A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing

Xavier Boyen[1], Thomas Haines[2], and Johannes Müller[3][0000−0003−2134−3099]

[1] Queensland University of Technology, Australia,
[2] Norwegian University of Science and Technology, Norway,
[3] SnT, University of Luxembourg, Luxembourg

Abstract. Mix nets are often used to provide privacy in modern security protocols, through shuffling. Some of the most important applications, such as secure electronic voting, require mix nets that are *verifiable*. In the literature, numerous techniques have been proposed to make mix nets verifiable. Some of them have also been employed for securing real political elections.

With the looming possibility of quantum computers and their threat to cryptosystems based on classical hardness assumptions, there is significant pressure to migrate mix nets to post-quantum alternatives. At present, no verifiable and practical post-quantum mix net with external auditing is available as a drop-in replacement of existing constructions. In this paper, we give the first such construction.

We propose a verifiable decryption mix net which solely employs practical lattice-based primitives. We formally prove that our mix net provides a high level of verifiability, and even accountability which guarantees that misbehaving mix servers can also be identified. Verification is executed by a (temporarily trusted) public auditor whose role can easily be distributed. To demonstrate practicality for real-world systems, we provide detailed performance benchmarks on our stand-alone implementation based only on the most conservative lattice hardness assumptions.

Keywords: lattice-based · verifiability · accountability · mix net · e-voting

1 Introduction

Mix nets are indispensable building blocks of many secure e-voting systems. Essentially, a mix net consists of a sequence of mix servers which take as input the encrypted messages provided by the senders (e.g., the voters' ballots), secretely shuffle them, and eventually output the permutated plain messages (e.g., votes). Unless all mix servers are corrupted, the mixing breaks the individual connections between the senders and their revealed messages in the output. In the context of e-voting, this property guarantees vote privacy.

However, for *secure* e-voting, it is also important to ensure that the voters' intent be reflected correctly in the election result, even if the mix servers are corrupted and actively try to tamper with the votes. Therefore, the employed mix net must be *verifiable* to guarantee that manipulating the senders' input, and generally incorrect mixing, can be detected. Moreover, in order to deter parties from misbehaving in the first place, *accountability* is often also desirable. This stronger form of verifiability provides identification of misbehaving parties and adjudication of possible disputes. In the literature, numerous mix nets [1, 2, 4, 10, 11, 14, 15, 17, 19–21, 24, 28, 29, 32–34, 36–38] have been proposed that aim to achieve verifiability and, in some cases, accountability. Some of them have also been used for securing real political elections (see, e.g., [12, 35]).

With more and more powerful quantum computers on the horizon (see, e.g., [3]), it is important to protect mix nets even when actively targeted by quantum attackers, either contemporary or future. Due to the stark possibility that future quantum attackers could retrospectively break vote privacy, there is significant pressure to employ verifiable post-quantum mix nets *already today.*

Unfortunately, to the best of our knowledge, only a single verifiable mix net scheme [24], named sElect, has been proposed so far that could employ *practical* post-quantum, e.g., lattice-based, cryptosystems. The unique characteristic of sElect, in contrast to all other known verifiable mix nets, is to avoid (zero-knowledge) proofs of correct decryption, for which, at present, there exist no practical solutions whose security can be reduced to hardness assumptions over lattices (see Section 2 for more details). Alas, although sElect is provably secure, its security relies on the assumption that the senders/voters themselves verify the correctness of the final outcome. While this assumption is reasonable for some election scenarios, it cannot be justified in general; in particular, recourse and adjudication in case of voter-detected fraud is problematic.

Therefore, it is still an open problem to construct a practical and provably secure mix net with external auditing that can defend against quantum attacks.

Our contributions. In this paper, we present the first highly efficient and practically realizable lattice-based decryption mix net that provides a high level of verifiability and even accountability. Verification is completely executed by a (temporarily trusted) public auditor whose role can easily be distributed. This structure is the same as the one of the prominent randomized partial checking (RPC) technique [20] which was, for instance, used for elections in the Australian state of Victoria [12].

To be more precise, our mix net employs a generalized version of the *trip wire technique* that was, in a specific variant, originally employed in the mix net by Khazaei et al. [21] as a subroutine. At a high level, in this technique, the input to the mix net consists of the real input messages plus a number of trip wire messages which to a mix server are indistinguishable from the real ones. Now, if a mix server wants to manipulate the outcome, it faces the risk of "touching" at least one trip wire, in which case the mix server would be caught cheating. In contrast to the specific variant in the mix net by Khazaei et al. [21], where each *mix server* can only inject a *single* trip wire in order to be able to guarantee correctness of the verification (which furthermore requires a proof of correct decryption), we depart from this as follows. First, we do not assume that the mix servers themselves inject the trip wires to "verify each other", but place that responsibility on a number of public auditors. Just one of these auditors needs to be trusted, and in fact only temporarily, because each auditor opens its inner state once mixing has finished—which incidentally greatly simplifies adjudication in case of dispute, and could not be done to the mixers themselves. Second, each auditor does not inject just a single but many trip wires, so that the probability of being caught cheating can be made very high even for manipulating just a few messages. Trip wires are cost effective, and since we further use only the most basic and black-box cryptographic primitives (namely, public-key encryption and digital signatures), the resulting mix net can be run with extremely efficient (lattice-based) primitives that more than compensate for the trip wires' overhead compared to ZKP-based approaches.

Altogether, our contributions are as follows:

1. We first discuss the unique constraints that come into play when building mix nets with quantum resistance, and related works (Section 2).
2. We describe how to extend an arbitrary *plain* (i.e., unverifiable, proof-less) decryption mix net (Section 3) with our general version of the trip wire technique (Section 4).
3. We precisely characterize how a decryption mix net with trip wires provides a high level of verifiability and even accountability (Section 5). A formal proof is provided in our technical report [7].
4. We instantiate the generic trip wire decryption mix net using *practical* lattice-based cryptography from conservative hardness assumptions (plain LWE). We have created a self-contained optimized implementation of the lattice construction, and provide detailed benchmarks that demonstrate its practicality for real-world elections at a high level of security (Section 6).
5. We candidly discuss the general properties, benefits and drawbacks of trip wire mix nets (Section 7) and conclude in Section 8.

2 Feasibility of Post-Quantum Secure Mixing

Existing mix nets can be divided into two classes: decryption mix nets and re-encryption mix nets. In this section, we describe the main ideas of these two different approaches, and explain why the re-encryption approach is currently impractical for defending against quantum attackers.

