SoK: Techniques for Verifiable Mix Nets

Thomas Haines
Norwegian University of Science and Technology, Norway
thomas.haines@ntnu.no

Johannes Müller
SnT/University of Luxembourg, Luxembourg
johannes.mueller@uni.lu

***Abstract*—Since David Chaum introduced the idea of mix nets 40 years ago, they have become widely used building blocks for privacy-preserving protocols. Several important applications, such as secure e-voting, require that the employed mix net be *verifiable*. In the literature, numerous techniques have been proposed to make mix nets verifiable. Some of them have also been employed in politically binding elections.**

Verifiable mix nets differ in many aspects, including their precise verifiability levels, possible trust assumptions, and required cryptographic primitives; unfortunately, these differences are often opaque, making comparison painful.

To shed light on this intransparent state of affairs, we provide the following contributions. For each verifiability technique proposed to date, we first precisely describe how the underlying basic mix net is to be extended and which (additional) cryptographic primitives are required, and then study its verifiability level, including possible trust assumptions, within one generic and expressive verifiability framework. Based on our uniform treatment, we are able to transparently compare all known verifiability techniques for mix nets, including their advantages and limitations.

Altogether, our work offers a detailed and expressive reference point for the design, employment, and comparison of verifiable mix nets.

I. Introduction

Mix nets are popular building blocks for privacy-preserving technologies, most prominently for secure e-voting systems. For example, mix nets have been employed in real political elections in Norway, Estonia, Switzerland and Australia [10, 14, 21, 38, 42, 43, 47]. Further applications include, but are not limited to, anonymous messaging [3, 33, 40], anonymous routing [9], and oblivious RAM [46].

On a high level, a mix net is run among a set of senders and a set of mix servers, and works as follows. Each sender provides its input to the mix servers who then privately shuffle all inputs and eventually publish them in random order. Unless all mix servers are corrupted, a mix net should guarantee that the individual links between the senders and their messages in the output remain secret. In the context of e-voting, where the senders are voters and their messages represent ballots, this property preserves vote privacy.

However, "plain" mix nets should not be used when misbehaving mix servers are a real threat; they are only suitable in the honest-but-curious model. In fact, for applications like secure e-voting, the employed mix net should also be *verifiable* to guarantee that if something goes wrong (e.g., the final result does not correspond to the submitted ballots), then this can be detected. Often, in order to deter parties from misbehaving, a stronger form of verifiability, *accountability*, is required to ensure that misbehaving parties can even be identified.

In the literature, numerous mix nets [1, 2, 4, 11, 12, 16–18, 20, 23–25, 28, 35, 36, 41, 44, 45, 48–51] have been proposed to date that aim to achieve verifiability and even accountability. However, these mix nets differ in several important aspects, including but not limited to:

- *Verifiability level:* Many mix nets [1, 2, 4, 11, 12, 16–18, 20, 23, 35, 36, 44, 45, 48, 50, 51] aim to guarantee a "perfect" verifiability level: even if only a single message is manipulated, then this will be detected with overwhelming probability. On the other hand, several mix nets [7, 24, 25, 28, 41, 49] were designed to guarantee a "relaxed" verifiability level, where some (small amount of) manipulations may not always be detectable, but which opens up advantages in other aspects (see next points).
- *Trust assumptions:* Ideally, verifiability and accountability should be guaranteed without any (unnecessary) trust assumptions. However, some mix nets are only verifiable if certain parties are (at least temporarily) trusted. While for some of these mix nets, the required trust assumptions are straightforward to see or made explicit (e.g., [25]), identifying them is non-trivial for others (e.g., it is unclear how to verifiably generate the common reference string in [17]).
- *Cryptographic primitives:* All mix nets that aim for a perfect verifiability level employ specifically tailored cryptographic primitives. This can be disadvantageous, for example in the following aspects. First, it is challenging to implement these primitives correctly, as recently demonstrated for the mix net employed in the Swiss e-voting system [13]. Second, the security of these techniques typically relies upon "traditional" hardness assumptions so that privacy (e.g., of voters) may retrospectively be broken in the future with quantum computers. On the other hand, some verifiable mix nets [7, 24, 28, 41, 49] employ only basic cryptographic primitives.

This confusing situation raises several questions: Which verifiability levels do the different verifiable mix nets provide and how do these levels relate? Which trust assumptions are made, possibly implicitly? Which cryptographic primitives are conceptually required? Which verifiable mix nets can be instantiated using practical post-quantum cryptographic primitives only? How complex are the different techniques computationally? Which guarantees do they provide in terms of message privacy? Answering these questions is non-trivial. It requires some common basis on which the resulting mix nets can be modeled and on which their security and computational complexity can be analyzed and compared.

Our contributions. To shed light on this intransparent state of affairs, we provide the following contributions:

1) For each verifiable mix net from the literature (see above), we distilled its underlying "atomic" verifiability technique(s). While some of the mix nets employ a single technique only (e.g., [32]), others combine two or more of them (e.g., [25, 28]). By this, we can study all atomic techniques independently in the next steps.
2) For each verifiability technique, we precisely describe how the protocol of the underlying basic mix net is to be extended and which additional cryptographic primitives are required.
3) We use the general verifiability/accountability framework by Küsters et al. [30] to study verifiability and accountability of each technique. This framework is particularly suitable for our purposes as it *measures* the verifiability/accountability level a mix net provides and makes trust assumptions (if any) transparent. Furthermore, some mix nets have already been analyzed in this framework before [7, 28, 29, 32]. The verifiability/accountability levels of the remaining ones follow either immediately from their specific properties (in the case of proofs of correct shuffle) or are formally analyzed in this work (see next point).
4) We provide the first formal verifiability and accountability analysis of the Khazaei-Moran-Wikström mix net [25]. Our result refines the security theorem that was stated by Khazaei et al. [25] but for which a proof has not been published prior to our work.
5) Based on the uniform and transparent treatment in the previous steps, we elaborate on the advantages, disadvantages, problems, and limitations of the various verifiability techniques. In particular, we identified several fundamental issues that have not been mentioned prior to our work, and show how to fix them (if possible).

Altogether, our work offers a detailed and transparent reference point for the design, employment, and comparison of verifiable mix nets.

Scope of our contributions. We have focused on surveying and analyzing the underlying techniques for *synchronous* (linear) mix nets which are particularly important for building secure e-voting systems. We do not cover continuous mixers, such as Loopix [40], nor systems which, though sometimes called mix nets, more properly extend mix nets into a dynamic system across an extended period of time, such as Miranda [34].

Structure of the paper. In Section II, we present the basic design of all verifiable mix nets, distinguishing between decryption mix nets (DMN) and re-encryption mix nets (RMN). In Section III, we introduce the computational model to later model the different verifiability techniques, and in Section IV, we introduce the general verifiability/accountability framework to formally analyze them.

Subsequently, each of the Sections V to X is dedicated to one of the verifiability techniques: we first explain how the underlying basic mix net from Section II is to be extended, and then elaborate on important properties, focussing on verifiability and accountability. The formal results are summarized in Table I (Primitives & Verifiability). Our main insights are distilled in Section XIII where we elaborate on the relations between the different verifiability techniques, their advantages, and limitations.

II. Basic Mix Nets

Secure mix nets can be classified into two categories: *decryption mix nets (DMN)* and *re-encryption mix nets (RMN)*. Originally, the concept of a DMN was proposed by Chaum [8], and the one of a RMN by Park et al. [39]. In Sections II-B and II-C, we describe the design of a plain (i.e., non-verifiable) DMN and RMN, respectively. In order to make these mix nets verifiable, different techniques have been proposed which we systematically study in this paper (Sections V-XI). Independently of the specific verifiability technique, the resulting verifiable DMNs and RMNs have a specific structure that we describe in Section II-D and that we call *basic DMN* and *basic RMN*, respectively.

A. General Structure

We start by describing the general structure of a mix net.

Protocol participants. A mix net protocol is run among the *senders* $\mathsf{S}_1, \ldots, \mathsf{S}_{n_\mathsf{S}}$, the *mix servers* $\mathsf{M}_1, \ldots, \mathsf{M}_{n_\mathsf{MS}}$, and a public, append-only *bulletin board* B. Re-encryption mix nets also include a number of trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_\mathsf{T}}$.

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.

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. Optionally, re-encryption mix nets also include a phase for *decryption*.

B. Plain DMN

The main idea of a DMN is as follows. Each sender S_i iteratively encrypts its plain input message m_i under the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_\mathsf{MS}}$ of the mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_\mathsf{MS}}$ in reverse order, and submits the resulting "nested" ciphertext c_i to the first mix server M_1. The first mix server M_1 uses its secret key sk_1 to decrypt the outermost encryption layer of all input ciphertexts, shuffles the decrypted messages, and forwards them to the second mix server M_2. The second mix server M_2 uses its secret key sk_2 to decrypt the next encryption layer, shuffles the result, and so on. Eventually, the last mix server $\mathsf{M}_{n_\mathsf{MS}}$ outputs the plain messages initially chosen by the senders in random order.

Cryptographic primitives. We use the following cryptographic primitives:

- An IND-CCA2-secure public-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$.
- An EUF-CMA-secure signature scheme $\mathcal{S}$.

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.[1]

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

1) Decrypt all ciphertexts in C_{k-1} under private key sk_k: $C'_k[i] \leftarrow \mathsf{Dec}(\mathsf{sk}_k, C_{k-1}[i])$ for all $i \in \{1, \ldots, n_{\mathsf{S}}\}$.
2) Choose a permutation σ_k over $\{1, \ldots, n_{\mathsf{S}}\}$ uniformly at random, and set $C_k[\sigma(i)] \leftarrow C'_k[i]$ for all $i \in \{1, \ldots, n_{\mathsf{S}}\}$.
3) Send C_k to 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 $(m_{\sigma(i)})_{i=1}^{n_{\mathsf{S}}}$, where $\sigma = \sigma_{n_{\mathsf{MS}}} \circ \ldots \circ \sigma_1$ is the overall permutation of the mix net.

C. Plain RMN

The main idea of a RMN is as follows. We use a public-key encryption scheme that allows for re-encrypting a given ciphertext without knowing the secret key or the encrypted message. Now, each sender S_i encrypts its plain input message m_i under a single public key pk whose secret key is typically shared among a number of trustees (e.g., the mix servers themselves). The first mix server M_1 re-encrypts all input ciphertexts (using random coins chosen independently and uniformly at random), shuffles the re-encrypted ciphertexts, and forwards them to the second mix server M_2. The second mix server re-encrypts these ciphertexts again, shuffles them, and so on. Eventually, the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ outputs a list of ciphertexts which encrypt the input messages initially chosen by the senders but under different random coins and in random order. Optionally, these ciphertexts can be decrypted by the trustees who hold the secret key shares.

