}^{(f)}$ and $\boldsymbol{s}^{(m)}$ for a noise rate 1/16 and their corresponding public key $\mathsf{pk}^{(f)}$ and $\mathsf{pk}^{(m)}$ (they may still use a concatenated scheme, although this time, they don't need a master trapdoor for that).

Finally, they provide three bootstrapping keys: $\mathrm{BK}_1 := \mathrm{BK}_{[(\boldsymbol{s}^{(f)},\frac{1}{4})\to(\boldsymbol{s}^{(m)},\frac{1}{4})]}$, $\mathrm{BK}_2 := \mathrm{BK}_{[(\boldsymbol{s}^{(m)},\frac{1}{4})\to(\boldsymbol{s}^{(m)},\frac{1}{16})]}$, and a larger one for low noise amplitude $\mathrm{BK}_3 := \mathrm{BK}_{[(\boldsymbol{s}^{(m)},\frac{1}{4})\to(\boldsymbol{s},\frac{1}{L^{3/2}})]}$. Since a bootstrapping key essentially consists in a public LWE encryption of each individual bit of the private key, each trustee can independently provide their part of the bootstrapping keys. Bootstrapping with BK_1 or BK_2 can be done in less than 700ms using the implementation of [13]. BK_3 uses secrets which are typically twice as large, and since the bootstrapping is essentially quartic in n, one should expect a slowdown by some constant factor ≈ 16.

4.2 Voter registration

$\mathsf{Register}(1^\lambda, \mathsf{id})$: The authority A_1 runs $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGenS}(1^\lambda)$. Its adds $\mathsf{upk}_{\mathsf{id}}$ in $\mathcal{L}_{\mathcal{U}}$ and outputs $(\mathsf{upk}_{\mathsf{id}}, \mathsf{usk}_{\mathsf{id}})$.

4.3 Voting phase

In our scheme, we suppose that the number of candidates $\ell = 2^k$ is a power of two. If it is not the case, we can always add null candidates. Then, if we choose a random value for the vote, no candidate will be favorite over the others. A valid vote v is thus assimilated to an integer between 0 and $\ell - 1$.

$\mathsf{Vote}(\mathsf{pk}, \mathsf{usk}, \mathsf{upk}, v)$: each user computes the binary decomposition $(v_0, \dots, v_{k-1}) \in \{0,1\}^k$ s.t $v = \sum_{j=0}^{k-1} v_j 2^j$. Let $\hat{v}_j$ denote $\frac{1}{2}v_j \in \mathbb{T}$, he encrypts each bit as $\boldsymbol{c_j} = \mathsf{LWEPubEncrypt}_{\mathsf{pk}^{(f)}}(\hat{v}_j)$ with noise amplitude $< \frac{1}{4}$. It bootstraps the c_j's as follow: $\boldsymbol{c_j'} = \mathsf{Bootstrap}_{BK_1}(\boldsymbol{c_j}, \frac{1}{2}, 0)$. It computes $\mathsf{aux} = \mathsf{Enc_{BB}}(\mathsf{pk_{BB}}, (\boldsymbol{c_0}, \dots, \boldsymbol{c_{k-1}}) || \mathsf{upk})$. It returns the final ballot $b = (\text{content}, \sigma)$, where $\text{content} = (\mathsf{aux}, \mathsf{upk}, (\boldsymbol{c_0'}, \dots, \boldsymbol{c_{k-1}'}), \mathsf{num})$ and $\sigma = \mathsf{Sign}(\mathsf{usk}, \text{content})$ and where num is the version number of the ballot for the revote policy[7].

4.4 Processing a ballot in BB

All the procedures performed by the BB and described in this section are summarized in figure 2.

Validity checks on a ballot $\mathsf{ProcessBB}(\mathsf{BB}, b, \mathsf{sk_{BB}})$: upon reception of a ballot b, it parses it as $(\text{content}, \sigma)$, with $\text{content} = (\mathsf{aux}, \mathsf{upk}, (\boldsymbol{c_0'}, \dots, \boldsymbol{c_{k-1}'}), \mathsf{num})$.

BB verifies that $\mathsf{upk} \in \mathcal{L}_\mathcal{U}$ and checks whether $\mathsf{VerifyS}(\mathsf{upk_{id}}, \text{content})$.