In a *decryption mix net*, originally proposed by Chaum [8], an IND-CCA2 secure public-key encryption scheme is employed. Each mix server holds a public/secret key pair. Each sender iteratively encrypts its input message under the mix servers' public keys in reverse order, forming a multi-layered *onion.* Mixing starts with the first mix server, which "peels off" the outermost encryption layer, shuffles the result, forwards it to the second mix server, and so on. Eventually, all encryption layers have been removed and the plain input messages are published in the resulting random order.

In a *re-encryption mix net*, originally proposed by Park et al. [30], an IND-CPA secure public-key encryption scheme with re-encryption is employed. There is one public key whose secret key shares are distributed among a number of trustees. Each sender encrypts its input message under this public key. Mixing starts with the first mix server which re-encrypts its input ciphertexts, shuffles the result, forwards it to the second mix server, and so on. Eventually, all re-encrypted input ciphertexts are published in random order. Depending on the application, the output ciphertexts are either decrypted by the trustees or not.

In their plain unverifiable modes, re-encryption mix nets are more lightweight than decryption mix nets because input messages are not encrypted iteratively but only once under a single public key. However, when *verifiability in the presence of quantum attackers* is required, the trade-offs get more complicated. In general, there are two different approaches for making re-encryption mix nets verifiable, namely, by using randomized partial checking (RPC) [20] or by a proof of correct shuffle [1, 2, 4, 10, 11, 14, 15, 17, 19, 28, 29, 33, 34, 37]. On the positive side, RPC could potentially be used for making a lattice-based re-encryption mix net verifiable, for instance using one of three recently proposed lattice-based proofs of correct shuffle [10, 11, 33], although it is unclear whether or not these are practical. On the negative side, both proof-based approaches merely guarantee that the output *ciphertexts* are in fact shuffled re-encryptions of the input ciphertexts. In order to be useful for our motivating application, i.e., secure e-voting, we also have to *decrypt* the output ciphertexts verifiably. Unfortunately, to the best of our knowledge, no *practical* zero-knowledge proofs of correct decryption for lattice-based encryption have been proposed so far, whose security can itself be reduced to lattice-based hardness assumptions. Even with recent developments on sublinear arguments from lattices [5], ZK proofs tend to be, and will likely remain, much heavier and more cumbersome than simple primitives such as public-key encryption based on comparable assumptions.

As the main purpose of our mix nets would be for quantum-secure e-voting where integrity, performance and simplicity of implementation are paramount, our best bet is to devise a lattice-based *decryption* mix net that provides external auditability using only the simplest fastest primitives as building blocks.

3 Plain Decryption Mix Net

In this section, we first recall the main idea of a plain unverifiable decryption mix net [8] and then precisely describe its protocol. In Section 4, we describe the generic trip wire technique to endow a plain decryption mix net with correctness verification (and external/third-party adjudication) of its outcome.

3.1 Idea

At a high level, a decryption mix net works as follows. It consists of a number of mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_{\mathsf{MS}}}$ each of which holds a public/private (encryption/decryption) key pair $(\mathsf{pk}_k, \mathsf{sk}_k)$. Each sender iteratively encrypts its plain input message m under the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{MS}}}$ of the mix servers in reverse order, and submits the resulting "nested" ciphertext c to the first mix server M_1. The first mix server uses its secret key sk_1 to "peel off" the outermost encryption layer of all input ciphertexts, then shuffles the decrypted messages, and forwards the permutated list to the second mix server M_2. The second mix server uses its secret key sk_2 to "peel off" the second encryption layer, then shuffles the result, and so on. Eventually, the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ outputs all the plain messages initially chosen by the senders in random order.

3.2 Protocol

We now precisely describe the protocol of a plain decryption mix net.

Protocol participants. A plain decryption mix net protocol is run among *senders*, $\mathsf{S}_1, \ldots, \mathsf{S}_{n_\mathsf{S}}$, and *mix servers*, $\mathsf{M}_1, \ldots, \mathsf{M}_{n_{\mathsf{MS}}}$, using a public, append-only *bulletin board* B.

Channels. For each sender S_i, we assume that there is an authenticated channel from S_i to the bulletin board B. These channels ensure that only eligible senders are able to submit their inputs.[4]

Cryptographic primitives. We use the following cryptographic primitives:

- An IND-CCA2-secure public-key encryption scheme $\mathcal{E}$.[5]
- An EUF-CMA-secure signature scheme $\mathcal{S}$.

Protocol overview. A protocol run consists of the following consecutive phases. In the *setup* phase, parameters are generated. In the *submission* phase, the senders generate and submit their input. In the *mixing* phase, the mix servers collaboratively mix the input.

We now describe each of the protocol phases in more detail.

Setup phase. Each mix server M_k runs the key generation algorithm of the digital signature scheme $\mathcal{S}$ to generate its public/private (verification/signing) keys. The verification keys are published on the bulletin board B.

Each mix server M_k runs the key generation algorithm KeyGen of the public-key encryption scheme $\mathcal{E}$ to generate its public/private (encryption/decryption) key pair $(\mathsf{pk}_k, \mathsf{sk}_k)$, and posts its public key pk_k on the bulletin board B.

Submission phase. Each sender S_i iteratively encrypts its secret input m_i under the mix servers' public keys in reverse order, i.e., starting with the public key $\mathsf{pk}_{n_{\mathsf{MS}}}$ of the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ to the public key pk_1 of the first mix server M_1:

$$\mathsf{c}_i = \mathsf{Enc}(\mathsf{pk}_1, (\ldots, \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{MS}}}, m_i))).$$

Mixing phase. The list of ciphertexts $C_0 \leftarrow (\mathsf{c}_i)_{i=1}^{n\mathsf{s}}$ posted by the senders on the bulletin board B is the input to the mixing phase. Starting with the first mix

[4] By assuming such authenticated channels, we abstract away from the exact method the senders use to authenticate to the bulletin board; in practice, several methods can be used, such as one-time codes, passwords, or external authentication services.

[5] We also require that $\mathcal{E}$, for every public-key and any two plaintexts of the same length, always yields ciphertexts of the same length. This seems to be satisfied by all practical schemes in existence, unless implemented with entropic compression.

server M_1, each mix server M_k takes C_{k-1} as input and performs the following tasks:

1. M_k decrypts all ciphertexts in C_{k-1} under its private key sk_k:
$$\forall i \in \{1, \dots, n_{\mathsf{S}}\}\colon C'_k[i] \leftarrow \mathsf{Dec}(\mathsf{sk}_k, C_{k-1}[i])$$
2. M_k chooses a permutation π_k over $\{1, \dots, n_{\mathsf{S}}\}$ uniformly at random, and sets
$$\forall i \in \{1, \dots, n_{\mathsf{S}}\}\colon C_k[\pi_k(i)] \leftarrow C'_k[i].$$
3. M_k posts C_k on the bulletin board B.

