

## RESEARCH ARTICLE

# Toward Round-Efficient Verifiable Re-Encryption Mix-Net

MYUNGSUN KIM 

Department of Mathematics, Gachon University, Seongnam 13120, Republic of Korea

e-mail: msunkim@gachon.ac.kr

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2017R1D1A1B04035209 and Grant NRF-2022R1F1A1062806.


**ABSTRACT** Re-encryption mix-nets (RMNs) provide an efficient cryptographic anonymous channel for useful applications such as e-voting and web browsing. Many studies have been devoted to achieving practically efficient RMN protocols, but less attention has been paid to dealing with their round efficiency than to computation and communication measures. However, in many interactive cryptographic protocols, network latency governs the overall execution time. Because e-voting systems are particularly interaction intensive, the design of a round-efficient RMN protocol is of particular interest. We propose a constant-round RMN protocol in a three-party model that consists of senders, mix servers and some number of receivers. Here, the main role of the receivers is to jointly decrypt a list of ciphertexts obtained from the mixing stage. Such an explicit three-party model is most suitable for e-voting applications. We define an ideal three-party RMN in the universally composable (UC) framework. We then present a constant-round RMN protocol based on the standard assumptions and prove that it UC-realizes the ideal three-party RMN with respect to a static adversary that can corrupt a minority of mix servers, disallowing receivers who collude with other players. We implemented and evaluated our RMN protocol over a various range in the number of senders and mix servers. Our evaluation shows that our protocol runs up to  $2.5\times$  faster than Universal RMN protocol. Besides, we provide a detailed theoretical analysis of our protocol in terms of computation, transmission, and round efficiency.

**INDEX TERMS** Additive secret sharing, homomorphic encryption, re-encryption mix-nets, round efficiency.

## I. INTRODUCTION

Mix networks (for short, mix-nets) are a cryptographic tool for establishing private communication channels in a wide range of applications, for example, secure e-voting systems [1], anonymous e-mail [2], and location privacy in mobile networks [3]. The mix-net of Chaum [4] is run among a set of senders and a set of mix servers, and it works as follows: each sender provides its input to the mix servers, which then privately shuffle all inputs and ultimately publish them in random order.

There are roughly two basic flavors of mix-nets. The first class is known as a *re-encryption mix-net* (RMN). In this type

The associate editor coordinating the review of this manuscript and approving it for publication was Alba Amato .

of mix-net, both inputs and outputs are ciphertexts under the public key of some semantically secure cryptosystem with the homomorphic property that allows re-encryption without knowing the corresponding private key (e.g., El Gamal [5] and Paillier [6]). Each mix server re-encrypts the inputs and then permutes them. Because the output is still in the ciphertext, other authorities that had generated a public and private key pair need to decrypt it to produce the set of original messages. The mix servers can take over this role in some cases.

The second class is known as decryption mix-nets (DMNs), which were originally designed by Chaum [4]. The inputs to the mix-net are ciphertexts produced through interactive encryption under the public keys of individual servers. When processing inputs, each mix server decrypts the

layer corresponding to its own public key in each ciphertext and then permutes the resulting ciphertexts. Thus, the final output is produced in cleartext.

Of the two types of mix-nets, our main interest is the RMN because of the weaknesses of the DMN, including the requirements of multiple encryptions by the sender and a predefined order of decryption. In particular, we primarily pay attention to the *round efficiency* of RMN protocols. Round efficiency is of great importance because network latency usually has an effect several magnitudes greater than that of local computation costs. The round complexity usually dictates the time cost of a cryptographic protocol, and much research effort has been devoted to finding round-optimal protocols (e.g., see [7], [8], [9] in the MPC area). Nevertheless, little attention has been paid to handling the round efficiency problem in the RMN field. Thus, we focus on the following simple question (Q1).

*Is it possible to construct a constant-round RMN protocol? (Q1)*

### A. PREVIOUS WORK ON THIS QUESTION

In the research on RMNs, just a couple of papers have suggested methods of improving round efficiency in different stages of the RMN. The universal RMN protocol by Golle *et al.* [10] removes the joint decryption stage from the classical RMN by sending a pair of two El Gamal ciphertexts, where one contains a message and the other contains the public key of the receiver. Relying on Abe's observation that an RMN does not require sequential ordered mixing [11], Golle and Juels [12] proposed a method to perform mixing in parallel, which is called the parallel mix-net.