It verifies that each $\boldsymbol{c_j} \in G \times \mathbb{T}$. It computes $(\boldsymbol{c_0}, \dots, \boldsymbol{c_{k-1}}) || \mathsf{upk}' = \mathsf{Dec_{BB}}(\mathsf{sk_{BB}}, \mathsf{aux})$. It checks whether $\mathsf{upk}' == \mathsf{upk}$ and whether $\boldsymbol{c_j'} = \mathsf{Bootstrap}_{BK_1}(\boldsymbol{c_j}, \frac{1}{2}, 0)$ for all $j = 0, \dots, k-1$. Then, it checks the revote policy with the version number num and adds the ballot if all the validity checks passed. Note that a syntaxical check on the $\boldsymbol{c_j}$'s is enough. Note also that unlike classical e-voting protocol, no semantic check or zero-knowledge proof is needed at this step, since all binary message are valid choices.

BB homomorphic operation Then, BB applies a sequence of public homomorphic operations on the encrypted vote $(\boldsymbol{c_1'}, \dots, \boldsymbol{c_k'})$. These homomorphic operations do not require the presence of the voter, and can therefore be performed offline by the cloud. To simplify, we will just describe what happens on the cleartext.

1. *Pre-Bootstrapping* $\mathsf{Bootstrap}_{BK_2}(\boldsymbol{c_j'}, \frac{1}{4}, 0)$ is applied on each $\boldsymbol{c_j'}$ to cancel its uncontrolled input noise and reduce it to $\frac{1}{16}$, and also to reexpress its content on the $\{0, \frac{1}{4}\}$ message space, which is suitable for boolean homomorphic operations.

2. *Homomorphic binary expansion* In order to compute the sum of the votes (homomorphically), BB transforms the vector $\hat{v} = (\hat{v}_0, \dots, \hat{v}_{k-1}) \in \{0, \frac{1}{4}\}^k$ into its characteristic vector $\hat{w} = \frac{1}{4}(w_0, \dots, w_{\ell-1}) = (0, \dots, 0, \frac{1}{4}, 0, \dots, 0)$ of length ℓ (number of the possible choices of votes) with a $\frac{1}{4}$ at position v. This transformation is very easy and, for every h of binary decomposition $h = \sum_{i=0}^{k-1} h_i 2^i$, w_h

[7] The revote policy consists in accepting the last vote sent for upk: BB accepts to overwrite a ballot for upk iff the new version number is strictly larger than the previous one.