The output $C_{n_{\mathsf{MS}}}$ of the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ is the output of the mixing phase. It equals $(m_{\pi(i)})_{i=1}^{n_{\mathsf{S}}}$, where $\pi = \pi_{n_{\mathsf{MS}}} \circ \ldots \circ \pi_1$ is the overall permutation of the mix net.

4 Trip Wire Technique

We describe how to extend a plain decryption mix net (Section 3) with trip wires. We will show in Section 5 that the resulting mix net provides a high level of verifiability and accountability in the presence of fully malicious mix servers.

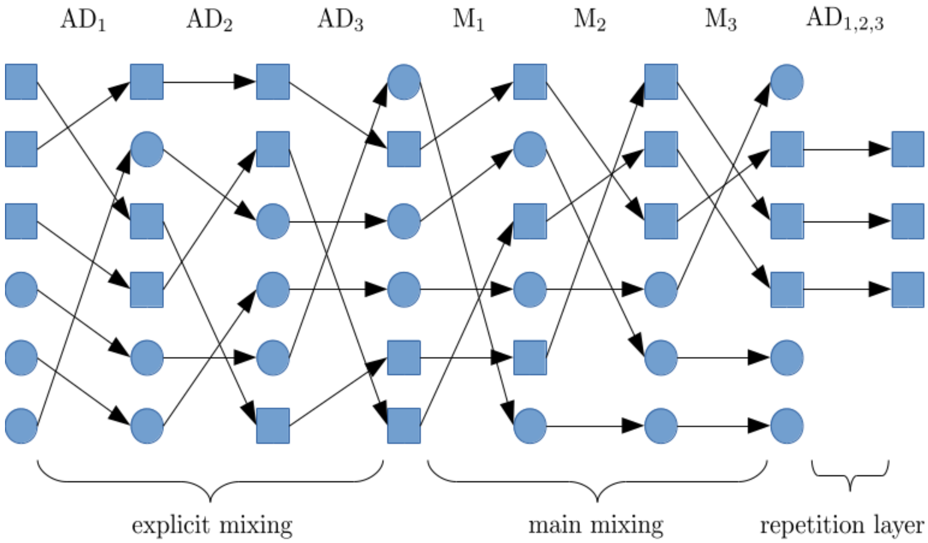

Fig. 1: Examplified run of a decryption mix net with trip wires, where $n_{\mathsf{AD}} = 3$, $n_{\mathsf{MS}} = 3$, $n_{\mathsf{S}} = 3$, and $n_{\mathsf{tw}} = 1$. Rectangles and circles symbolize senders' and auditors' message traces, respectively.

4.1 Idea

At a high level, the trip wire technique works as follows. The plain decryption mix net is extended with a number of auditors $\mathsf{AD}_1, \dots, \mathsf{AD}_{n_{\mathsf{AD}}}$ each of which executes the submission program of the senders n_{tw} times. For this purpose, AD_j chooses dummy input messages (e.g., 0^l) and encrypts them in layers as a normal user would. The resulting ciphertexts are called AD_j's *trip wires*. Furthermore, AD_j stores the random coins that it has used to generate its n_{tw} trip wires.

Now, the plain decryption mix net (with only "main mixing" servers for now) is run with this extended set of inputs. Once mixing has finished, each auditor AD_j reveals its inner states, including its trip wires' random coins. With this, the traces of AD_j's trip wires through the mix net can publicly be verified. If a mix server M_k did manipulate one of these dummy traces, this can be detected, and furthermore M_k can be held accountable through its digital signature (more on this later).

Even though this high-level description gives some intuition on the "integrity challenge" underlying the trip wires, verifiability is obviously not yet guaranteed:

1. At the start of the mix, it is clear which input ciphertexts belong to the senders and which ones to the auditors. Hence, if the first mix server M_1 is malicious, then the adversary can completely manipulate the outcome of the mix net without being detected.
2. In general, we cannot assume that the auditors are able to simulate the senders' message distribution. Therefore, realistically, the auditors' and the senders' plaintext distributions are distinguishable. Now, recall that the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ knows the final plaintext output before it publishes it. Hence, if $\mathsf{M}_{n_{\mathsf{MS}}}$ is malicious, then the adversary can undetectably manipulate the outcome of the mix net.

We propose the following additional mechanisms to address the above problems:

1. Prior to the main mixing, the input ciphertexts are "pre-mixed" using the same kind of plain decryption mix net, but now run by the auditors. This phase is called *explicit mixing* (see below for the reason). Unless all auditors are corrupted, it is no longer possible, for the original *main mixing* servers, to distinguish between the senders' ciphertexts and the auditors' trip wires.
2. An additional layer of encryption (whose private key is secret-shared among the auditors) is added directly to the plain input messages. This is called the *repetition layer*. Unless all auditors are corrupted, the last mix server gets to know only the still encrypted output.

Since secrecy of the explicit mixing and of the repetition layer is required only during main mixing, these two phases can be verified explicitly once the main mixing has finished. For this purpose, each auditor is supposed to reveal its explicit-mixing secret key as well as its secret key share of the repetition encryption layer after the final mix server has published its output.

4.2 Protocol

In this section, we precisely describe how to extend a plain decryption mix net (Section 3) with the trip wire technique.

To preserve readibility, we make the following implicit assumptions:

- Whenever a party (mix server or auditor) holding a verification/signing key pair publishes information, it signs this data with its secret signing key.
- Whenever a mix server or an auditor deviates from its honest program in an obvious way (e.g., refuses to participate, or publishes an invalid secret key), then the protocol aborts immediately and the misbehaving party is held accountable.
- In order to protect against replay attacks which may affect message privacy of senders (see, e.g., [9]), ciphertext deduplication is always in effect, where only the first instance of a multiply occurring ciphertext is retained.

Protocol participants. The set of protocol participants is extended by a number of *auditors* $\mathsf{AD}_1, \ldots, \mathsf{AD}_{n_{\mathsf{AD}}}$.

Cryptographic primitives. We additionally use an IND-CCA2-secure $(n_{\mathsf{AD}}, n_{\mathsf{AD}})$-threshold public-key encryption scheme $\mathcal{E}_{\mathsf{d}}$.[6]

Setup phase. The following additional steps are executed.

Each auditor AD_j runs the key generation algorithm of the digital signature scheme $\mathcal{S}$ to generate its public/private (verification/signing) keys. The verification keys are published on the bulletin board B.

Each auditor AD_j runs the key generation algorithm KeyGen of the public-key encryption scheme $\mathcal{E}$ to generate a public/private key pair $(\mathsf{pk}_j^{\mathsf{expl}}, \mathsf{sk}_j^{\mathsf{expl}})$, and posts the public key $\mathsf{pk}_j^{\mathsf{expl}}$ on the bulletin board B.