While these efficiency techniques clearly contribute to reducing network latency in mix-nets, their round efficiency still depends on the number of mix servers. For example, the universal RMN aims to reduce network latency during the group decryption stage. Thus, the round complexity of the universal RMN is linear in the number of mix servers  $n_m$  (i.e.,  $O(n_m)$ ). Moreover, the strong point of the universal RMN can be a weakness in certain circumstances (e.g., when there are multiple receivers); since the mix server outputs a list of only message-containing entries, the receiver needs to exhaustively search every output list received for possible messages encrypted under its public key. The parallel mix-net is somewhat complex. Indeed, the parallel mix-net requires  $\tilde{n}_m$  rounds for mixing (see Figure 3 in [12]), where the parameter  $\tilde{n}_m$  is the number of corrupt mix servers. Clearly,  $\tilde{n}_m < n_m$ , but additionally, the round complexity depends on the threat model, which determines the threshold  $\tilde{n}_m$ . Consequently, prevailing cases such as an honest majority (i.e.,  $\tilde{n}_m < \frac{n_m}{2}$ ) still have  $O(n_m)$  round complexity, while in some extreme cases (e.g.,  $\tilde{n}_m = n_m - 1$ ), almost constant rounds can be achieved. Of course, their benefits in terms of latency are very clear.

### B. PROBLEM STATEMENT

We refer to the problem of designing an RMN protocol with constant rounds across mix servers as the problem of finding a

round-efficient RMN. More formally, a round-efficient RMN protocol takes as input a list of ciphertexts and outputs a list of plaintexts while hiding the correspondence between the inputs and outputs within a constant number of rounds. Ensuring efficiency in rounds during the mixing stage makes the problem a nontrivial one.

We want to solve the problem in a model that consists of  $n_s$  senders that each have a plaintext,  $n_m$  mix servers, and  $n_r$  receivers. As usual, a mix server takes as input a list of ciphertexts from senders or another mix server, re-encrypts these ciphertexts, and outputs a new list of ciphertexts in a permuted order. Then, a quorum of receivers that share the private key decrypts the final list and outputs a set of plaintexts. In this work, we explicitly separate receivers from mix servers. This explicit three-party structure for the RMN is widely accepted in applications such as e-voting systems, where the tallying authorities can be viewed as the receivers (e.g., [13]).

### C. OUR CONTRIBUTIONS AND KEY IDEAS

Inspired by these challenges, the primary contribution of this paper is a round-efficient RMN protocol in the three-party model. We simply call it a three-party RMN (3RMN) protocol. Specifically, the contributions of this work are summarized below.

- We give the formal 3RMN protocol definition and the notion of security in the 3RMN and instantiate it by making use of several cryptographic tools (more precisely, they should be universally composable). In particular, we show that our 3RMN protocol requires only  $O(1)$  round complexity in the number of mix servers.
- To prove the practical effectiveness of our approach, we implement our RMN protocol. Our experimental results show that our protocol runs upto  $2.5\times$  faster than our competitor [10].
- We formally prove the correctness and security of our 3RMN protocol in the universally composable (UC) framework [14], guaranteeing that our 3RMN protocol is secure when composed with other UC-realized primitives.

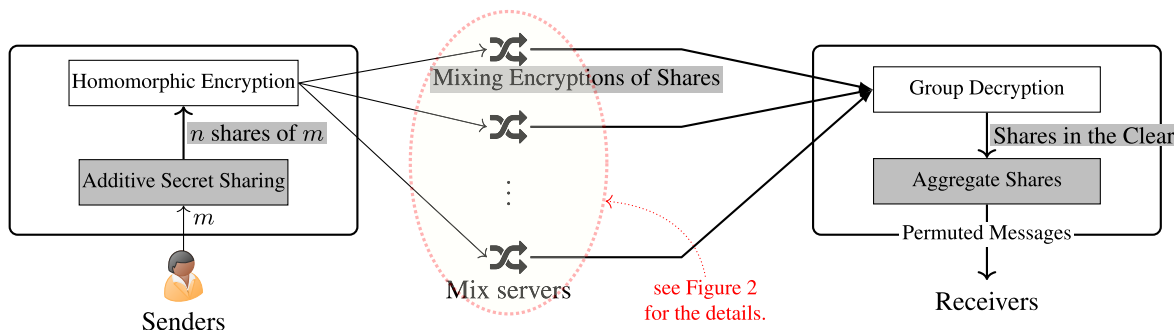
#### 1) OVERVIEW OF OUR APPROACH

In the design of an RMN, encryption and secret permutation are central primitives used to hide the correspondence between messages and senders. We also begin with a threshold El Gamal encryption [5] owing to its good capabilities, including distributed key generation, joint decryption, and multiplicative homomorphism.