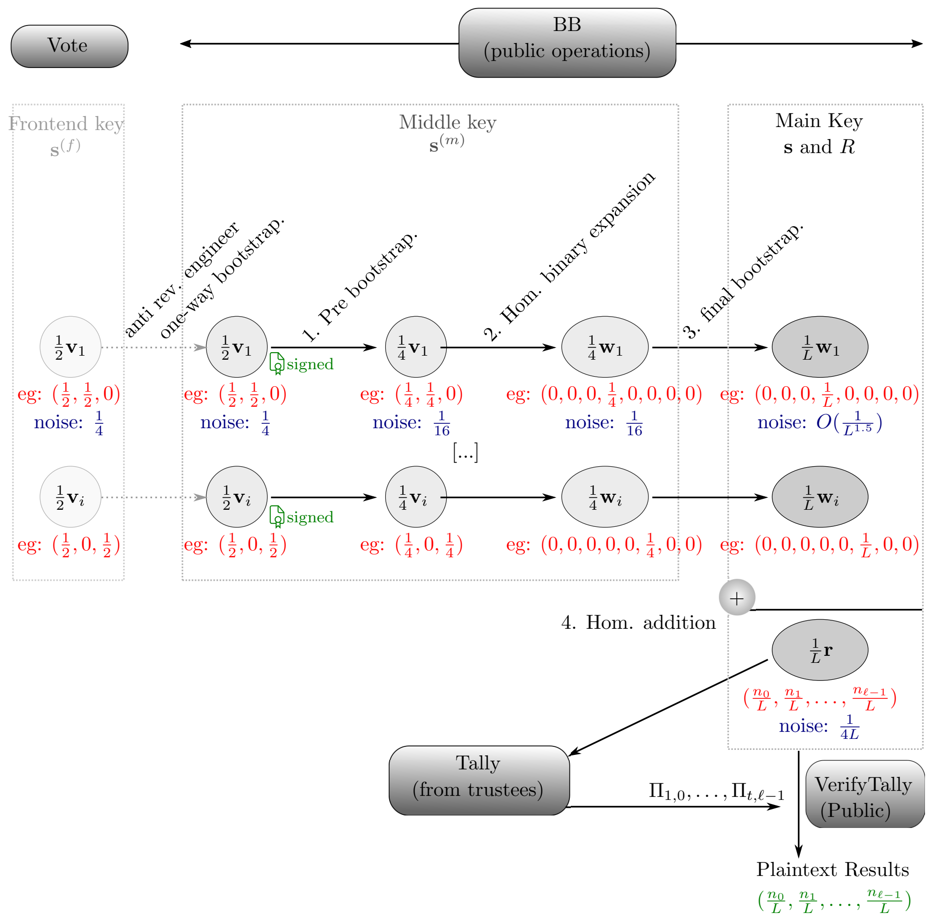
Vote
BB
(public operations)
Frontend key
$\mathbf{s}^{(f)}$
Middle key
$\mathbf{s}^{(m)}$
Main Key
$\mathbf{s}$ and R
anti rev. engineer
one-way bootstrap.
1. Pre bootstrap.
2. Hom. binary expansion
3. final bootstrap.
$\frac{1}{2}\mathbf{v}_1$
$\frac{1}{4}\mathbf{v}_1$
$\frac{1}{4}\mathbf{w}_1$
$\frac{1}{L}\mathbf{w}_1$
signed
eg: $(\frac{1}{2}, \frac{1}{2}, 0)$
noise: $\frac{1}{4}$
eg: $(\frac{1}{4}, \frac{1}{4}, 0)$
noise: $\frac{1}{16}$
eg: $(0,0,0,\frac{1}{4},0,0,0,0)$
eg: $(0,0,0,\frac{1}{L},0,0,0,0)$
noise: $O(\frac{1}{L^{1.5}})$
[...]
$\frac{1}{2}\mathbf{v}_i$
$\frac{1}{4}\mathbf{v}_i$
$\frac{1}{4}\mathbf{w}_i$
$\frac{1}{L}\mathbf{w}_i$
eg: $(\frac{1}{2}, 0, \frac{1}{2})$
eg: $(\frac{1}{4}, 0, \frac{1}{4})$
eg: $(0,0,0,0,0,\frac{1}{4},0,0)$
eg: $(0,0,0,0,0,\frac{1}{L},0,0)$
+
4. Hom. addition
$\frac{1}{L}\mathbf{r}$
$(\frac{n_0}{L}, \frac{n_1}{L}, \dots, \frac{n_{\ell-1}}{L})$
noise: $\frac{1}{4L}$
Tally
(from trustees)
$\Pi_{1,0}, \dots, \Pi_{t,\ell-1}$
VerifyTally
(Public)
Plaintext Results
$(\frac{n_0}{L}, \frac{n_1}{L}, \dots, \frac{n_{\ell-1}}{L})$

Fig. 2. Schematic of the protocol In red and green an example : red means encrypted, green means decrypted

corresponds to this boolean term:

$$w_h(v) = \left(\bigwedge_{\substack{i\in[0,k-1]\\ h_i=0}} \overline{v_i} \right) \wedge \left(\bigwedge_{\substack{i\in[0,k-1]\\ h_i=1}} v_i \right)$$

The formula seems complicated, but it is just a conjunction of k variables v_i or their negation ($k = \log_2 \ell$ is in general smaller than 5 in typical elections).

These conjunctions can be easily evaluated on ciphertexts using these homomorphic gates, keeping in mind that $\mathsf{Bootstrap}_{BK_2}$ runs in less than 700ms, as in [13]:

$$\begin{aligned} \mathsf{HomAND}(\boldsymbol{c_1}, \boldsymbol{c_2}) &= \mathsf{Bootstrap}_{BK_2}((\boldsymbol{0}, -\tfrac{1}{8}) + \boldsymbol{c_1} + \boldsymbol{c_2}, \tfrac{1}{4}, 0) \\ \mathsf{HomANDNot}(\boldsymbol{c_1}, \boldsymbol{c_2}) &= \mathsf{Bootstrap}_{BK_2}((\boldsymbol{0}, \tfrac{1}{8}) + \boldsymbol{c_1} - \boldsymbol{c_2}, \tfrac{1}{4}, 0) \end{aligned}$$

3. *Generalized Bootstrapping.* BB then uses the main bootstrapping key BK_3 to convert these ℓ ciphertexts into a new ciphertext of $(0, \dots, 0, \frac{1}{L}, 0, \dots, 0)$ with noise $O(L^{-3/2})$.