Cryptographic primitives. We use the following cryptographic primitives:

- A distributed IND-CPA-secure public-key encryption scheme $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{DecShare}, \mathsf{Dec})$ which allows for re-encryption ReEnc.
- An EUF-CMA-secure signature scheme $\mathcal{S}$.

Setup phase. The mix servers and trustees generate and publish their verification keys in the same way as the mix servers do in the DMN.

Each trustee T_l runs the key share generation algorithm $\mathsf{KeyShareGen}$ of the distributed public-key encryption scheme $\mathcal{E}$ to generate its public/private (encryption/decryption) key share pair $(\mathsf{pk}_l, \mathsf{sk}_l)$, and posts its public key share pk_l on the bulletin board B. With $\mathsf{PublicKeyGen}$, everyone can then compute the (overall) public key pk.

Submission phase. Each sender S_i encrypts its secret input m_i under the trustees' joint public key pk: $\mathsf{c}_i = \mathsf{Enc}(\mathsf{pk}, m_i)$.

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

1) Re-encrypt all ciphertexts in C_{k-1} under random coins $r_k^1, \ldots, r_k^{n_{\mathsf{S}}}$ chosen uniformly at random: $C'_k[i] \leftarrow \mathsf{ReEnc}(r_k^i, C_{k-1}[i])$ for all $i \in \{1, \ldots, n_{\mathsf{S}}\}$.
2) Choose a permutation σ_k over $\{1, \ldots, n_{\mathsf{S}}\}$ uniformly at random, and set $C_k[\sigma(i)] \leftarrow C'_k[i]$ for all $i \in \{1, \ldots, n_{\mathsf{S}}\}$.
3) Send C_k to 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 $(\mathsf{c}'_{\sigma(i)})_{i=1}^{n_{\mathsf{S}}}$, where $\sigma = \sigma_{n_{\mathsf{MS}}} \circ \ldots \circ \sigma_1$ is the overall permutation of the mix net and $\mathsf{c}'_{\sigma(i)}$ is the overall re-encryption of S_i's input ciphertext c_i.

Decryption phase (optional). For every ciphertext $C_{n_{\mathsf{MS}}}[i]$ in the output of the mixing phase $C_{n_{\mathsf{MS}}}$, each trustee T_l uses its secret key share sk_l to compute a decryption share $\mathsf{dec}_{i,l} \leftarrow \mathsf{DecShare}(\mathsf{sk}_l, C_{n_{\mathsf{MS}}}[i])$, and publishes $\mathsf{dec}_{i,l}$ on the bulletin board B. With Dec, everyone can then decrypt $C_{n_{\mathsf{MS}}}[i]$.

D. Basic DMN and RMN

We describe the basic structures of all verifiable DMNs and RMNs extending the plain versions described above. These *basic DMN* and *basic RMN* can be regarded as the "greatest common divisors" of all verifiable DMNs and RMNs.

First of all, we note that each verifiable DMN or RMN also includes a phase for *auditing*, where everyone can perform the required checks to guarantee that the output is correct (provided the trust assumptions hold). The auditing phase always includes checking whether a mix net authority (e.g., mix server) deviated from its honest program in an obvious, i.e., trivially detectable, way (e.g., refuses to participate). In such a case, the protocol aborts immediately and the misbehaving party is held accountable.

Basic DMN. The plain DMN (Section II-B) is extended as follows. Ciphertext duplicates are continuously removed. In particular, if the input of a sender S_i contains a ciphertext that was already submitted before, then S_i is held accountable, and if C_k contains duplicates for some $k < n_{\mathsf{MS}}$, then M_k is held accountable.

[1] In what follows, we implicitly assume that whenever a party (e.g., M_k) holding a verification/signing key pair publishes information, it signs this data with its secret signing key.

Basic RMN. The plain RMN (Section II-C) is extended as follows. Ciphertext duplicates are continuously removed, and, additionally, each sender S_i provides a NIZKP of plaintext knowledge for her input ciphertext. Input ciphertexts without valid NIZKPs are removed.

In the setup phase, each trustee T_l provides a NIZKP for proving knowledge and correctness of its secret key share sk_l.

In the decryption phase (if any), each trustee T_l provides a NIZKP for proving correctness of its decryption shares $\mathsf{dec}_{1,l}, \ldots, \mathsf{dec}_{n_\mathsf{T},l}$.

III. Protocol Model

The general computational model that we use follows the one in [30]. This model introduces the notions of processes, protocols, and instances, which we briefly recall. In this way, we then model the basic mix net protocols which can easily be extended to also capture the more complex mix nets that we study later in this paper.

Process. A *process* is a set of probabilistic polynomial-time (ppt) interactive Turing machines (ITMs, also called *programs*) which are connected via named tapes (also called *channels*). We write a process π as $\pi = p_1 \| \cdots \| p_l$, where $p_1, \ldots, p_l$ are programs. If π_1 and π_2 are processes, then $\pi_1 \| \pi_2$ is a process, provided that the processes have compatible interfaces. A process π where all programs are given the security parameter 1^ℓ is denoted by $\pi^{(\ell)}$. In the processes we consider, the length of a run is always polynomially bounded in ℓ. Clearly, a run is uniquely determined by the random coins used by the programs in π.

Protocol. A *protocol* P is defined by a finite set of agents Σ (also called *parties* or *protocol participants*), and for each agent $\mathsf{a} \in \Sigma$ its *honest program* $\hat{\pi}_\mathsf{a}$, i.e., the program this agent is supposed to run. Agents are pairwise connected by tapes/channels and every agent has a channel to the adversary (see below). If $\hat{\pi}_{\mathsf{a}_1}, \ldots, \hat{\pi}_{\mathsf{a}_l}$ are the honest programs of the agents of P, then we denote the process $\hat{\pi}_{\mathsf{a}_1} \| \ldots \| \hat{\pi}_{\mathsf{a}_l}$ by $\hat{\pi}_P$.

The process $\hat{\pi}_P$ is always run with an *adversary* A, an arbitrary ppt program with channels to all protocol participants in $\hat{\pi}_P$. For any program π_A run by the adversary, we call $\hat{\pi}_P \| \pi_\mathsf{A}$ an *instance* of P. Now, a *run r of P* with the adversary π_A is a run of the process $\hat{\pi}_P \| \pi_\mathsf{A}$. We consider $\hat{\pi}_P \| \pi_\mathsf{A}$ to be part of the description of r so that it is always clear to which process, including the adversary, the run r belongs to.

We say that an agent a is *honest in a protocol run* r if the agent has not been corrupted in this run: an adversary π_A can corrupt an agent by sending a $\mathsf{corrupt}$ message; once corrupted, an adversary has full control over an agent. For the mix nets protocols studied in this paper, we assume *static corruption*, i.e., agents can only be corrupted at the beginning of a run. In particular, the corruption status of each party is determined at the beginning of a run and does not change during a run. Also, for some agents, we will assume that they cannot be corrupted (see below).

Modeling of basic mix nets. A basic mix net protocol can be modeled in a straightforward way either as a protocol $P_\mathsf{DMN}(n_\mathsf{S}, n_\mathsf{MS})$ (DMN) or a protocol $P_\mathsf{RMN}(n_\mathsf{S}, n_\mathsf{MS}, n_\mathsf{T})$ (RMN). The protocol participants consist of n_S senders, n_MS mix servers, n_T trustees (RMN), 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., [15, 27]).

Using the different verifiability techniques presented in this paper, we will then obtain specific mix net protocols extending the basic mix net protocols modeled above.

IV. Verifiability & Accountability

Our systematic comparison of the different mix nets in terms of verifiability and accountability uses the generic verifiability and accountability framework by Küsters, Truderung, and Vogt [30]. We briefly recall these frameworks in Section IV-A and IV-B, respectively. These frameworks are particularly suitable for our purposes because (i) they do not require a specific mix net structure, (ii) they make trust assumptions (if any) transparent, and (iii) they *measure* the level of verifiability/accountability a mix net provides.

A. Verifiability Framework

Intuitively, a mix net is verifiable if an incorrect final outcome is accepted only with small probability $\delta \in [0, 1]$.

Judge. To model whether the final outcome of a protocol run should be accepted, the verifiability definition by Küsters et al. assumes an additional protocol participant J, called the *judge*. The judge can be thought of as a "virtual" entity; in reality, the program of J can be carried out by any party, including external observers or the senders themselves, since its input is merely public information. On a high level, the judge performs certain checks to ensure the correctness of the final outcome (e.g., verifying all zero-knowledge proofs). Typically, as for all the mix nets in this paper, the program of J follows immediately from the protocol description. Formally, to either accept or reject a protocol run, the judge writes accept or reject on a dedicated channel $\mathsf{decision}_\mathsf{J}$.

Goal. To specify which runs are "correct" in some protocol-specific sense, Küsters et al. use the notion of a goal γ. Formally, a goal γ is simply a set of protocol runs. For mix nets, γ would contain those runs where the announced mix net result corresponds to the actual messages of the senders.

In what follows, we describe the goal $\gamma(k, \varphi)$ that we use to analyze all the different mix nets. This goal has already been applied in [28, 29, 32] to analyze some of the presented mix nets. The parameter φ is a Boolean formula that describes which protocol participants are assumed to be honest in a run, i.e., not corrupted by the adversary. On a high level, the parameter k denotes the maximum number of messages submitted by the honest senders that the adversary is allowed to manipulate. So, roughly speaking, the goal $\gamma(k, \varphi)$ consists of those runs of a mix net protocol P where either (i) φ is false or (ii) where φ holds true and where the adversary has manipulated at most k messages of honest senders. We recall the formal definition of $\gamma(k, \varphi)$ in Appendix A.

Verifiability. Now, the idea behind the verifiability definition is very simple. The judge J should accept a protocol run only if the goal γ is met: as discussed, if we use the goal $\gamma(k, \varphi)$, then this essentially means that the published mix net result corresponds to the actual messages of the senders up to k messages of honest senders. More precisely, the definition requires that the probability (over the set of all protocol runs) that the goal γ is not satisfied but the judge nevertheless accepts the run is δ-bounded.[2] Certainly, $\delta = 0$ is desirable but it can only be achieved by one of the presented types of mix nets (namely, RMN with a proof of correct shuffle, Section XI), however at the cost of other properties. Hence, if we strictly required $\delta = 0$, then this would deem many reasonable mix nets insecure even though they provide good (but not perfect) levels of verifiability, e.g., for some δ_k that converges exponentially fast against 0 in the number of manipulated inputs k. The parameter δ is called the *verifiability tolerance* of the protocol.