Each auditor AD_j runs the key share generation algorithm $\mathsf{KeyShareGen}$ of the distributed public-key encryption scheme $\mathcal{E}_{\mathsf{d}}$ to generate a public/private key share pair $(\mathsf{pk}_j^{\mathsf{rep}}, \mathsf{sk}_j^{\mathsf{rep}})$, and posts the public key share $\mathsf{pk}_j^{\mathsf{rep}}$ on the bulletin board B. From those, using the deterministic algorithm $\mathsf{PublicKeyGen}$, everyone can then compute the joint public key $\mathsf{pk}^{\mathsf{rep}}$.

Altogether, the public parameters consist of the public keys $\mathsf{pk}_1^{\mathsf{expl}}, \ldots, \mathsf{pk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}$ for the explicit decryption mix net, the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{MS}}}$ for the main decryption mix net, and the joint public key $\mathsf{pk}^{\mathsf{rep}}$ for the repetition encryption layer.

Submission phase (senders). Each sender S_i first encrypts its message m_i under the auditors' joint public key $\mathsf{pk}^{\mathsf{rep}}$:

$$\mathsf{c}_i^{\mathsf{rep}} = \mathsf{Enc}(\mathsf{pk}^{\mathsf{rep}}, m_i).$$

After that, S_i encrypts $\mathsf{c}_i^{\mathsf{rep}}$ under the mix servers' public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{MS}}}$ of the main decryption mix net in reverse order:

$$\mathsf{c}_i^{\mathsf{main}} = \mathsf{Enc}(\mathsf{pk}_1, (\ldots, \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{MS}}}, \mathsf{c}_i^{\mathsf{rep}}))).$$

[6] Note that to jointly decrypt a ciphertext in $\mathcal{E}_{\mathsf{d}}$, all secret key shares are required.

Afterwards, S_i encrypts $\mathsf{c}_i^{\mathsf{main}}$ under the auditors' public keys $\mathsf{pk}_1^{\mathsf{expl}}, \ldots, \mathsf{pk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}$ of the explicit decryption mix net in reverse order:

$$\mathsf{c}_i^{\mathsf{expl}} = \mathsf{Enc}(\mathsf{pk}_1^{\mathsf{expl}}, (\ldots, \mathsf{Enc}(\mathsf{pk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}, \mathsf{c}_i^{\mathsf{main}}))).$$

The resulting ciphertext $\mathsf{c}_i \leftarrow \mathsf{c}_i^{\mathsf{expl}}$ is S_i's input to the mix net.

Submission phase (auditors). Each auditor AD_j executes n_{tw} times the senders' submission steps described above, every time with (dummy) input message $m = 0^l$ (where l is the bit size of a sender's message). We denote AD_j's *trip wire* ciphertexts by $(\mathsf{c}_{n_{\mathsf{S}}+(j-1)\cdot n_{\mathsf{tw}}+l})_{l=1}^{n_{\mathsf{tw}}}$. Furthermore, AD_j stores the random coins that were used to generate its trip wire ciphertexts.

Mixing phase. The input to the mixing phase is $(\mathsf{c}_i)_{i \in I^{\mathsf{expl}}}$ which consists of (a subset of)[7] the n_{S} ciphertexts submitted by the senders and the $n_{\mathsf{AD}} \cdot n_{\mathsf{tw}}$ ciphertexts submitted by the auditors. Then, the overall mixing phase consists of two consecutive parts:

1. Explicit mixing: The auditors use their secret decryption keys $\mathsf{sk}_1^{\mathsf{expl}}, \ldots, \mathsf{sk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}$ to run the plain decryption mix net (Section 3) with input $(\mathsf{c}_i)_{i \in I^{\mathsf{expl}}}$. The output of this mix net is $(\tilde{\mathsf{c}}_i^{\mathsf{main}})_{i \in I^{\mathsf{main}}}$, where $I^{\mathsf{main}} \subseteq I^{\mathsf{expl}}$.

2. Main mixing: The mix servers use their secret decryption keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{MS}}}$ to run the plain decryption mix net (Section 3) with input $(\tilde{\mathsf{c}}_i^{\mathsf{main}})_{i \in I^{\mathsf{main}}}$. The output of this mix net is $(\tilde{\mathsf{c}}_i^{\mathsf{rep}})_{i \in I^{\mathsf{rep}}}$, where $I^{\mathsf{rep}} \subseteq I^{\mathsf{main}}$.

Auditing phase. Each auditor AD_j publishes its secret key $\mathsf{sk}_j^{\mathsf{expl}}$ associated to the explicit decryption mix net. With this, everyone can verify that the explicit mixing was executed correctly. If verification fails, a misbehaving auditor is identified through its signature and the whole protocol stops.

After that, each auditor AD_j publishes the random coins that it used to create its trip wires. With this, everyone can verify the integrity of trip wires' traces through the main decryption mix net. If verification fails, a misbehaving mix server is identified and the whole protocol stops.

Final decryption phase. Each auditor AD_j publishes its secret key share $\mathsf{sk}_j^{\mathsf{rep}}$ on the bulletin board B. Then, for each ciphertext $\tilde{\mathsf{c}}_i^{\mathsf{rep}}$ ($i \in I^{\mathsf{rep}}$), the decryption key share is publicly computed: $\mathsf{dec}_{j,i}^{\mathsf{rep}} \leftarrow \mathsf{DecShare}(\mathsf{sk}_j^{\mathsf{rep}}, \tilde{\mathsf{c}}_i^{\mathsf{rep}})$. After that, the decryption shares are combined to decrypt $\tilde{\mathsf{c}}_i^{\mathsf{rep}}$: $\tilde{m}_i \leftarrow \mathsf{Dec}(\mathsf{dec}_{1,i}^{\mathsf{rep}}, \ldots, \mathsf{dec}_{n_{\mathsf{AD}},i}^{\mathsf{rep}})$. Alternatively, and more efficiently if the threshold encryption scheme supports it (it normally would), the joint secret key $\mathsf{sk}^{\mathsf{rep}}$ iz explicitly reconstituted from the published secret key shares $(\mathsf{sk}_j^{\mathsf{rep}})_{j \in [n_{\mathsf{AD}}]}$ and from there using $\mathsf{sk}^{\mathsf{rep}}$ each ciphertext $\tilde{\mathsf{c}}_i^{\mathsf{rep}}$ is directly decrypted into $\tilde{m}_i$.

The list of decrypted messages $(\tilde{m}_i)_{i \in I^{\mathsf{rep}}}$ is the final outcome of the mix net.