This consists in applying $\mathsf{Bootstrap}_{\mathrm{BK}_3}((\boldsymbol{0}, \frac{1}{8}) + \boldsymbol{c}, \frac{1}{L}, 0)$ to each of the ℓ ciphertexts.

4. *Homomorphic addition.* At the end of the voting phase, BB sums (homomorphically) all ciphertexts, which yields to the final LWE ciphertexts $(C_0, \dots, C_{\ell-1})$ of $(\frac{n_0}{L}, ..., \frac{n_{\ell-1}}{L})$, with noise $O(L^{-1})$. No bootstrapping is needed for this step, it just uses the standard addition on ciphertexts.

4.5 Tallying and verification

Denote as $(C_0, \dots, C_{\ell-1})$ the final ciphertext processed by BB. Each LWE sample C_j encodes the message $\frac{n_j}{L}$ with noise amplitude $O(1/L)$, where n_j is the number of votes for candidate j.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk} = (\mathsf{sk}_1, \dots, \mathsf{sk}_t))$: for each C_j, the trustees independently perform the distributed decryption described in section 3.6, and publish a ciphertext trapdoor $\Pi_{i,j} \in \mathbb{Z}^m$ (for $i = 1, \dots, t$ and $j = 0, \dots, \ell - 1$) as in definition 3.6, which is revealed to everyone.

$\mathsf{VerifyTally}(\mathsf{BB}, (\Pi_1, \dots, \Pi_t))$: given the main public key pk, anyone is able to check the validity of the ciphertext trapdoors. If a trapdoor $\Pi_{i,j}$ is invalid, it publicly proves that the i-th trustee is not honest and in this case $\mathsf{VerifyTally}$ returns $\perp$. If all the trapdoors are valid, anyone can use $(\Pi_{1,j}, \dots, \Pi_{t,j})$ to decrypt C_j, and thus, recover n_j for all j, which gives the number of votes for the candidate j. This gives the result of the election. And $\mathsf{VerifyTally}$ returns the result $(n_0, \dots, n_{\ell-1})$.

5 Correctness and Security Analysis

5.1 Correctness and Verifiability

In order to prove verifiability and correctness, we show that our scheme verifies this more general theorem.

Theorem 5.1 (Intermediate theorem for proving Verifiability and Correctness). *Let* pk *be a valid e-voting public key (this includes also* $\mathsf{pk}^{(f)}$*,* $\mathsf{pk}^{(m)}$*, and the bootstrapping keys* BK_1*,* BK_2*,* BK_3 *with the parameters defined in section 4.1, together with their respective NIZK proofs of validity). Let* $\mathcal{S}$ *be an existentially unforgeable scheme and* $\mathcal{E}$ *a non-malleable encryption scheme both quantum resistant. Let* BB *be a sequence of bits that can syntaxically be interpreted as the public view of a bulletin board after the end of the voting phase:* i.e. *a* BB *key pair (*$\mathsf{sk_{BB}}$*,* $\mathsf{pk_{BB}}$*), a list* $[b_1, \ldots, b_p]$ *of ballots where each* $b_i = (\mathsf{aux}, \mathsf{upk}, \boldsymbol{c'}, num, \sigma)$ *passed the verification check from* $\mathsf{ProcessBB}$ *and the whole sequence of public homomorphic operations from Section 4.4 until the final ciphertexts. Then, the following facts holds with overwhelming probability*

1. *For each* $b_i \in \mathsf{BB}$*, let* $(\mathsf{upk}, \mathsf{usk})$ *be the credential pair associated to this ballot, there exists a unique* $v_i \in \mathbb{V} = [0, \ell - 1]$ *such that* b_i *can be expressed as* $\mathsf{Vote}(\mathsf{pk}, \mathsf{upk}, \mathsf{usk}, v_i)$[8].
2. *If* BB *was produced in less than* 2^λ *elementary operations, then each ballot* b_i *has been generated with the knowledge of its associated* usk.
3. *The final* ℓ *ciphertexts after all the homomorphic operations in* BB *are* $\mathit{LWE}(\mathsf{sk}, 1/L^{1.5}, G)$ *samples of the plaintext result* $\boldsymbol{r} = \rho(v_1, \ldots, v_p)$.
4. *For all integer matrix* $(\Pi_1, \ldots, \Pi_t) \in \mathbb{Z}^{\ell t m}$*,* $\mathsf{VerifyTally}(\mathsf{BB}, \Pi_1, \ldots, \Pi_t)$ *is equal to either* $\boldsymbol{r}$ *or* $\perp$.