By $\Pr[\pi^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ we denote the probability that π, with security parameter 1^ℓ, produces a run which is not in γ but nevertheless accepted by J.

Definition 1 (Verifiability): Let P be a protocol with the set of agents Σ. Let $\delta \in [0, 1]$ be the tolerance, $\mathsf{J} \in \Sigma$ be the judge, and γ be a goal. Then, we say that *the protocol P is (γ, δ)-verifiable by the judge* J if for all adversaries π_A and $\pi = (\hat{\pi}_P \| \pi_\mathsf{A})$, the probability $\Pr[\pi^{(\ell)} \mapsto \neg\gamma, (\mathsf{J}\colon \mathsf{accept})]$ is δ-bounded as a function of ℓ.[3]

B. Accountability

Verifiability merely requires that, if some goal of the protocol is not achieved, then the run is rejected, but it does not guarantee that the responsible parties can be identified in such a case. However, being able to single out misbehaving parties is important in practice since malicious behaviour should have consequences, in particular to deter parties from misbehaving. This property is called *accountability*, and it is a stronger form of verifiability as demonstrated in [30]. More specifically, accountability requires that, if some desired goal of the protocol is not met in a run (due to the misbehavior of one or more protocol participants), then the judge J individually blames those participants who misbehaved, or at least one of them. So, using $\gamma(\varphi, k)$ as the goal, accountability guarantees that, if more than k inputs of honest senders were manipulated, then (one or more of) the misbehaving parties are held accountable by J. Due to space limitations, we recall the formal accountability framework in our technical report [22].

[2] A function f is *δ-bounded* if, for every $c > 0$, there exists ℓ_0 such that $f(\ell) \leq \delta + \ell^{-c}$ for all $\ell > \ell_0$.

[3] We note that the original definition in [30] also captures soundness: if the protocol runs with a benign adversary, which, in particular, would not corrupt parties, then the judge accepts all runs. This kind of soundness can be considered to be a sanity check of the protocol, including the judging procedure, and is typically easy to check. For brevity of presentation, we omit this condition here.

V. Message Tracing (DMN)

The following technique easily extends a basic DMN. Since each sender S_i knows the trace of its own input through the mix net, S_i can look up the mix net output (which also includes the mix servers' intermediate results) to verify whether this trace was broken. Message tracing was employed in [28, 49] and formally analyzed in [28].

A. Description

We describe how to extend a basic DMN (Section II-D) for message tracing.

Channels. We additionally assume that there is an anonymous channel from each sender S_i to the bulletin board B. The sender can use this channel for submitting blaming evidence in case its input was manipulated (see below) without sacrificing its message privacy.

Submission phase. In addition to generating its encrypted input c_i, each sender S_i stores its message m_i, all random coins $r_i^1, \ldots, r_{n_\mathsf{MS}}^i$ used to iteratively encrypt m_i, and all (intermediate) ciphertexts $\mathsf{c}_i^0, \ldots, \mathsf{c}_i^{n_\mathsf{MS}-1}$.

Auditing phase. Each sender S_i can read the mix servers' outputs $C_1, \ldots, C_{n_\mathsf{MS}}$ from the bulletin board B, and perform the following check. S_i first verifies whether c_i^1 is in the signed output C_1 of the first mix server M_1. If this is not the case, S_i retrieves the locally stored random coins r_i^1 that it used to encrypt the (missing) ciphertext c_i^1 under pk_1 to obtain the ciphertext c_i^0. Then, S_i sends $(\mathsf{M}_1, \mathsf{c}_i^0, \mathsf{c}_i^1, r_i^1)$ to the bulletin board B via its anonymous channel. Due to the correctness of the underlying public-key encryption scheme $\mathcal{E}$, this tuple serves as a public evidence that M_1 misbehaved, hence holding M_1 accountable. Otherwise, if c_i^1 is in the signed output C_1 of M_1, the sender checks whether c_i^2 is in C_2, and so on. If there is a mix server M_k for which c_i^k is not in C_k, S_i publishes the respective evidence, analogously to the case of a misbehaving M_1.

B. Properties

We summarize the main properties of the message tracing technique. Accountability was formally proven in [28], Theorem 3. We also elaborate on two intrinsic issues of the message tracing technique (one of which had not been identified prior to our work), and show how to effectively solve them.

Verifiability. Observe that, to ensure verifiability, the tracing technique requires that the probability of the event that two distinct senders choose the same input message is negligible, or at least sufficiently small. Otherwise, the last mix server could undetectably replace all but identical messages by different ones (*clash attack* [31]). There are different mechanism to ensure this property, for example, by combining tracing with verification codes (Section VI) as in [28], or by adding a further encryption layer, as we recommend further below.

Under the previous assumption, in a DMN with message tracing, the probability that manipulating more than k honest inputs remains undetected is bounded by $(1-p_\mathsf{verify})^{k+1}$, where

p_{verify} denotes the probability that an honest sender performs the verification procedure described above.

The intuition behind this formula is the following one. If an adversary manipulates more than k honest senders' inputs, then this remains undetected only if none of the affected senders does the auditing. Assuming that senders verify pairwise independently, the probability that the adversary succeeds is bounded by $(1 - p_{\mathsf{verify}})^{k+1}$.[4]

We note that we do not have to assume (or be able to verify) that the public/secret key pairs were generated correctly. In fact, for each mix server M_k and each sender S_i, the individual relation of S_i's input and output message in the k-the mixing step (see above) can also be verified even if M_k published a malformed public key pk_k.

In order to guarantee that p_{verify} is sufficiently large in practice, an *automated verification* procedure was proposed in [28]. This mechanism is carried automatically by a sender's device as soon as the sender looks up the final result. In order to evaluate its effectiveness, two mock elections were carried out: in both cases, the automated verification rates reported by [28] were $\geq 55\%$. As a result, the verifiability property of the tracing technique ensures that manipulating, for example, 5 votes in one of these mock elections would have been detected with probability $\geq 95\%$.

Accountability. If a sender detects that the trace of its message through the mix net is broken, it reveals sufficient information to publicly identify a misbehaving mix server. Hence, message tracing also provides individual accountability.

Anonymous channel issue. The anonymous channels between the senders and the bulletin board are required for the following reason. Assume, for example, that the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ drops an arbitrary message m. Then, if the affected sender S_i performs the auditing described above, S_i publishes $(\mathsf{M}_{n_{\mathsf{MS}}}, \mathsf{c}_i^{n_{\mathsf{MS}}-1}, m_i, r_i^{n_{\mathsf{MS}}})$, where m_i is S_i's plain message. If the publication was not anonymous, then everyone could link S_i's identity to its message m_i.

Fairness issue. Let us consider an adversary who controls some dishonest senders and the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$. Furthermore, assume, for example, that we run an election with candidates A, B, C, and that a candidate wins if he gets more votes than any other candidate. Now, the adversary wants B or C to win but, before submission closes, does not know which one has better chances. However, since the adversary controls the last mix server, he gets to know the final outcome after he has received $C_{n_{\mathsf{MS}}-1}$. Therefore, he can change the dishonest senders' choices in $C_{n_{\mathsf{MS}}}$ "on the fly" such that one of his favourite choices wins. For example, the adversary could adaptively swap all dishonest choices from B to C such that C wins against A. Therefore, the election is not fair because B and C have an advantage. At the same time, this manipulation is not detected because the honest senders' traces remain intact.[5]

Recommendation. In order to solve the two issues mentioned above, we recommend to extend the message tracing technique as follows.

After the "main" mix net, we add a further mix net (with the same mix servers) or a single layer of encryption for which the secret key is $(n_{\mathsf{MS}}, n_{\mathsf{MS}})$-shared among the mix servers. Once the main mixing phase is over, the mix servers run the second mix net or distributed decryption, respectively. After that, during the auditing phase, each sender can verify as before whether the trace of its message through the main mix net is broken. If it is broken, then S_i *non-anonymously* sends its blaming evidence to the bulletin board B, and the whole protocol aborts. Otherwise, if no sender publishes a valid complaint during the auditing phase, then the second mix net/shared encryption layer is verified explicitly by asking each mix server to publicly reveal its secret key of the second mix net/secret key share for the final encryption layer. (Observe that verification is still possible even if the scheduled auditing phase is over.)

Extending the DMN with message tracing in this way makes the assumption of anonymous channels dispensable, and resolves the fairness issue described above (assuming one honest mix servers).

VI. Verification Codes (DMN)

The following technique is particularly simple and intuitive to extend a basic DMN. It was originally proposed in [41], and later used in [28] where it was also formally analyzed.

A. Description

We describe how to extend a basic DMN (Section II-D) for verification codes.

Submission phase. In addition to its actual message m_i, each sender S_i chooses an individual verification code n_i (of a given size) uniformly at random. Then, S_i encrypts the message-code pair (m_i, n_i) according to the underlying basic DMN.

Auditing phase. Each sender S_i can verify whether its verification code n_i appears next to its message m_i in the final outcome. If this is not the case, the sender S_i files a complaint.

B. Properties

We summarize the main properties of the verification code technique. Security was formally proven in [28], Theorem 1. We also elaborate on the feasibility of employing verification codes for RMNs.

Verifiability. In a DMN with verification codes, the probability that manipulating more than k honest inputs remains undetected is bounded by $(1 - p_{\mathsf{verify}})^{k+1}$, where p_{verify} is the probability that an honest sender performs the verification

[4] We note that in [28], a more refined verifiability tolerance was formally proven: the resulting formula also reflects that a malicious bulletin board could undetectably stuff input ciphertexts for abstaining senders with a certain probability if, for example, no PKI among the senders is assumed. Since this issue is orthogonal to the actual verifiability technique employed, we abstract away from it in this paper and present a slightly simplified formula here.

[5] This is not a contradiction to the verifiability result because the goal $\gamma(k, \varphi)$ that we used to instantiate the general verifiability definition (Definition 1) is only concerned with protecting *honest* senders' choices (as long as no dishonest choices are stuffed).

procedure described above. For this result to hold true, we need to assume that the probability of clashes [31] between codes is negligible. Then, with a similar reasoning as for the message tracing technique (Section VI-B), the verifiability result follows.