5 Verifiability

In this section, we analyze verifiability of the decryption mix net with trip wires in the generic verifiability framework by Küsters, Truderung, and Vogt [26]. We

[7] Recall that ciphertext duplicates or invalid ciphertexts are continuously removed.

briefly recall a specific instance of their general framework (Section 5.2) that was previously applied to analyze a number of further mix nets [18, 24, 25, 27] and that we now apply to the decryption mix net with trip wires (Section 5.3).

5.1 Notation

The decryption mix net extended with the trip wire technique can be modeled in a straightforward way as a protocol $P^{\mathsf{tw}}_{\mathsf{DMN}}(n_{\mathsf{S}}, n^{\mathsf{hon}}_{\mathsf{S}}, n_{\mathsf{MS}}, n_{\mathsf{AD}}, n_{\mathsf{tw}})$, described next. The protocol participants consist of n_{S} senders (in total), $n^{\mathsf{hon}}_{\mathsf{S}}$ honest senders, n_{MS} mix servers, n_{AD} auditors, a scheduler SC, and a public append-only bulletin board B. The scheduler SC plays the role of the mix net authority and schedules all other agents in a run according to the protocol phases. We assume that SC and the bulletin board B are honest, i.e., they are never corrupted. While SC is merely a virtual entity, in reality, B should be implemented in a distributed way (see, e.g., [13, 22]). The parameter n_{tw} denotes the number of trip wires per auditor.

5.2 Verifiability Definition

Intuitively, a mix net is verifiable if an incorrect final outcome is not accepted. More precisely, an outcome of the mix net should be rejected if it does not correspond to the actual input as provided by the senders. However, such a naïve definition of verifiability would be too strong for most reasonably verifiable mix nets. Instead, the intuitive definition is judiciously adjusted as follows:

1. Completeness is relaxed such that an incorrect outcome may falsely be accepted with some (small) probability $\delta \in [0, 1]$. This parameter is called the *verifiability tolerance* of the mix net.
2. Many verifiable mix nets (besides the ones equipped with a proof of correct shuffle) do not aim to ensure that *all* input messages are reflected correctly in the final outcome but *almost of them*. Therefore, we allow for manipulating a small number of k input messages. (Typically, the verifiability tolerance $\delta = \delta_k$ decreases when k increases.)
3. Since corrupted senders may not (necessarily) complain in case their messages were dropped or manipulated by a colluding mix net authority (e.g., mix server), it is often sufficient to guarantee the integrity of the final result only w.r.t. the honest input messages (as long as no input message stuffing by dishonest senders occurs.)

These refinements lead to the following expressive, widely applicable and currently accepted definition of verifiability. Due to space limitations, we state it informally, and refer to [26] for the complete formal definition.

Definition 1 (Verifiability (informal)). *A mix net protocol P* provides (δ, k)-verifiability *if and only if an outcome of the mix net is accepted with probability at most δ in case more than k honest input messages were manipulated (or any dishonest messages were inserted).*

5.3 Verifiability Result

We are now able to precisely state the verifiability level offered by the decryption mix net with trip wires according to Definition 1. The level depends on the number of honest senders $n_{\mathsf{S}}^{\mathsf{hon}}$ and the number of dummy messages per auditor n_{tw}, as described in Section 5.1.

Assumptions. We prove the verifiability result under the following assumptions:

(V1) The public-key encryption scheme $\mathcal{E}$ is IND-CCA2-secure.
(V2) The $(n_{\mathsf{AD}}, n_{\mathsf{AD}})$-threshold public-key encryption scheme $\mathcal{E}_{\mathsf{d}}$ is IND-CCA2-secure.
(V3) The signature scheme $\mathcal{S}$ is EUF-CMA-secure.
(V4) The scheduler SC, the bulletin board B, and at least one auditor are honest.
(V5) For all honest senders and auditors, the length of the message plaintext has the same size in each run of the protocol (given a security parameter).
(V6) For $\mathcal{E}$ and $\mathcal{E}_{\mathsf{d}}$, we require that for any two plaintexts of the same length, their encryption always yields ciphertexts of the same length.

Our Result. Intuitively, the following theorem states that the probability that, in a run of the trip wire decryption mix net, more than k honest sender inputs have been manipulated, but the final result of this run is nevertheless accepted, is bounded by a function $\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{tw}})$ which we can quantify.

Theorem 1 (Verifiability). *Under the assumptions (V1) to (V6) stated above, the decryption mix net protocol with trip wires $P_{\mathsf{DMN}}^{\mathsf{tw}}(n_{\mathsf{S}}, n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{MS}}, n_{\mathsf{AD}}, n_{\mathsf{tw}})$ is $(\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{tw}}), k)$-verifiable, where*

$$\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{tw}}) = \frac{\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1}}{\binom{n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}}{k+1}}.$$

The main reasoning behind this theorem is as follows. Since the explicit mixing and the shared decryption of the repetition layer are perfectly verifiable, an adversary can only manipulate honest senders' messages in the main mix net without being detected. However, due to the IND-CCA2-security of the underlying public-key encryption schemes, the adversary has to do this manipulation "blindly" as the $n_{\mathsf{S}}^{\mathsf{hon}} + n_{\mathsf{tw}}$ ciphertexts related to the honest input parties (one ciphertext for each of the $n_{\mathsf{S}}^{\mathsf{hon}}$ honest senders plus n_{tw} ciphertexts by the honest auditor) are indistinguishable. Now, if an adversary wants to manipulate $k+1$ honest inputs, the probability that he is not caught cheating is captured by the following urn experiment. An urn contains $n_{\mathsf{S}}^{\mathsf{hon}}$ white and n_{tw} black balls, representing honest messages and trip wires respectively. Upon picking $k+1$ balls from this urn without replacement, the probability that none of the removed balls was black (i.e., no trip wire was touched) is exactly $\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1} / \binom{n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}}{k+1}$.

Importantly, for all k, the verifiability tolerance $\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{tw}})$ is bounded by $(n_{\mathsf{S}}^{\mathsf{hon}}/(n_{\mathsf{S}}^{\mathsf{hon}} + n_{\mathsf{tw}}))^{k+1}$ which converges exponentially fast to 0 in the number of manipulated honest inputs k. For example, if we choose $n_{\mathsf{tw}} = n_{\mathsf{S}}$, then the adversary's risk is more than 90% for manipulating more than 4 honest messages, and even more than 99% for manipulating more than 7 honest messages.

Theorem 1 follows immediately from the even stronger result of *accountability* which we state and formally prove in our technical report [7]. Precisely, we show that a decryption mix net with trip wires even provides *individual accountability*. This security property not only guarantees that the correctness of the mix net outcome can be verified and adjudicated externally, but also that misbehaving parties can be identified and held accountable. Since Küsters et al. [26] proved that accountability is a stronger form of verifiability, the formal proof of our accountability result [7] implies the verifiability result (Theorem 1) stated above.