The theorem holds even if one does not perform the check on the auxiliary information, and thus, providing ($\mathsf{sk_{BB}}$*,* $\mathsf{pk_{BB}}$*) is optional.*

Proof (Sketch). Suppose that the hypothesis of the theorem are satisfied.

1. Let $b_i \in \mathsf{BB}$ be a ballot. By construction it can be parsed as $b_i = (\mathsf{aux}, \mathsf{upk}, \boldsymbol{c'}, \mathsf{num}, \sigma)$, and since b_i passes the tests from $\mathsf{ProcessBB}$, $\mathsf{Dec}(\mathsf{sk_{BB}}, \mathsf{aux}) = (\mathsf{upk}, \boldsymbol{c})$ where $\boldsymbol{c'} = \mathsf{Bootstrap}_{\mathrm{BK}_1}(c, \frac{1}{2}, 0)$. Let $\boldsymbol{c}" = \mathsf{Bootstrap}(BK_2, \boldsymbol{c'}, \frac{1}{4}, 0)$ be the result of the pre-bootstrapping of c'. Then v_i is the integer whose binary decomposition is $\boldsymbol{m} = 4 \cdot \mathsf{LWEDecrypt}_{\mathsf{sk}^(m),0,\frac{1}{4}((c'))}$. Then by theorem 3.4 on the bootstrapping, $\boldsymbol{c}$ encodes necessarily $\frac{1}{2}\boldsymbol{m}$ with noise amplitude $\frac{1}{4}$, and thus, b_i can be expressed as $\mathsf{Vote}(\mathsf{pk}, \mathsf{upk}, \mathsf{usk}, v_i)$. Reciprocally, if b_i is expressed as $\mathsf{Vote}(\mathsf{pk}, \mathsf{upk}, \mathsf{usk}, x)$, then we get back the same $v_i = x$ since we are just encrypting, bootstrapping twice and decrypting. This proves unicity.

[8] the v_i exists and is unique, but b_i might have been generated without its knowledge, or more generally, without calling the Vote procedure

2. Each ballot b_i contains a signature σ, which cannot be forged in less than 2^λ elementary operations without the private key usk assuming $\mathcal{S}$ is secure.
3. From the plaintext point of view, computing the binary expansion to transform the vote into its characteristic vector and summing these characteristic vectors (over L) yields the correct result. By theorem 3.4, the same operations are correct on the ciphertexts, which proves that the ciphertexts that are decrypted by the Tally encrypt the correct election result $\boldsymbol{r} = \rho(v_1, \ldots, v_p)$.
4. Let $(\mathit{\Pi}_1, \ldots, \mathit{\Pi}_t) \in \mathbb{Z}^{\ell t m}$ be an integer matrix. If $\mathsf{VerifyTally}(\mathsf{BB}, \mathit{\Pi}_1, \ldots, \mathit{\Pi}_t)$ is not $\perp$, then $\mathit{\Pi}_1, \ldots, \mathit{\Pi}_t$ form ciphertexts trapdoors for the final ciphertexts, which satisfy the conditions of definition 3.6. Therefore, $\mathsf{VerifyTally}(\mathsf{BB}, \mathit{\Pi}_1, \ldots, \mathit{\Pi}_t)$ is the decryption of the final ciphertexts, which are the election result $\boldsymbol{r} = \rho(v_1, \ldots, v_p)$ by point 3.

Corollary 1 (Correctness). *Assuming that the signature scheme $\mathcal{S}$ is existentially unforgeable and $\mathcal{E}$ is a non-malleable encryption scheme and that the public homomorphic operations performed by the* BB *are correct, then our protocol is* Correct*.*

Proof. The Correctness is a direct consequence of theorem 5.1, in the particular case where the public view of BB is generated by honest voters which follow the protocol, and $\mathit{\Pi}_1, \ldots, \mathit{\Pi}_t$ are generated by the Tally function.