Accountability. In contrast to all other verifiability techniques in this paper, the verification code technique does not guarantee accountability. To see this, observe that in case a sender complains, the judge cannot be sure whether (i) the sender (is dishonest and) falsely claims that its message-verification code pair does not appear in the final result, or (ii) the sender (is honest and) legitimately complains.

Human verifiability. A unique feature of the verification code technique is that, for applications like secure e-voting where a human being inputs the message m_i, we can also let the human sender choose (part of) the verification code. Then, the protocol can be verified even if all computers are manipulated (assuming that the senders can look up the correct final outcome). This property is called *human verifiability*. Yet, one should keep in mind that human-generated nonces have low entropy [5, 6]. Hence, if two human senders S_i and S_j choose the same message $m_i = m_j$ and the same verification code $n_i = n_j$, then a malicious mix server can undetectably manipulate one of these two clashing messages [31].

Fairness issue. The verification code technique suffers from the same fairness issue as the message tracing technique (Section V). Therefore, when using the verification code technique, we recommend to extend the mix net as described in Section V-B.

Verification codes for RMN. Obviously, it would be possible to also apply the verification code technique for extending a basic RMN. However, we note that the resulting mix net does not provide privacy without any further means. To see this, assume that the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ is dishonest. Instead of re-encrypting and shuffling its actual input $C_{n_{\mathsf{MS}}-1}$, $\mathsf{M}_{n_{\mathsf{MS}}}$ simply re-encrypts and shuffles the input C_0 to the mix net. By this, the adversary bypasses all previous mix servers, hence, breaks privacy. This privacy attack is not detectable because none of the message/verification code pairs was manipulated. Therefore, verification codes should only be employed for RMNs in conjunction with one of the other verifiability techniques presented in this paper.

VII. Randomized Partial Checking (DMN)

The basic idea of randomized partial checking (RPC), originally proposed in [24], is as follows: each mix server M_k has to open some of the links between its input messages C_{k-1} and its output messages C_k. The set of links to be opened is chosen uniformly at random by a set of auditors. In a DMN with RPC, opening a link refers to proving that the input message decrypts to the output message, and in a RMN with RPC, this refers to proving that the output message is a re-encryption of the input message. Since the details of RPC differ for DMN and RMN, we handle them separately, starting with DMN here, and RMN in Section VIII.

A. Description

We describe how to extend a basic DMN (Section II-D) for RPC.

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

Cryptographic primitives. We additionally assume that the IND-CCA2-secure public-key encryption scheme $\mathcal{E}$ also allows for NIZKPs of correct decryption. Furthermore, we use a computationally hiding and computationally binding commitment scheme $(\mathcal{M}, \mathcal{C}, \mathcal{R}, \mathsf{Comm})$.

Protocol overview. Typically, pairs of mix servers are audited. For the sake of presentation, we will therefore assume that each mix server performs two mixing steps.

Setup phase. Each auditor AD generates its verification/signing key pair and publishes the verification key. Each mix server M_k generates two public/private key pairs $(\mathsf{pk}_{2k-1}, \mathsf{sk}_{2k-1}), (\mathsf{pk}_{2k}, \mathsf{sk}_{2k})$, and publishes the public keys.

Mixing phase. Each mix server M_k with input C_{2k-2} performs the following steps:

1) Decrypt C_{2k-2} using sk_{2k-1}, shuffle the resulting messages using σ_{2k-1}, and send the outcome C_{2k-1} to B.
2) Decrypt C_{2k-1} using sk_{2k}, shuffle the resulting messages using σ_{2k}, and send the outcome C_{2k} to B.
3) Commit to the values of $\sigma_{2k-1}(i)$ and commit to the values of $\sigma_{2k}^{-1}(i)$ for all $i \leq |C_{2k-1}|$. Send the commitments to B.

Auditing phase. After the final outcome was published,[6] the challenges for the mix servers are generated as follows:

1) *Generating randomness:* Each auditor AD_j chooses a bit string uniformly at random, commits to it, and sents the commitment to B. Once all auditors have published their commitments, each AD_j opens its commitment and the resulting bit strings are then combined into one (e.g, by XOR).

2) *Generating verification sets:* Using the random string produced by the auditors, for all M_k and for all $i \leq |C_{2k-1}|$, it is randomly decided, independently of other elements, whether i is added to the initially empty set I_k.

Then, each mix server M_k with input I_k performs the following steps:

1) *Opening/proving left links:* For all $i \in I_k$, open the i-commitment (on the value $\sigma_{2k-1}(i)$) from the first sequence of commitments on B. For all $i \in I_k$, create a NIZKP for proving that $C_{2k-1}[i]$ is obtained by decrypting $C_{2k-2}[\sigma_{2k-1}(i)]$ (using sk_{2k-1}), and send the proof to B.

2) *Opening/proving right links:* For all $i \notin I_k$, open the i-commitment (on the value $\sigma_{2k}^{-1}(i)$) from the second sequence of commitments on B. For all $i \notin I_k$, create a NIZKP for proving that $C_{2k}[\sigma_{2k}^{-1}(i)]$ is obtained by decrypting $C_{2k-1}[i]$ (using sk_{2k}), and send the proof to B.

[6] We note that there are two variants of RPC decryption mix nets, depending on when auditing takes places: either during the mixing phase or afterwards. Since, the verifiability/accountability level is the same for both variants [32], we only describe the latter variant for the sake of simplicity.

Now, everyone can verify the correctness of M_k's output by checking whether all commitments were opened correctly, the opened indices do not contain duplicates, and the decryption proofs are valid. If one of these checks fails, M_k is blamed individually.

B. Properties

We summarize the main properties of the randomized partial checking technique. Security was formally proven in [32].

Verifiability. The auditing described above guarantees that for a message from C_{2k-1} either the direct connection to some message from C_{2k-2} or to some message from C_{2k} is revealed (but never both, which would break privacy). Nevertheless, there are different kinds of manipulation that may not always be detectable [26, 32]. For example, if a malicious mix server replaces a ciphertext, either in the first or the second mixing, by a different one, then this misbehaviour remains undetected with probability $\frac{1}{2}$.

However, as demonstrated by [26, 32], there are more subtle attacks. If the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ is malicious, it can manipulate its outcome as follows. Let i and j be two different positions in $C_{2n_{\mathsf{MS}}-1}$ which encrypt the same message. First, place the honest decryption at $C_{\sigma_{2n_{\mathsf{MS}}}(i)}$ and any other message at $C_{\sigma_{2n_{\mathsf{MS}}}(j)}$, and then commit to $\sigma_{2n_{\mathsf{MS}}-1}(i)$ at both positions i and j. This manipulation can only be detected if both i *and* j belong to the set of indices I_k for which M_k has to open the right links, and hence, remains undetected with probability $\frac{3}{4}$.

Küsters et al. [32] formally proved that this attack is "optimal": if an arbitrary adversary (controlling one or more mix servers) manipulates more than k honest senders' inputs, then this remains undetected with probability at most $\left(\frac{3}{4}\right)^{k+1}$.

Accountability. The auditing procedure described above always ensures that a misbehaving mix server can be identified in case some manipulation is detected. Hence, the DMN with RPC provides individual accountability.

VIII. Randomized Partial Checking (RMN)

In this section, we describe how RPC works for RMN.

A. Description

We describe how to extend a basic RMN (Section II-D) for RPC.

Protocol participants. As for the DMN with RPC, we assume that there is a number of *auditors* $\mathsf{AD}_1, \ldots, \mathsf{AD}_{n_{\mathsf{AD}}}$.

Cryptographic primitives. In addition to the standard requirements, we use the following cryptographic primitives (the NIZKPs relate to the underlying distributed public-key encryption scheme $\mathcal{E}$):

- A computationally hiding and computationally binding commitment scheme $(\mathcal{M}, \mathcal{C}, \mathcal{R}, \mathsf{Comm})$.
- A NIZKP of knowledge of the private key share (for a given public key share).
- A NIZKP of correct re-encryption (such a proof would typically simply reveal the random coins used for re-encryption).
- A NIZKP of correct (shared) decryption.

Setup phase. In addition to the steps taken in the basic RMN, each mix server M_k generates and publishes a NIZKP of knowledge of its private key share sk_k for its public key share pk_k.

Mixing phase. The steps taken by a mix server M_k are analogous to the ones of a mix server in a DMN with RPC, except for that M_k does not decrypt but re-encrypts its inputs.

Auditing phase. The auditing phase is analogous to the one of the DMN with RPC. The only difference is that, for the RMN with RPC, each mix server has to provide a NIZKP of correct re-encryption instead of a NIZKP of correct decryption.

Decryption phase (optional). In addition to the steps taken in the basic RMN, each mix server M_k generates and publishes a NIZKP of correct decryption.

B. Properties

Security was formally analyzed in [29]. As the security results are analogous to the ones for the DMN with RPC, we refer to Section VII-B for details.

IX. Trip Wires (DMN)

The concept of the trip wire technique was, in a specific variant, originally employed in the mix net by Khazaei et al. [25] (Section X) as a subroutine. Subsequently, Boyen et al. [7] generalized the original trip wire technique so that it can be used as an independent verifiability technique.

The basic idea of the trip wire technique is to hide the senders' inputs by a set of indistinguishable *dummy inputs* which serve as trip wires. The trip wires are injected by a set of auditors one of which needs to be trusted temporarily (similar to the RPC technique, Section VII). Now, the mix net is run with this extended set of inputs. Once mixing has finished, the auditors publicly reveal the trip wires' traces through the mix net. If a mix server M_k manipulated one of the dummy traces, then M_k can be identified and be held accountable.

In order to guarantee that the mix servers cannot distinguish between real and dummy inputs, two mechanisms are integrated. First, the complete input is "pre-mixed" by the auditors using a plain DMN (*explicit mix net*). Second, an additional layer of encryption (with shared secret key among the auditors) is added directly to the plain input messages (*repetition encryption layer*). Hence, the input to the main DMN consists of a list of secretly shuffled real and dummy ciphertexts, and its output is still encrypted under the innermost layer of encryption. Now, once the mix server have published the (still encrypted) outcome of the main DMN, each auditor is supposed to reveal its secret key of the first DMN so that its correctness can be verified perfectly. Furthermore, all dummy traces are revealed. If one of these checks is negative, the misbehaving party can be singled out and the whole process aborts. Otherwise, each auditor reveals its share of the innermost secret key so that the DMN outcome can publicly be decrypted.