6 Implementation

In terms of efficiency, the core component of the verifiable mix net protocol is the (post-quantum) IND-CCA2-secure public-key encryption scheme: this component must be fast and robust enough to process thousands, possibly millions, of untrusted encrypted ballots, and do so safely and efficiently. Decryption performance is of particular importance since each mix server will be decrypting (one layer of) the entire set of encrypted ballots, while encryption is naturally done piecemeal in a distributed way by the individual voters. Encryption performance will start to matter (for the auditors) if the number of trip wires is large, or (for the voters) if there are many mix servers hence encryption layers.

6.1 Design

We implement essentially the textbook Regev scheme (technically its dual), which is provably secure under the now-classic LWE hardness assumption [31]. Our implementation attempts to remain faithful to the theoretical scheme, but rearranges it to optimize its computation. We merely summarize the salient points in Appendix A, while referring the reader to standard texts or surveys on lattice-based cryptography for background. We also elaborate on our implementation rationale in our technical report [7], in particular on why we refrained from choosing one of the current NIST proposals.

6.2 Technical Details

The concrete IND-CCA2-secure scheme we implement is a hybrid consisting of a lattice-based CCA2-secure KEM, combined with an AES256-based DEM with MAC. The KEM closely follows the original Regev cryptosystem [31]. For efficiency, much of the secret data is obtained from privately or randomly seeded AES256-based PRNG, and likewise much of the public key is generated on the fly from a publicly seeded AES128-based PRNG. The data is aligned and ordered so as to maximize performance of decryption over that of encryption. Standard techniques are used to provide chosen-ciphertext security for each of the KEM and the DEM, albeit only *implicitly* in the sense of [6], causing malformed ciphertexts to decrypt indistinguishably randomly rather than be explicitly rejected.

Our implementation targets the 240-bit security level, and accordingly uses 240-bit or wider data paths everywhere including the KEM-crypted symmetric session key and the DEM redundancy. As stated, we erred on the side of overshooting our target, and used lattices of dimension $n = 1024$, modulus $q = 2^{16}$ and sampler-mandated LWE discrete Gaussian noise $\sigma \approx 2$, providing sufficient headroom to reliably encode 5-bit payload per 16-bit ciphertext component. These parameters are conservative but not normative, and were selected mainly for the purpose of conducting a realistic performance evaluation.

As stated in the theoretical part of the paper, the final decryption (in the repetition layer) does not need to operate as a true threshold scheme, as long as the private key can be reconstituted from the revealed private-key shares. Regev key generation supports this, by linearity of the public key in the private key. We can thus reuse the same implementation for the final layer, by letting each auditor create its own private-key share and publish the corresponding share of the public key. The "dependent part" of the public key is reconstituted as the modular sum of the public shares. The "independent part" of the public key, namely the large public matrix "A", does not need to be shared and continues to be pseudorandomly expanded from a public random seed that the auditors will have agreed on. The private-key shares eventually revealed by the auditors can be verified for correctness based on the corresponding public-key shares, before the final decryption of the repetition layer takes place.

Our implementation is completely independent and does not borrow any code from anywhere, other than a few lines for the canonical usage of AESNI.

6.3 Local-Scale Performance

Our test platform is a 2019 Dell XPS 13 Intel i7-8565U CPU, fully mitigated in microcode and OS (Linux) against all known speculative execution/loading attacks, and running a single core at 4.1GHz measured clock frequency. At the 240-bit target security level, using 1024-dimension lattices, the performance of our IND-CCA2 subsystem (assuming 240-bit canary and 16-bit payload for the DEM plaintext) is as follows:

- Public-key size: 93 kB
- Ciphertext overhead incl. canary: 2.3 kB
- Key generation time: 36 μs (0.036 s)
- Encryption time: 201 μs (0.000201 s)
- Decryption time: 133 μs (0.000133 s)

For the verifiable mix net application, except when the number of ballots is extremely small, the processing time for each mix layer will be almost entirely dominated by the time it takes to decrypt the incoming ballots. As one would expect, the total decryption time for one layer of the mix net using a single core scales almost perfectly linearly with the number of ballots (see Section 6.4), and we measure (on the same hardware as above):

- 7500 ballots in 1.02 s, or
- 1 million ballots in 132.22 s.

In practice, the decryption running time for a large number of independent ciphertexts can be divided almost exactly by the number of available CPU cores.

6.4 Whole-System Performance

The random permutation of the ballots in each layer of the mix net does not add any appreciable time to the mixing, as long as it can be assumed that the entire set fits in random-access memory (normally a reasonable assumption). Likewise, while lattice-based signatures are generally much more expensive than lattice-based encryption, the overhead of issueing a single signature on the published mix does not make any difference with a large number of ballots.

Therefore, when considering the performance of the entire mix net, the two principal factors are the sequential nature of the encryption and decryption operations (by the voter and the mix servers respectively), and the growth of the multi-layer encrypted ballot with the number of layers. Clearly, the first consideration introduces a linear factor in the total mixing time, since each mix server must finish its mixing task on the entire set of ballots before certifying the result and passing the baton to the next mix server.

The ciphertext growth is also linear in the number of layers (or equivalently, mix servers). In our implementation at 240-bit security level, each layer adds an overhead of 2.3 kB (consisting of 2.1 kB of KEM data plus 0.2 kB of redundancy, to be added to the size of the plaintext, which in every layer except the first one is the total size of the previous layer's ciphertext). In theory, this makes the total mixing time quadratic in the number n of mix servers as $n \to \infty$. In practice, however, the hybrid encryption and decryption running times are dominated by the public-key KEM component, the processing of which at each layer is independent of the size of the DEM hence the number of layers.

Our experiments (Table 1) show the evolution of encryption and decryption running time of one layer of the "onion" or encrypted ballot, in function of the number of layers of encryption beneath it (level 0 indicates direct encryption of the plaintext vote, while level 1,000,000 is clearly impractical and provided only to show asymptotic behavior).

In practice, each layer corresponds to a different mixing server, so the total number of layers will likely remain small (less or much less than 100). Nevertheless, the experiments show that encryption and decryption times remain essentially constant (per layer) far beyond the range of practical applications, and that it is the size of the encrypted onions, rather than the time to encrypt or decrypt them, that is likely to be a limiting factor. The asymptotic linearity of encryption and decryption times (for each layer) only starts to show at very high numbers of layers. We also note that only the total number of layers and

Table 1: Encryption/decryption times and ciphertext size in function of layer height.