Corollary 2 (Verifiability). *Assuming that* BB *accepts only valid ballots and that all other operations performed by the* BB *are public, then our protocol is* verifiable *in the sense of definition 2.2.*

Proof. The Verifiability is a direct consequence of point 4 of theorem 5.1. In fact, it states that the sole possible results of VerifyTally can be $\boldsymbol{r} = \rho(v_1, \ldots, v_p)$ or $\perp$. This implies that the result of the game defined in figure 1 is equal to 0 with overwhelming probability.

5.2 Privacy

Theorem 5.2. *Assuming Assumption 3.5 holds, our protocol verifies* Privacy *in the sense of definition 2.1.*

Proof (Sketch). The challenger sets two empty bulletin boards BB_0 and BB_1 picks a random bit β. The adversary may choose at most $k \leq t-1$ LWE secrets $\boldsymbol{s_1}, \ldots, \boldsymbol{s_{k-1}} \in \mathbb{B}^n$ and the challenger chooses the remaining $t-k$ secrets randomly and independently (even from the ones chosen by the adversary). As long as one trustee's secret key part is uniformly generated and unknown from the adversary, the three LWE instances generated in the Setup are λ-bit secure. All the oracles ORegister, OCorrupt, OCast, Tally and VerifyTally follow the normal protocol described in section 4. The OVote oracle follows the protocol to vote v_0 on BB_0 and v_1 on BB_1, but it chooses the output of the random oracle[9]

[9] This works well in random oracle model as in Assumption 3.5. Getting it in the standard model remains open.

$\mathsf{Bootstrap}_{BK_3}$ function on BB_1 so that it uses exactly the same left-hand term for corresponding samples in BB_0 and BB_1. Since the left term of a LWE sample is uniform in G, this is consistent with the expected output distribution of $\mathsf{Bootstrap}_{BK_3}$. Finally, $\mathsf{SimTally}$ is simply the identity function, since all LWE samples in BB_0 and BB_1 have the same left term, and the ciphertext trapdoors only depends on it. The only oracle which depends on a secret that is unknown to the adversary is Tally. We already know from Thm 3.8 that the ciphertext trapdoors of the tally do not leak any information on the master trapdoors, nor on the LWE secret. It remains to show that the result of the election does not bring any new information to the adversary. Obviously, the attacker does not get any information from OVote, since he knows the ballot plaintexts. And finally, our auxiliary information prevents the adversary from using public data in BB_β to craft a valid ballot for OCast.

□

6 Discussion and conclusion

In this paper, we presented a new post quantum e-voting protocol. Our new scheme is simple and the procedures are transparent. The construction exploits the versality of LWE-based homomorphic encryption to build a scheme reaching all the security properties, without relying on zero knowledge proofs for proving the validity of a vote, nor correct decryption. Instead, we make use of ciphertext trapdoors and rely on a new way to distribute LWE decryption which is not based on Shamir secret sharing to ensure the public verifiability of the decryption of the final result. We also introduce a new approach for preventing replay attacks, by using the one-wayness of the bootstrapping letting the user send some encrypted auxiliary information. We leave as a possible direction for future work the extension of our model to a possibly dishonest bulletin board. Lastly, our scheme is a first instanciation of an LWE-based e-voting protocol and we leave as an open problem the improvement of our scheme that would make lattice-based e-voting scheme close to practice.

References

1. Ben Adida, Olivier de Marneffe, Oliver Pereira, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *Proceedings of the 2009 conference on Electronic voting technology/workshop on trustworthy elections*, 2009.
2. Ben Adida, Olivier de Marneffe, and Olivier Pereira. Helios voting system. http://www.http://heliosvoting.org.
3. M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Proc. of 30th STOC*. ACM, 1998. Available at ECCC as TR97-047.
4. Jacob Alperin-Sheriff and Chirs Peikert. Faster bootstrapping with polynomial error. In *Crypto*, pages 297–314, 2014.