TABLE I: Techniques for Verifiable Mix Nets: Primitives and Verifiability

Technique	Cryptographic primitives	Trust assumptions	Verifiability tolerance	Accountability
Basic DMN (Sec. II-D)	• IND-CCA2-secure PKE			
Tracing (Sec. V)	+ none		$(1-p_{\mathsf{verify}})^{k+1}$	indiv.
Verification Codes (Sec. VI)	+ none		$(1-p_{\mathsf{verify}})^{k+1}$	none
RPC (Sec. VII)	+ NIZKPs of correct decryption + Commitment scheme	$\bigvee_j \mathsf{hon}(\mathsf{AD}_j)$	$\left(\frac{3}{4}\right)^{k+1}$	indiv.
Trip Wires (Sec. IX)	+ IND-CCA2-secure distributed PKE	$\bigvee_j \mathsf{hon}(\mathsf{AD}_j)$	$\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1} / \binom{n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}}{k+1}$	indiv.
Replication (Sec. X)	+ IND-CCA2-secure distributed PKE + NIZKP of correct decryption	$\bigvee_j \mathsf{hon}(\mathsf{M}_j)$	$\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1} / \binom{n_{\mathsf{repl}}(n_{\mathsf{S}}^{\mathsf{hon}}+1)}{n_{\mathsf{repl}}\cdot(k+1)}$	indiv.
Basic RMN (Sec. II-D)	• IND-CPA-secure threshold PKE (with re-encryption) • NIZKP of private key share knowledge • NIZKP of plaintext knowledge • NIZKP of correct (shared) decryption			
RPC (Sec. VIII)	+ NIZKP of correct re-encryption + Commitment scheme	$\bigvee_j \mathsf{hon}(\mathsf{AD}_j)$	$\left(\frac{3}{4}\right)^{k+1}$	indiv.
Correct Shuffle (Sec. XI)	+ NIZKP of correct shuffle	N/A	negl.	indiv.

Cryptographic primitives: "+" denotes additional cryptographic primitives to basic DMN/RMN. *Trust assumptions:* Parties required to be honest for verifiability/accountability (modeled by φ in the goal $\gamma(k,\varphi)$). *Verifiability tolerance:* Upper bound δ for the probability that manipulating more than k honest inputs remains undetected (under given trust assumptions). See Section IV for details on verifiability and accountability definitions.

A. Description

We describe how to extend a basic DMN (Section II-D) for trip wires.

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

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

Setup phase. The following additional steps are executed. Each auditor AD_j generates its verification/signing key pair, a public/private key pair $(\mathsf{pk}_j^{\mathsf{expl}}, \mathsf{sk}_j^{\mathsf{expl}})$ for the explicit mix net, and a public/private key share pair $(\mathsf{pk}_j^{\mathsf{rep}}, \mathsf{sk}_j^{\mathsf{rep}})$ (w.r.t. $\mathcal{E}_{\mathsf{d}}$) for the repetition encryption layer. AD_j publishes the respective public keys on B. With $\mathsf{PublicKeyGen}$ everyone can then compute the joint public key $\mathsf{pk}^{\mathsf{rep}}$ for the repetition layer.

Submission phase (senders). Each sender S_i first encrypts its message m_i under the auditors' joint public key $\mathsf{pk}^{\mathsf{rep}}$, then under the mix servers' public keys $\mathsf{pk}_1, \dots, \mathsf{pk}_{n_{\mathsf{MS}}}$ of the main decryption mix net, and eventually under the auditors' public keys $\mathsf{pk}_1^{\mathsf{expl}}, \dots, \mathsf{pk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}$ of the explicit decryption mix net in reverse order. The resulting ciphertext c_i 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). Furthermore, AD_j stores the random coins that it used to generate its trip wire ciphertexts.

Mixing phase. The input to the mixing phase consists of 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 $\mathsf{sk}_1^{\mathsf{expl}}, \dots, \mathsf{sk}_{n_{\mathsf{AD}}}^{\mathsf{expl}}$ to run the basic DMN.
2) *Main mixing:* The mix servers use their secret decryption keys $\mathsf{sk}_1, \dots, \mathsf{sk}_{n_{\mathsf{MS}}}$ to run the basic DMN.

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 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, each output ciphertext of the main mix net can publicly be decrypted.

B. Properties

We summarize the main properties of the trip wire technique. Verifiability and accountability were formally analyzed in [7].

Verifiability. The verifiability theorem for the trip wire technique states that, assuming at least one honest auditor, the probability of manipulating more than k honest inputs without being detected is bounded by $\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{tw}}) =$

TABLE II: Techniques for Verifiable Mix Nets: Computational Complexity

Mix Net Technique	Cost Setup (per authority)	Cost Submission (per sender)	Cost Mixing (per authority)	Cost Audit (per mix server)	Cost Audit (public)
Basic DMN (Sec. II-D)	$\mathcal{C}_{\mathsf{key}}(n_{\mathsf{MS}})$	$\mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}})$	$n_{\mathsf{S}} \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}})$	N/A	N/A
Tracing (Sec. V)	No Overhead	No Overhead	No Overhead	N/A	Per claimed cheat: $\mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}})$
Verification Codes (Sec. VI)	No Overhead	No Overhead	No Overhead	N/A	N/A
RPC (Sec. VII)	$2 \cdot \mathcal{C}_{\mathsf{key}}(2n_{\mathsf{MS}})$	$\mathcal{C}_{\mathsf{enc}}(2n_{\mathsf{MS}})$	$2n_{\mathsf{S}} \cdot \mathcal{C}_{\mathsf{dec}}(2n_{\mathsf{MS}})$	$n_{\mathsf{S}} \cdot \mathcal{C}^{\mathsf{proof}}_{\mathsf{dec}}(2n_{\mathsf{MS}})$	$n_{\mathsf{MS}} \cdot n_{\mathsf{S}} \cdot \mathcal{C}^{\mathsf{verif}}_{\mathsf{dec}}(2n_{\mathsf{MS}})$
Trip Wires (Sec. IX)	Mix server: $\mathcal{C}_{\mathsf{key}}(n_{\mathsf{MS}}+1)$ Auditor: $2 \cdot \mathcal{C}_{\mathsf{key}}(n_{\mathsf{MS}}+n_{\mathsf{AD}}+1) + n_{\mathsf{tw}} \cdot \mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}}+n_{\mathsf{AD}}+1)$	$\mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}}+n_{\mathsf{AD}}+1)$	Mix server: $(n_{\mathsf{S}}+n_{\mathsf{tw}} \cdot n_{\mathsf{AD}}) \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}}+1)$ Auditor: $(n_{\mathsf{S}}+n_{\mathsf{tw}} \cdot n_{\mathsf{AD}}) \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}}+n_{\mathsf{AD}}+1)$	N/A	Pre-mix net: $n_{\mathsf{AD}} \cdot (n_{\mathsf{S}}+n_{\mathsf{tw}} \cdot n_{\mathsf{AD}}) \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}}+n_{\mathsf{AD}}+1)$ Main mix net: $n_{\mathsf{tw}} \cdot n_{\mathsf{AD}} \cdot n_{\mathsf{MS}} \cdot \mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}}+2)$
Replication (Sec. X)	$5 \cdot \mathcal{C}_{\mathsf{key}}(2n_{\mathsf{MS}}+3) + \mathcal{C}_{\mathsf{enc}}(2n_{\mathsf{MS}}+3)$	$n_{\mathsf{repl}} \cdot \mathcal{C}_{\mathsf{enc}}(2n_{\mathsf{MS}}+3)$	$n_{\mathsf{repl}} \cdot (n_{\mathsf{S}}+n_{\mathsf{MS}}) \cdot \mathcal{C}_{\mathsf{dec}}(2n_{\mathsf{MS}}+2)$	Per backwards/forward trace: $n_{\mathsf{repl}} \cdot \mathcal{C}^{\mathsf{proof}}_{\mathsf{dec}}(n_{\mathsf{MS}}+2)$	Pre-mix net: $n_{\mathsf{MS}} \cdot n_{\mathsf{repl}} \cdot (n_{\mathsf{S}}+n_{\mathsf{MS}}) \cdot \mathcal{C}_{\mathsf{dec}}(2n_{\mathsf{MS}}+2)$ Main mix net: $n_{\mathsf{tw}} \cdot n_{\mathsf{MS}} \cdot n_{\mathsf{MS}} \cdot \mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}}+2)$ Per backwards/forward trace: $n_{\mathsf{repl}} \cdot n_{\mathsf{MS}} \cdot \mathcal{C}^{\mathsf{verif}}_{\mathsf{dec}}(n_{\mathsf{MS}}+2)$
Basic RMN (Sec. II-D)	$\mathcal{P}_{\mathsf{key}}(1) + \mathcal{P}^{\mathsf{proof}}_{\mathsf{key}}(1)$	$\mathcal{P}_{\mathsf{enc}}(1) + \mathcal{P}^{\mathsf{proof}}_{\mathsf{enc}}(1)$	$n_{\mathsf{S}} \cdot \mathcal{P}_{\mathsf{reenc}}(1)$	N/A	$n_{\mathsf{MS}} \cdot \mathcal{P}^{\mathsf{verif}}_{\mathsf{key}}(1) + n_{\mathsf{S}} \cdot \mathcal{P}^{\mathsf{verif}}_{\mathsf{enc}}(1)$
RPC (Sec. VIII)	No Overhead	No Overhead	Overhead: $n_{\mathsf{S}} \cdot \mathcal{P}_{\mathsf{reenc}}(1)$	$n_{\mathsf{S}} \cdot \mathcal{P}^{\mathsf{proof}}_{\mathsf{reenc}}(1)$	Overhead: $n_{\mathsf{MS}} \cdot n_{\mathsf{S}} \cdot \mathcal{P}^{\mathsf{verif}}_{\mathsf{reenc}}(1)$
Correct Shuffle (Sec. XI)	No Overhead	No Overhead	No Overhead	$\mathcal{P}^{\mathsf{proof}}_{\mathsf{shuffle}}$	Overhead: $n_{\mathsf{MS}} \cdot \mathcal{P}^{\mathsf{verif}}_{\mathsf{shuffle}}$

Remark: See Section XII for the notation.

$\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1}/\binom{n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}}{k+1}$. The intuition behind this formula is the following one. Since the explicit mixing and the repetition encryption layer are perfectly verifiable, an adversary can only manipulate 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} trip wires by the honest auditor) are pairwise indistinguishable. Hence, the probability of not being caught cheating equals to the one of picking more than k out of $n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}$ balls such that all of them belong to the first group of $n_{\mathsf{S}}^{\mathsf{hon}}$ balls. The probability of this event is at most $\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1}/\binom{n_{\mathsf{S}}^{\mathsf{hon}}+n_{\mathsf{tw}}}{k+1}$. 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.