# layers	ctx size (kB)	encrypt time	decrypt time
0	2,144	201 us	133 us
1	4,256	201 us	134 us
10	23,264	209 us	141 us
30	65,504	214 us	154 us
100	213,344	254 us	194 us
300	635,744	368 us	308 us
1,000	2,114,144	792 us	753 us
1,000,000	2,112,002,144	0.641 s	0.607 s

the total number of ciphertexts will matter, in terms of performance. How these are partitioned between explicit and main mixers, as well as between actual and trip wire ballots, has no significant impact on running time.

On the voter's size, encrypting a complete onion even for an exceedingly large 1000-layer mixnet would still require less than one second on most modern commodity consumer hardware.

7 Discussion

In this section, we discuss the main properties of the decryption mix net with trip wires.

Verifiability and Accountability. We have formally proven that, even if all mix servers are malicious, an adversary's risk of being caught cheating is high.

More precisely, our accountability result implies that, if an adversary wants to manipulate more than k honest inputs, then (at least) one misbehaving mix server is identified with probability at least $1 - (n_{\mathsf{S}}^{\mathsf{hon}}/(n_{\mathsf{S}}^{\mathsf{hon}} + n_{\mathsf{tw}}))^{k+1}$, where $n_{\mathsf{S}}^{\mathsf{hon}}$ is the given number of honest senders and n_{tw} is the given number of trip wires per auditor. In particular, an adversary knows upfront that its risk of being caught cheating converges exponentially fast against 1 in the number of manipulated messages k.

Moreover, recall that during the main mixing, both the explicit mixing and the repetition layer are still locked. Hence, even if the race between two candidates A and B was very close, an adversary trying to manipulate the election outcome in favor of A by swapping just a few votes from B to A, has to do this "blindly". In particular, the adversary may accidentally swap a message from A to A. Hence, an adversary's chance of successfully manipulating the outcome is significantly reduced, independently of whether the adversary is caught cheating or not.

Altogether, for applications like secure e-voting where misbehaving parties have to face severe financial or legal penalties, an adversary knows a priori that manipulating the mix net outcome would be completely unreasonable.

External auditing. At a high level, the verification procedure of the trip wire mix net can be regarded as an "integrity experiment" that is run between an adversary (controlling all mix servers) and an external auditor who challenges the adversary by "injecting" trip wires. If the adversary is able to manipulate (a significant number of) honest inputs without touching one of the trip wires, then the adversary wins. Our verifiability/accountability result (see above) provides an upper bound for an adversary's advantage in this experiment.

Obviously, the external auditor needs to be trusted for the integrity experiment but this trust assumption is mitigated by two means. First, the auditor's role can easily be distributed among several auditors, only one of which needs to be trusted. Second, the auditor opens its complete inner states once the integrity challenge has finished so that the correctness of its internal computation can publicly be verified.

Privacy. The original purpose of employing a mix net is to break the individual links between the senders and their plain input messages. This property is called *(message) privacy.* Assuming one honest mix server and one honest auditor, the trip wire mix net guarantees privacy. A formal proof of this statement can be based on a sequence of games similar to the one of our accountability proof.

Post-quantum practicality. We experimentally benchmarked our verifiable mix net scheme using an optimized post-quantum IND-CCA2-secure hybrid encryption scheme, consisting of a lattice-based CCA2-secure KEM, combined with an AES256-based DEM/MAC. The benchmarks on our prototype demonstrate that our verifiable mix net with trip wires is highly practical, even for large-scale elections run entirely on commodity hardware.

Example: Practical PQ-secure e-voting. We now demonstrate how to put all these pieces together. For this purpose, we consider two different kinds of elections, one with few and one with many voters. Clearly, for an election with few voters, manipulating just a single message can have a major impact on the election result with significant probability, whereas this is much less likely for an election with many voters. In what follows, we exemplify how the decryption mix net with trip wires can be set up to take this aspect into account.

Assume we have one election with 100 and one with 100,000 voters. We choose $n_{\mathsf{tw}} = 100,000$ for both elections. (For the sake of simplicity, we assume that all voters are honest, i.e., $n_{\mathsf{S}} = n_{\mathsf{S}}^{\mathsf{hon}}$.) From the verifiability theorem, it follows that the risk of being caught cheating is $\geq 99\%$ both in the election with 100 voters for manipulating $k \geq 1$ votes, and in the election with 100,000 voters for manipulating $k \geq 7$ votes. Therefore, in both cases, an adversary knows upfront that tampering *significantly* with the election result is extremely risky.

At the same time, our benchmarks demonstrate that increasing n_{tw}, and hence tightening the verifiability tolerance, is practically negligible for applications like secure e-voting where the tallying phase is typically not too time-critical.

8 Conclusion

We have presented the first practical and verifiable lattice-based decryption mix net with external auditing which can be dropped into existing e-voting schemes. Our mix net is fully implemented and supports arbitrarily many authorities.

Acknowledgements

All authors acknowledge support from the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint INTER project SURCVS (Number 11747298). Xavier Boyen thanks the Australian Research Council for support as Future Fellow under ARC grant FT140101145.

Bibliography

[1] Ben Adida and Douglas Wikström. How to Shuffle in Public. In *TCC 2007, Proceedings*, pages 555–574, 2007.

[2] Ben Adida and Douglas Wikström. Offline/Online Mixing. In *ICALP 2007, Proceedings*, pages 484–495, 2007.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A. Buell, et al. Quantum Supremacy using a Programmable Superconducting Processor. *Nature*, 574(7779):505–510, 2019.

[4] Stephanie Bayer and Jens Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 2012.

[5] Baum C., Bootle J., Cerulli A., del Pino R., Groth J., and Lyubashevsky V. Sub-Linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits. In *CRYPTO 2018*, volume 10992 of *Lecture Notes in Computer Science*. Springer, 2018.

[6] Xavier Boyen. Miniature CCA2 PK Encryption : Tight Security Without Redundancy. In *ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2007.

[7] Xavier Boyen, Thomas Haines, and Johannes Mueller. A Verifiable and Practical Lattice-Based Decryption Mix Net with External Auditing. *IACR Cryptology ePrint Archive*, 2020:115, 2020.

[8] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[9] Véronique Cortier and Ben Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *IEEE CSF, 2011*, pages 297–311, 2011.

[10] Núria Costa, Ramiro Martínez, and Paz Morillo. Proof of a Shuffle for Lattice-Based Cryptography. In *NordSec 2017, Proceedings*, pages 280–296, 2017.

[11] Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-Based Proof of a Shuffle. *IACR Cryptology ePrint Archive*, 2019:357, 2019.

[12] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[13] Chris Culnane and Steve A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE CSF 2014*, pages 169–183, 2014.

[14] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An Efficient Pairing-Based Shuffle Argument. In *ASIACRYPT 2017, Proceedings, Part II*, pages 97–127, 2017.