5. David Bernhard, Veronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. A comprehensive analysis of game-based ballot privacy definitions. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15)*, San Jose, CA, USA, May 2015. IEEE Computer Society Press.
6. Z. Brakerski, A. Langlois, C. Peikert, O. Regev, and D.Stehlé. Classical hardness of learning with errors. In *Proc. of 45th STOC*, pages 575–584. ACM, 2013.
7. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Crypto*, pages 868–886, 2012.
8. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.
9. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Proc. of Asiacrypt*, pages 1–20, 2011.
10. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for helios under weaker trust assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14)*, volume 8713 of *LNCS*, pages 327–344, Wroclaw, Poland, September 2014. Springer.
11. Véronique Cortier and Ben Smyth. Attacking and fixing Helios: An analysis of ballot secrecy. In *CSF*, pages 297–311. IEEE Computer Society, 2011.
12. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 40–56, 2013.
13. Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *Eurocrypt*, pages 617–640, 2015.
14. Nicolas Gama, Malika Izabachène, Phong Q. Nguyen, and Xiang Xie. Structural lattice reduction: Generalized worst-case to average-case reductions. *IACR Cryptology ePrint Archive*, 2014:48, 2014.
15. Craig Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.
16. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Crypto*, pages 75–92, 2013.
17. Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*, pages 37–63, 2010.
18. Fabien Laguillaumie, Adeline Langlois, Benoît Libert, and Damien Stehlé. Lattice-based group signatures with logarithmic signature size. In *asiacrypt*, pages 41–61, 2013.
19. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 738–755, 2012.
20. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *In Proc. of EUROCRYPT, volume 6110 of LNCS*, pages 1–23. Springer, 2010.
21. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *Eurocrypt '12*, LNCS. Springer-Verlag, 2012.
22. Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *Crypto '13*, volume 8042 of *LNCS*, pages 21–39, 2013.
23. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.
24. Ben Smyth. Replay attacks that violate ballot secrecy in helios, 2012.

A Appendix

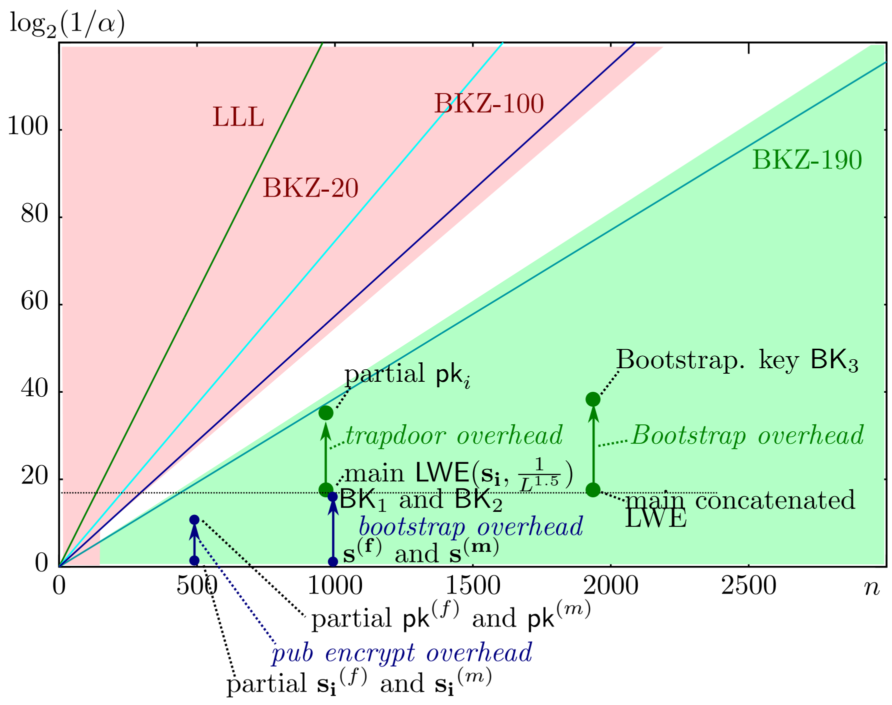

Fig. 3. 128-bit secure LWE parameters, as a function of (n, α)
Assuming a medium-scale election with $L \approx 2000$ voters, the main partial keys should allow a $1/L^{1.5} \approx 2^{-17}$ noise parameter. Taking into account the overhead for publicly verifiable ciphertext trapdoors and bootstrapping key, the overall scheme can easily be instantiated with at most 2000-bit secrets.