Accountability. The auditing procedure described above ensures that a misbehaving mix server can be identified. Hence, the DMN with trip wires provides individual accountability (assuming at least one honest auditor).

X. Message Replication (DMN)

The basic idea of the message replication technique, originally proposed by Khazaei, Moran, and Wikström [25], is to let each sender S_i replicate its input several times so that all of its replications are part of its input, too. Now, if a malicious mix server M_k tries to manipulate S_i's input, then M_k has to simultaneously manipulate S_i's replicated inputs *in the same way* as well.

In order to guarantee that a malicious mix server M_k cannot distinguish between groups of associated ciphertexts, the following mechanisms are integrated. Similarly to the trip wire technique (Section IX), the input ciphertexts are "premixed" using a basic DMN (*explicit mixing*), and an additional encryption layer is added to the plain input messages (*repetition encryption layer*). In contrast to the trip wire technique (Section IX), where a number of external auditors establish these two mechanisms, in the mix net by Khazaei et al. [25], the mix servers themselves execute them.

Furthermore, in order to resolve possible disputes between senders and mix servers, the auditing phase contains a tracing mechanism. Using this mechanism, it is possible to single out whether a malicious sender submitted mal-formed input ciphertexts or a malicious mix server manipulated a sender's message. Since the tracing mechanism reveals the links between a sender and some output messages, two more encryption layers are required to protect the senders' message privacy (*outer encryption layer* and *final encryption layer*).

Observe that in the version of the replication technique described so far, a malicious mix server could simply replace *all* (honest) messages at once which would remain undetected. In order to protect against this attack, Khazaei et al. [25] employed the following variant of the trip wire technique (Section IX): each mix server M_k injects a single dummy message to the input. Obviously, this mechanism protects against the verifiability attack described above.

A. Description

We describe how to extend a basic DMN (Section II-D) for replications.

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

Setup phase. The following additional steps are executed. Each mix server M_k (i) generates a public/private key pair $(\mathsf{pk}_k^{\mathsf{expl}}, \mathsf{sk}_k^{\mathsf{expl}})$ for the explicit mix net and publishes the public key $\mathsf{pk}_k^{\mathsf{expl}}$, and (ii) generates three public/private key share pairs $(\mathsf{pk}_k^{\mathsf{out}}, \mathsf{sk}_k^{\mathsf{out}})$, $(\mathsf{pk}_k^{\mathsf{rep}}, \mathsf{sk}_k^{\mathsf{rep}})$, $(\mathsf{pk}_k^{\mathsf{fin}}, \mathsf{sk}_k^{\mathsf{fin}})$ (w.r.t. $\mathcal{E}_{\mathsf{d}}$) and publishes the public key shares $\mathsf{pk}_k^{\mathsf{out}}, \mathsf{pk}_k^{\mathsf{rep}}, \mathsf{pk}_k^{\mathsf{fin}}$. With PublicKeyGen, everyone can then compute the joint public keys $\mathsf{pk}^{\mathsf{out}}, \mathsf{pk}^{\mathsf{rep}}, \mathsf{pk}^{\mathsf{fin}}$ for the outer, repetition, and final encryption layer, respectively.

Submission phase (senders). Each sender S_i first encrypts its message m_i under the mix servers' joint public key $\mathsf{pk}^{\mathsf{fin}}$. Then, S_i makes n_{repl} identical copies of this ciphertext, and encrypts each copy first under the mix servers' joint public key $\mathsf{pk}^{\mathsf{rep}}$, then under the mix servers' public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{MS}}}$ of the main decryption mix net in reverse order, and then under the mix servers' public keys $\mathsf{pk}_1^{\mathsf{expl}}, \ldots, \mathsf{pk}_{n_{\mathsf{MS}}}^{\mathsf{expl}}$ of the explicit decryption mix net in reverse order. The resulting ciphertexts are concatenated and the concatenation is encrypted under the mix servers' joint public key $\mathsf{pk}^{\mathsf{out}}$. The resulting ciphertext c_i is S_i's input to the mix net.

Submission phase (mix servers). Each mix server M_k executes one time the senders' submission steps described above with (dummy) input message $m = 0^l$ (where l is the bit size of a sender's message). Furthermore, M_k stores the random coins that it used to generate its trip wire ciphertexts.

Mixing phase. The input to the mixing phase consists of the n_{S} ciphertexts submitted by the senders and the n_{MS} ciphertexts submitted by the mix servers. Then, the overall mixing phase consists of the following consecutive parts:

1) *Decrypting outer layer:* The mix servers reveal their secret key shares $\mathsf{sk}_1^{\mathsf{out}}, \ldots, \mathsf{sk}_{n_{\mathsf{MS}}}^{\mathsf{out}}$ and all input ciphertexts are publicly decrypted.
2) *Splitting:* Each resulting message is split into n_{repl} ciphertexts.
3) *Explicit mixing:* The mix servers use their secret decryption keys $\mathsf{sk}_1^{\mathsf{expl}}, \ldots, \mathsf{sk}_{n_{\mathsf{MS}}}^{\mathsf{expl}}$ to run the basic DMN.
4) *Main mixing:* The mix servers use their secret decryption keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{MS}}}$ to run the basic DMN.

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

After that, each mix server M_k 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.

Afterwards, each mix server M_k publishes its secret key share $\mathsf{sk}_k^{\mathsf{rep}}$ on the bulletin board B. Then, each output ciphertext of the main mix net can publicly be decrypted.

After that, it is publicly verified whether the list of ciphertexts (without the revealed dummy messages) can be decomposed into groups of n_{repl} identical ciphertexts. If there is a ciphertext which does not belong to a group of (a multiple of) n_{repl} identical ciphertexts, this can be due to a misbehaving sender or due to a misbehaving mix server. For each such ciphertext c', this is publicly verified as follows:

1) *Backward tracing:* First, the last mix server $\mathsf{M}_{n_{\mathsf{MS}}}$ reveals the ciphertext that $\mathsf{M}_{n_{\mathsf{MS}}}$ decrypted to c' under its secret key. Furthermore, $\mathsf{M}_{n_{\mathsf{MS}}}$ generates and publishes a NIZKP of correct decryption π^{Dec} for proving the correctness of this relation. If this proof is not valid, then the whole process stops and $\mathsf{M}_{n_{\mathsf{MS}}}$ is held accountable. Otherwise, the second but last mix server continues, and so on. Eventually, the ciphertext c' can be traced back to one of the senders.

2) *Forward tracing:* Each input ciphertext from the identified sender is now verifiably traced forward to a ciphertext in the output of the mixing phase, analogously to the backward tracing. Again, if one the mix servers does not publish a required NIZKP of correct decryption π^{Dec}, the whole process stops and this mix server is held accountable. Eventually, all ciphertexts linked to the identified sender are removed.

Altogether, the auditing phase either identifies a misbehaving mix server or outputs a list of ciphertexts which are to be decrypted (see next step).

Final decryption phase. Each mix server M_k publishes its secret key share $\mathsf{sk}_k^{\mathsf{fin}}$ on the bulletin board B. Then, each output ciphertext of the auditing phase can publicly be decrypted.

B. Properties

We summarize the main properties of the replication technique. We state and prove its formal verifiability/accountability theorem in our technical report [22]. We note that in the original publication [25], an unproven security theorem was stated. In this work, we give the first formal proof of the original statement—in fact, our result even refines Khazaei et al.'s theorem.

Verifiability. The verifiability theorem for the replication technique states that, assuming at least one honest mix server, the probability of manipulating more than k honest inputs without being detected is bounded by $\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{repl}}) = \binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1} / \binom{n_{\mathsf{repl}}(n_{\mathsf{S}}^{\mathsf{hon}}+1)}{n_{\mathsf{repl}} \cdot (k+1)}$. The intuition behind this formula is the following one. Since the explicit mixing, as well as the outer, repetition, and final encryption layer are perfectly verifiable, an adversary can only manipulate 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 $n_{\mathsf{repl}}(n_{\mathsf{S}}^{\mathsf{hon}}+1)$ ciphertexts related to the honest input parties (n_{repl} ciphertexts for each of the $n_{\mathsf{S}}^{\mathsf{hon}}$ honest senders plus n_{repl} by the honest mix server) are pairwise indistinguishable. Hence, the probability of not being caught cheating equals to the one of picking more than k associated groups (each of size n_{repl}) out of $n_{\mathsf{repl}}(n_{\mathsf{S}}^{\mathsf{hon}}+1)$ balls such that all of them belong to the first group of $n_{\mathsf{S}}^{\mathsf{hon}}$ balls. The probability of this event is at most $\binom{n_{\mathsf{S}}^{\mathsf{hon}}}{k+1}/\binom{n_{\mathsf{repl}}(n_{\mathsf{S}}^{\mathsf{hon}}+1)}{n_{\mathsf{repl}}\cdot(k+1)}$. In particular, for all k, the verifiability tolerance $\delta_k(n_{\mathsf{S}}^{\mathsf{hon}}, n_{\mathsf{repl}})$ is bounded by $(1/n_{\mathsf{S}}^{\mathsf{hon}})^{n_{\mathsf{repl}}-1}$ (which proves Theorem 1 in [25]).

Accountability. The auditing procedure described above ensures that a misbehaving mix server can be identified. Hence, the DMN with message replication provides individual accountability (assuming at least one honest mix server).

XI. Proof of Correct Shuffle (RMN)

One of the most popular techniques to transform a basic RMN into a verifiable one is to let each mix server prove in zero-knowledge that it correctly re-encrypted and shuffled its input ciphertexts. There are numerous instantiations of this technique [1, 2, 4, 16–18, 20, 23, 35, 36, 45, 50].

A. Description

We describe how to extend a basic RMN (Section II-D) for proofs of correct shuffle.

Cryptographic primitives. In addition to the standard requirements, we use a NIZKP for the following relation (w.r.t. $\mathcal{E}$): given two ciphertext vectors C and C' of size n, there exists a permutation σ over $\{1,\ldots,n\}$ and a list of random coins $r_1,\ldots,r_n$ such that for every $i \in \{1,\ldots,n\}$, the $\sigma(i)$-th ciphertext in C' is a re-encryption of the i-th ciphertext in C using the random coins r_i. This is called a NIZKP of *correct shuffle*.

Mixing phase. In addition to the steps taken in the basic RMN, each mix server M_k generates and publishes a NIZKP of correct shuffling for its output C_k. If mix server M_k receives C_{k-1} as input without a (valid) NIZKP of correct shuffle, it immediately aborts.