[15] Prastudy Fauzi, Helger Lipmaa, and Michal Zajac. A Shuffle Argument Secure in the Generic Model. In *ASIACRYPT 2016, Proceedings, Part II*, pages 841–872, 2016.

[16] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO '99, Proceedings*, pages 537–554, 1999.

[17] Jun Furukawa and Kazue Sako. An Efficient Scheme for Proving a Shuffle. In *CRYPTO 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2001.

[18] Thomas Haines and Johannes Müller. SoK: Techniques for Verifiable Mix Nets. In *IEEE CSF 2020*, to appear, 2020.

[19] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Linearly-Homomorphic Signatures and Scalable Mix-Nets. *IACR Cryptology ePrint Archive*, 2019:547, 2019.

[20] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making Mix Nets Robust for Electronic Voting by Randomized Partial Checking. In *USENIX Security Symposium, 2002*, pages 339–353, 2002.

[21] Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In *ASIACRYPT 2012, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[22] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the Security Properties of e-Voting Bulletin Boards. In *SCN 2018, Proceedings*, pages 505–523, 2018.

[23] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE SP 2019*, pages 1–19, 2019.

[24] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE CSF 2016*, pages 341–354, 2016.

[25] Ralf Küsters and Tomasz Truderung. Security Analysis of Re-Encryption RPC Mix Nets. In *IEEE EuroS&P 2016*, pages 227–242, 2016.

[26] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: Definition and Relationship to Verifiability. In *ACM CCS 2010*, pages 526–535, 2010.

[27] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Formal Analysis of Chaumian Mix Nets with Randomized Partial Checking. In *IEEE SP 2014*, pages 343–358, 2014.

[28] Helger Lipmaa and Bingsheng Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In *SCN 2012. Proceedings*, pages 477–502, 2012.

[29] C. Andrew Neff. A Verifiable Secret Shuffle and its Application to E-Voting. In *ACM CCS 2001*, pages 116–125. ACM, 2001.

[30] Choonsik Park, Kazutomo Itoh, and Kaoru Kurosawa. Efficient Anonymous Channel and All/Nothing Election Scheme. In *EUROCRYPT '93, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 248–259. Springer, 1993.

[31] Oded Regev. On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, 2005*, pages 84–93, 2005.

[32] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C, 2nd Edition.* Wiley, 1996.

[33] Martin Strand. A Verifiable Shuffle for the GSW Cryptosystem. In *FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Revised Selected Papers*, pages 165–180, 2018.

[34] Björn Terelius and Douglas Wikström. Proofs of Restricted Shuffles. In *AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2010.

[35] Verificatum Mix Net (VMN). `https://www.verificatum.org/html/product_vmn.html`.

[36] Douglas Wikström. A Sender Verifiable Mix-Net and a New Proof of a Shuffle. In *ASIACRYPT 2005, Proceedings*, pages 273–292, 2005.

[37] Douglas Wikström. A Commitment-Consistent Proof of a Shuffle. In *ACISP 2009, Proceedings*, pages 407–421, 2009.

[38] Douglas Wikström and Jens Groth. An Adaptively Secure Mix-Net Without Erasures. In *ICALP 2006, Proceedings, Part II*, pages 276–287, 2006.

A Optimizations

As mentioned in Section 6, our implementation attempts to remain faithful to Regev's theoretical scheme [31], but rearranges it to optimize its computation. In what follows, we summarize the salient points.

Our first optimization, which does deviate from the theoretical scheme, is, rather than to publish the encryption key as a truly random matrix, we publish a random seed from which the key is pseudo-randomly generated it using AES. This is a trick used by several NIST submissions, including the "front runners" still in play, but we have the opportunity to do it much faster without function calls as explained in our technical report [7].

We also mentioned the use of a strictly data-independent integer Gaussian sampler for generating the secret LWE noise. Using the Central Limit Theorem, we build a novel circuit-based sampler, which, when paired with hardware-accelerated AES, is able to produce i.i.d. integer samples of zero mean and small fixed variance (e.g., $\sigma \approx 2$) with provable 64-bit or 128-bit accuracy, suitable as LWE noise, in a few clock cycles. [8] For comparison, we note that FrodoKEM which also implements plain-LWE Regev encryption, samples from a cumulative probability table of about 20-bit effective accuracy, and goes to lengths to show that this is okay. Our equally fast sampler is far more accurate, and closely matches the theoretical Regev scheme which requires high accuracy. It is also data-independent (unlike table lookups whose access patterns could lead to certain cache-based side-channel leakage). The main downside of our sampler is that it is highly inflexible and specifically suited for that particular usage. [9]

Another extension to the textbook Regev scheme that we make, is the addition of an "all-or-nothing" transform such as [16] to obtain chosen-ciphertext security, as is standard practice. Unlike [16], though, our all-or-nothing transform does not cause invalid ciphertexts to be *rejected*, but only *scrambled* (or randomized), as proposed in [6]. We do this to ensure that there truly is no data-dependent test anywhere in the crypto code. We still get true CCA2 security, and we can recover the classic explicit rejection behavior simply by adding and testing a known string such as 0^λ to the plaintext, i.e., outside of the crypto code, to act as a "canary".

Other that those differences, the mathematical functions computed by our implementation are functionally very similar to the NIST submission FrodoKEM, which both implement the Regev scheme. This allows us to borrow from its extensive security analyses and use similar lattice dimension parameters to target similar security levels. In particular, we were pleasantly surprised that the FrodoKEM designers *chose* a Gaussian noise variance parameter close to that which was *forced* on us by our optimized but inflexible sampler circuit design—making their analysis a good match for our implementation. Nevertheless, to err on the side of caution, we collected lattice hardness estimates from multiple sources and, seeing that they loosely agreed with the FrodoKEM recommendations, we still rounded up the main lattice dimension to the higher power of 2. Minor optimizations included selecting the modulus $q = 2^{16}$ "`sizeof(short)`" for its ability to give us vectorized (SIMD) modular reductions for free. [10]

We reiterate that our optimizations mostly affect not *what* we compute but *how* we compute it. Unbound from the NIST rules, our code is not only faster, but also safer, not in a cryptographic sense but against side-channel attacks. None of our code borrows from the NIST contest; we merely frame this discussion in relation with NIST to preempt any preconception than official standardization would necessarily produce an optimal outcome.

[8] Sampling accuracy is here meant in the sense of KL divergence to a true integer Gaussian; clearly the output itself is just a small integer that fits in a few bits.

[9] Describing and analyzing the sampler is very much out of the scope of this paper, but it is one example of a very impactful optimization we could make that does not involve *what* we compute, only *how* we do it.

[10] FrodoKEM had nearly the same idea, but for reasons unclear chose $q = 2^{15}$ not 2^{16}, perhaps because they could not use x86_64 vectorization intrinsics.