Auditing phase. In addition to the checks in the basic mix net, for each mix server M_k, it is verified whether M_k published a valid NIZKP of correct shuffle. If this is not the case, M_k is held accountable and the whole process stops. Otherwise, the decryption phase starts.

B. Properties

Verifiability. A proof of correct shuffle by definition includes a sound verifier for checking claimed proofs. The one caveat is that often the proof of correct shuffle is more properly a Zero Knowledge *Argument* rather than a Zero Knowledge *Proof*; in this case, care must be taken that whatever additional conditions are introduced are satisfied. Normally, this means that for any ppt algorithm which produces a valid proof, with non-negligible probability either the shuffle was performed correctly or the trapdoor key to the CRS can be extracted.

Accountability. Since the proofs are, normally, independent for each mix server, this very naturally converts verifiability into accountability; any mix server which fails to produce a proof which verifies is held accountable. In the case of a cumulative proof (e.g., [23]), each step should be published in addition to the result. The first authority who outputs a proof which does not verifies successfully should be blamed.

Instantiations. In practice, the most common proofs of correct shuffle appear to be [45, 50] and [4] (which produces smaller proof transcripts than [45, 50]). They apply to any public-key encryption scheme which allows for re-encryption and for which a sigma protocol for correct re-encryption is known. There are also more efficient proofs of correct shuffle which have since emerged [16–18]. These new proofs are roughly three times faster than [4, 45, 50] and the cost of the verifiable mixing is close to optional, meaning little further improvement is possible. Recent work [23] has suggested using updatable proofs, where each mix server updates the proof as it mixes. This results in a verification complexity which is independent of the number mix severs. There are approaches for post-quantum proofs of correct shuffle [11, 12, 44] which are, however, not practical. This may begin to change in the near future but for now both the space and time requirements are prohibitive.

One of the issues with proofs of correct shuffles is that they rely upon a Common Reference String (CRS). If an adversary knows the trapdoor to the CRS, then it can efficiently create fake proofs which pass verification. In some cases, like electronic voting, where verifiability should hold without relying on any trust assumptions, generating the CRS is hard. For [4, 45, 50], this CRS is a collection of group elements and the trapdoor is the discrete log relationship between them. Fortunately, in this case, there are standards such as the one contained in FIPS 186.4 [19] (A.2.3) which allow the verifiable generation of the CRS without any trust assumptions. Some of the newer, and more efficent, proofs of correct shuffle [16–18] use more complicated CRSs which it is unclear how to generate without creating trust assumptions.

XII. Computational Analysis

In Section XII-A, we first introduce some notation to analyze the computational cost of the various verifiability techniques. Afterwards, in Section XII-B, we analyze the computational cost of the basic DMN and the basic RMN. For each verifiability technique studied in this paper, the cost of the resulting verifiable mix net to the various parties activate in the different stages is explained in the respective section. In Table II, we summarize and compare all of these results.

A. Notation

The computational complexity of each verifiability techniques depends on the number of senders n_{S}, mix servers n_{MS}, and trustees n_{T}. Some techniques also involve auditors whose number n_{AD} is a basic complexity parameter, too.

We describe the computational complexity of all verifiable mix nets depending on the underlying costs of the respective cryptographic primitives, i.e., the public-key encryption scheme and the ZKPs.[7] By $\mathcal{C}$, we refer to the IND-CCA2 public-key encryption scheme in a DMN, and by $\mathcal{P}$ to the IND-CPA public-key encryption scheme in an RMN. For both, $\mathcal{C}$ and $\mathcal{P}$, let the subscripts $_{\mathsf{key}}$, $_{\mathsf{enc}}$, and $_{\mathsf{dec}}$ denote the computational cost of the respective key generation, encryption, and decryption algorithm.

Decryption mix nets. For the IND-CCA2 public-key encryption scheme $\mathcal{C}$, we parameterize the above costs on the number of layers of nested ciphertexts. For example, $\mathcal{C}_{\mathsf{enc}}(3)$ denotes the cost of iteratively using the encryption algorithm of $\mathcal{C}$ three times. Depending on the concrete instantiation of $\mathcal{C}$, iteratively encrypting a message can significantly increase the overall computational complexity of the DMN. For example, using ElGamal, doubling the group size with each nesting, results in an exponential blowup, whereas using AES (in an appropriate mode of operation) with a random key encrypted under an IND-CCA2 secure public-key encryption scheme has a linear increase. Observe that RMNs do not suffer from this issue!

Furthermore, let $\mathcal{C}_{\mathsf{dec}}^{\mathsf{proof}}$ and $\mathcal{C}_{\mathsf{dec}}^{\mathsf{verif}}$ denote the cost of generating and verifying the zero knowledge proof of correct decryption for the IND-CCA2 encryption scheme, respectively. They are also parameterized on the level of nesting.

Re-encryption mix nets. For the IND-CPA encryption scheme $\mathcal{P}$, let $\mathcal{P}_{\mathsf{reenc}}$ denote the cost of the re-encryption algorithm. Let $\mathcal{P}_{\mathsf{key}}^{\mathsf{proof}}$, $\mathcal{P}_{\mathsf{key}}^{\mathsf{verif}}$, $\mathcal{P}_{\mathsf{enc}}^{\mathsf{proof}}$, $\mathcal{P}_{\mathsf{enc}}^{\mathsf{verif}}$, $\mathcal{P}_{\mathsf{reenc}}^{\mathsf{proof}}$, $\mathcal{P}_{\mathsf{reenc}}^{\mathsf{verif}}$ denote the cost of generating and verifying the ZKPs of correct key generation, encryption, and re-encryption for $\mathcal{P}$, respectively.

B. Computational Complexity of the Basic Mix Nets

We now analyze the computational complexity of the basic DMN and the basic RMN. This allows us to easily describe the overhead introduced by each of the verifiability techniques.

Basic DMN. In the setup phase, each mix server generates its public key at a cost bounded by $\mathcal{C}_{\mathsf{key}}(n_{\mathsf{MS}})$. In the submission phase, each sender iteratively encrypts its input under the public key of each mix server at a total cost bounded by $\mathcal{C}_{\mathsf{enc}}(n_{\mathsf{MS}})$. In the mixing phase, each mix server M_k decrypts and permutes its input at a cost bounded by $n_{\mathsf{S}} \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}} - i + 1) \leq n_{\mathsf{S}} \cdot \mathcal{C}_{\mathsf{dec}}(n_{\mathsf{MS}})$. The audit phase has no cryptographic work and therefore negligible cost.

Basic RMN. In the setup phase, each trustee generates its share of the public key and proves that it knows the corresponding secret key share at a cost bounded by $\mathcal{P}_{\mathsf{key}}(1) + \mathcal{P}_{\mathsf{key}}^{\mathsf{proof}}(1)$. In the submission phase, each sender encrypts its input under the mix servers' joint public key and proves knowledge of the plaintext at cost bounded by $\mathcal{P}_{\mathsf{enc}}(1) + \mathcal{P}_{\mathsf{enc}}^{\mathsf{proof}}(1)$. In the mixing phase, each mix server re-encrypts and permutes its input at a cost bounded by $n_{\mathsf{S}} \cdot \mathcal{P}_{\mathsf{reenc}}(1)$. In the audit phase, the ZKPs of secret key share knowledge and the ZKPs of plaintext knowledge are verified at the cost of $n_{\mathsf{MS}} \cdot \mathcal{P}_{\mathsf{key}}^{\mathsf{verif}}(1) + n_{\mathsf{S}} \cdot \mathcal{P}_{\mathsf{enc}}^{\mathsf{verif}}(1)$.

[7]We have elected to exclude from this analysis the commitment and signature schemes since the computational cost they bring is insignificant and would serve only to obscure the write up.

XIII. Discussion

Based on the uniform and transparent treatment of the different verifiability techniques in the previous sections, we can now precisely elaborate on the questions that motivated this systemization of knowledge.

Q: Which verifiability levels do the different verifiability techniques provide and how do these levels relate?

A: At a high level, we identified three different classes of verifiability levels which can be ordered as follows, starting with the strongest one:

1) Manipulating at least one honest input remains undetected with at most negligible probability (proof of correct shuffle).
2) Manipulating at least one honest input remains undetected with probability at most $(1/n_{\mathsf{S}}^{\mathsf{hon}})^{n_{\mathsf{repl}}-1}$ (replication technique).
3) Manipulating more than k honest inputs remains undetected with probability at most f^{k+1} where f is some linear function. We have $f = (1 - p_{\mathsf{verify}})$ for the tracing and verification code technique, $f = 3/4$ for RPC, and $f = n_{\mathsf{S}}^{\mathsf{hon}}/(n_{\mathsf{S}}^{\mathsf{hon}} + n_{\mathsf{tw}})$ for the trip wire technique.

Within the latter class, we have the following order. If $3n_{\mathsf{tw}} > n_{\mathsf{S}}$, then the trip wire technique offers a better verifiability level than RPC (under the same trust assumptions, see next question). For a given protocol run, the tracing and verification code technique provide a better verifiability level than RPC or the trip wire technique if and only if $p_{\mathsf{verify}} > 1/4$ or if $p_{\mathsf{verify}} > n_{\mathsf{tw}}/(n_{\mathsf{S}}^{\mathsf{hon}} + n_{\mathsf{tw}})$, respectively. Note, however, that p_{verify} crucially depends on the concrete application as well as the specific protocol run.

Q: Which trust assumptions do the different verifiability techniques make?

A: We identified four different classes of trust assumptions:

1) No trust is required (tracing, verification codes).
2) At least one auditor needs to be trusted (RPC, trip wires). This assumption is qualitatively weaker for RPC than for trip wires as it neither affects efficiency nor robustness.
3) At least one mix server needs to be trusted (replication).[8]
4) A general statement is not possible (proof of correct shuffle). Whether trust is required, typically depends on how the CRS can be constructed (Section XI-B).

[8]Note that it is possible to define replication such that a set of external auditors injects replicated messages. In that variant, one auditor needs to be trusted. In the original paper the two roles were combined and we preserved that in our description of replication.

Q: Are there any limitations in terms of what can be verified?

A: Using the tracing, verification code, trip wire or replication code technique, an adversary can manipulate dishonest senders' choices "on the fly" during the mixing phase without being detected. The reason is that all of these techniques merely protect the integrity of the honest senders' choices (and prevent dishonest input stuffing). However, based on internal or external information (e.g., exit polls), an adversary may adaptively tamper with the outcome of the mix net to take advantage over the honest senders (e.g., move dishonest voters' choices from one to another candidate). We refer to Section V-B for a concrete example.

Q: Which cryptographic primitives do the different verifiability techniques require?

A: The most basic cryptographic primitives are required by the tracing and the verification code technique (no additional primitives), followed by the trip wire technique (black-box distributed decryption). On the next level, we have the RPC technique and the replication technique, both of which additionally require a (black-box) NIZKP of correct decryption/re-encryption. Finally, the proofs of correct shuffle require the most sophisticated cryptographic primitives in this line.

Q: Which verifiability techniques can be instantiated using practical post-quantum cryptographic primitives only?

A: There are efficient instantiations and highly practical implementations of lattice-based IND-CCA2-secure (distributed) public-key encryption schemes and EUF-CMA-secure signature schemes (see, e.g., [37]). However, no practical (NI)ZKP of correct decryption for lattice-based encryption schemes has been published so far (whose security can be reduced to lattice assumptions, too). From the previous answer, it follows that if the objective is to instantiate a verifiable mix net with existing practical lattice-based cryptographic primitives only, then one has to extend a basic DMN with the tracing, verification code, or trip wire technique (or, if suitable, a combination thereof).[9]

Q: What concrete instantiations of the verifiability techniques are used?

A: All of the techniques reviewed, with the exception of replication, have been implemented. The most common concrete instantiation used in national elections is ElGamal encryption with a proof of correct shuffle; this has been used in Norway [21], Estonia [38], Switzerland [42, 43] and Australia [10]. RPC with ElGamal encryption has also been used in Australia [14].

Q: Which computational complexities do the different verifiability techniques have?

A: We refer to Fig. II for the technical details and summarize the main insights in what follows. The most lightweight verifiability techniques are tracing and verification codes, followed by RPC. Importantly, the verifiability level of these techniques is independent of their complexity. In contrast to that, the trip wire and the replication technique have the disadvantage that improving their verifiability levels (either by increasing n_{tw} or n_{repl}) increases the complexity in all phases and for all participants (senders, auditors, mix servers). Regarding the proofs of correct shuffle, a general statement is not possible as the respective complexity is typically very specific (recall Section XI-B for details).

Q: Which message privacy guarantees do the different verifiability techniques provide?

A: The only verifiability technique that does not provide privacy even in the presence of honest-but-curious adversaries is RPC because some information about the permutations used is always revealed which weakens privacy to a certain degree (see [32] for details).

We have already pointed out that a RMN with verification codes does not provide privacy without any further means (Section VI-B). For a DMN with tracing and for a DMN with verification codes, it was formally proven in [28] that these two mix nets with $n_{\mathsf{S}}^{\mathsf{hon}}$ honest senders provide the same level of privacy as an ideal mix net with $n_{\mathsf{S}}^{\mathsf{hon}} - k$ honest senders if an adversary manipulates at most k honest inputs. We conjecture that this also holds true for the trip wire and the replication technique. It is also worth noting that many of the proof of correct shuffle mix nets do not prove the privacy of the mix net; rather, they prove that the proof of correct shuffle is zero knowledge. The gap here is that the mix net includes not only the proof of correct shuffle but also the decryption of the ciphertexts, which also needs to be simulated. This is easily fixed by requiring the input to be coupled with proofs of plaintext knowledge, as we do in our definition of a basic RMN (Section II-D).

XIV. Conclusion

We have extracted all existing verifiability techniques for mix nets from the literature and presented them in a uniform framework. We have systematically studied and compared all of these techniques in terms of their precise verifiability levels, underlying trust assumptions, required cryptographic primitives, and computational overheads. Furthermore, we have discovered additional issues that had not been published prior to our work. We also provide the first formal verifiability proof for the message replication technique.

Altogether, our systemization of knowledge demonstrates that there does not exist a "one-size-fits-all" verifiable mix net. Instead, one always has to find a balance between different properties that fits well to a given application.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback. We also thank Peter B. Rønne and David Mestel for helpful remarks. Both authors acknowledge support from the Luxembourg National Research Fund (FNR) and the Research Council of Norway for the joint INTER project SURCVS (Number 11747298).

[9]For example, Boyen et al. [7] implemented a DMN (with trip wires) using a robust IND-CCA2-secure hybrid encryption scheme, consisting of a lattice-based CCA2-secure KEM, combined with an AES256-based DEM/MAC. They report that a single shuffle of 1 million ciphertexts takes less than 2.5 min (on commodity consumer hardware, targeting the 240-bit security level).

REFERENCES

[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] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous Messaging via Secure Multiparty Computation. *USENIX Security 2017*, pages 1217–1234, 2017.

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

[5] Joseph Bonneau. *Guessing Human-Chosen Secrets*. PhD thesis, University of Cambridge, 2012.

[6] Joseph Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *2012 IEEE S&P*, pages 538–552. IEEE, 2012.

[7] Xavier Boyen, Thomas Haines, and Johannes Müller. 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] Chen Chen, Daniele Enrico Asoni, David Barrera, George Danezis, and Adrian Perrig. HORNET: High-speed Onion Routing at the Network Layer. In *ACM CCS, 2015*, pages 1441–1454, 2015.

[10] NSW Electoral Commission. iVote refresh project for the 2019 NSW State election. https://www.elections.nsw.gov.au/NSWEC/media/NSWEC/Reports/iVote%20reports/iVote-Refresh.pdf, 2019.

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

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

[13] Chris Culnane, Aleksander Essex, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. Knights and Knaves Run Elections: Internet Voting and Undetectable Electoral Fraud. *IEEE Security & Privacy*, 17(4):62–70, 2019.

[14] 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.

[15] 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.

[16] Prastudy Fauzi and Helger Lipmaa. Efficient Culpably Sound NIZK Shuffle Argument Without Random Oracles. In *CT-RSA 2016, Proceedings*, pages 200–216, 2016.

[17] 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.

[18] 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.

[19] PUB FIPS. 186-4: Federal information processing standards publication. Digital Signature Standard (DSS). *Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD*, pages 20899–8900, 2013.

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

[21] Kristian Gjøsteen. The Norwegian Internet Voting Protocol. *VoteID 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[22] Thomas Haines and Johannes Müller. SoK: Techniques for Verifiable Mix Nets. *IACR Cryptology ePrint Archive*, 2020:490, 2020.

[23] Chloé Hébant, Duong Hieu Phan, and David Pointcheval. Linearly-Homomorphic Signatures and Scalable Mix-Nets. In *Public-Key Cryptography - PKC 2020, Proceedings. To appear*, 2020.

[24] 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.

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

[26] Shahram Khazaei and Douglas Wikström. Randomized Partial Checking Revisited. *CT-RSA 2013, Proceedings*, volume 7779 of *Lecture Notes in Computer Science*, pages 115–128. Springer, 2013.

[27] 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.

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

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

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

[31] Ralf Küsters, Tomasz Truderung, and Andreas Vogt.

Clash Attacks on the Verifiability of E-Voting Systems. In *IEEE S&P 2012*, pages 395–409, 2012.

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

[33] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally Scaling Strong Anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles, 2017*, pages 406–422, 2017.

[34] Hemi Leibowitz, Ania M. Piotrowska, George Danezis, and Amir Herzberg. No Right to Remain Silent: Isolating Malicious Mixes. *IACR Cryptology ePrint Archive*, 2017:1000, 2017.

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

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

[37] NIST. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography (accessed 24.11.2019).

[38] State Electoral Office of Estonia. General Framework of Electronic Voting and Implementation thereof at National Elections in Estonia, Jun 2017.

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

[40] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The Loopix Anonymity System. In *USENIX Security 2017*, pages 1199–1216, 2017.

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

[42] Scytl. Scytl sVote Protocol Specifications – Software version 2.1 – Document version 5.1, 2018.

[43] The state of Geneva. Chvote. https://github.com/republique-et-canton-de-geneve/chvote-protocol-poc, 2018.

[44] Martin Strand. A Verifiable Shuffle for the GSW Cryptosystem. *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Revised Selected Papers*, pages 165–180, 2018.

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

[46] Raphael R. Toledo, George Danezis, and Isao Echizen. Mix-ORAM: Using Delegated Shuffles. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 51–61, 2017.

[47] Verificatum Mix Net (VMN). https://www.verificatum.org/html/product_vmn.html (accessed 09.01.2020).

[48] Douglas Wikström. A Universally Composable Mix-Net. *TCC 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 317–335. Springer, 2004.

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

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

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

APPENDIX A
FORMAL DEFINITION OF GOAL $\gamma(k, \varphi)$

In this section, we formally define the goal $\gamma(k, \varphi)$ which we have described on a high level in Section IV. This goal can be defined for an arbitrary result function ρ that takes as input a vector of input messages (as provided by the senders) and then outputs the overall result (e.g., a vector of plain messages or of encrypted messages). Recall that, on a high level, $\gamma(k, \varphi)$ is achieved if less than k honest inputs are manipulated in case the trust assumptions (modeled by) φ hold true.

Definition 2 (Goal $\gamma(k, \varphi)$): Let $P(n_{\mathsf{S}}, n_{\mathsf{MS}}, \mu)$ be a mix net protocol, let π be an instance of $P(n_{\mathsf{S}}, n_{\mathsf{MS}}, \mu)$, and let r be a run of π. Let $\mathsf{S}_1, \ldots, \mathsf{S}_{n_{\mathsf{S}}^{\mathsf{hon}}}$ be those senders that are honest in r. Let $\vec{m} = m_1, \ldots, m_{n_{\mathsf{S}}^{\mathsf{hon}}}$ be the plaintext inputs of these senders in r, chosen according to μ. Then, $\gamma(k, \varphi)$ is satisfied in r if either (a) the trust assumption φ does not hold true in r, or if (b) φ holds true in r and there exist valid messages $\tilde{m}_1, \ldots, \tilde{m}_{n_{\mathsf{S}}}$ such that the following conditions hold true:

- The multiset $\{\tilde{m}_1, \ldots, \tilde{m}_{n_{\mathsf{S}}}\}$ contains at least $n_{\mathsf{S}}^{\mathsf{hon}} - k$ elements of the multiset $\{m_1, \ldots, m_{n_{\mathsf{S}}^{\mathsf{hon}}}\}$.
- The mix net outcome as published in r (if any) equals to $\rho(\{\tilde{m}_1, \ldots, \tilde{m}_{n_{\mathsf{S}}}\})$.

If φ does not hold true in r and no outcome is published in r, then $\gamma(k, \varphi)$ is not satistied in r.