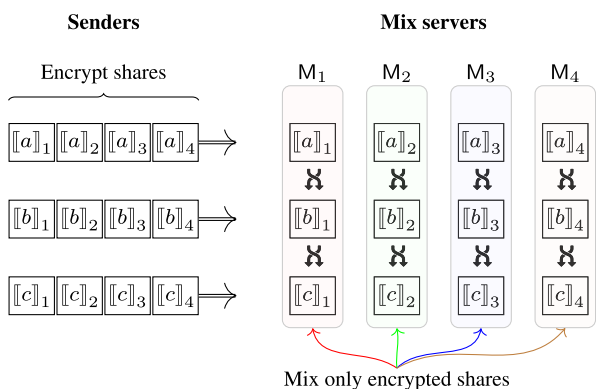
Our first attempt at a round-efficient 3RMN protocol is to have each of the mix servers shuffle the input ciphertexts from all senders and send the resulting list to the receivers without re-encrypting the ciphertexts in a cascade manner. However, after decrypting all lists, the input messages are duplicated  $n_s$  times, which is obviously undesirable. Even in a case in which decrypting one of the  $n_s$  lists is allowed, it is

**TABLE 1. Complexity comparison.** Here **S**: sender, **M**: mix server, **R**: receiver, **E**: modular exponentiation,  $a \in \mathbb{G}_q$ :  $a$  elements in  $\mathbb{G}_q$ ,  $n_s$ : the number of users,  $n_m$ : the number of mix servers,  $\tilde{n}_m$ : the number of corrupt mix servers,  $\Delta$ : conditionally satisfied,  $\times$ : not satisfied, and  $\circ$ : satisfied.

	Computation costs			Communication costs		Round	UC Security
	S	M	R	S	M		
Universal mix [10]	$4\mathbb{E}$	$2n_s\mathbb{E}$	$n_s\mathbb{E}$	$4\mathbb{G}_q$	$4n_s\mathbb{G}_q$	$n_m + 1 = O(n_m)$	$\Delta$
Parallel mix [12]	$2\mathbb{E}$	$2n_s \left( \frac{(\tilde{n}_m+1)}{n_m} \right) \mathbb{E}$	$n_s\mathbb{E}$	$2\mathbb{G}_q$	$2n_s \left( \frac{(\tilde{n}_m+1)}{n_m} \right) \mathbb{G}_q$	$2(\tilde{n}_m + 1) = O(\tilde{n}_m)$	$\times$
Ours	$2n_m\mathbb{E}$	$2n_s\mathbb{E}$	$n_s n_m\mathbb{E}$	$2n_m\mathbb{G}_q$	$2n_s\mathbb{G}_q$	$3 = O(1)$	$\circ$



**FIGURE 1. Schematic overview of our RMN protocol.** Shaded areas indicate components introduced by our solution and thick arrow lines indicate a vector of two or more messages are transmitted.



**FIGURE 2. Key insight of achieving constant rounds.** For a message  $a$ ,  $\llbracket a \rrbracket_i$  indicates the  $i$ -th additive share of  $a$  and  $\llbracket a \rrbracket$  means an ElGamal ciphertext of  $a$ . By  $\Rightarrow$ , we mean that a sender transmits its  $i$ -th share to the  $i$ -th mix server.

ambiguous to determine which of the  $n_s$  lists of ciphertexts should be chosen for the decryption stage.

Keeping only this structure, our second attempt is to remove useless duplications. To this end, we introduce secret sharing and make the sender split the input message into a set of shares before encrypting it. Then, the mix servers shuffle the list of ciphertexts as before, but each ciphertext carries its shares rather than the message itself. However, after decrypting them, there is no efficient way to restore the original messages.

To fix this problem, we make use of a unique padding method. More specifically, the sender concatenates random padding to every share of its message. This is the prototype of our round-efficient 3RMN protocol, which is depicted in Figure 1. In what follows, we give an example to show a key idea to achieve a constant round RMN protocol.

2) A TOY EXAMPLE

The example considers just three senders and four mix servers since the receivers are the same as those of existing RMN solutions.

First, the senders encrypt *additive shares* of their message but not the message itself. For example, the 1st sender begins with writing her message  $a$  into four shares  $a = \bigoplus_{i=1}^4 \llbracket a \rrbracket_i$ , where 4 indicates the number of mix servers. Then for each  $i \in \{1, \dots, 4\}$ , encrypt and send each of shares  $\llbracket a \rrbracket_i$  to the counterpart mix server  $M_i$ .

Next, each mix server just *mixes a list of encrypted shares* by applying re-randomization and a private permutation *without communicating other mix servers*. The remaining steps are the same as the conventional RMN protocols.

D. LIMITATIONS

As an unavoidable tradeoff between efficiency measures, our approach increases the overhead of computation and communication costs. As discussed above, since the senders transmit ciphertexts as many as the number of mix servers, they have to invoke the ElGamal encryption algorithm  $n_m$  times more than the classical RMN protocol. The computation and communication costs on the side of the mix servers are the same as before. Moreover, when the number of input ciphertexts increases by  $n_m$  times, the overhead of the receivers grows at the same rate.

Thus, when deploying our RMN protocol in a real-world application, one needs to identify the main bottleneck of the application. If the bottleneck is heavy computation such as modular exponentiation in an algebraic group of large size, our solution is not a good choice. On the other hand, when the bottleneck is network latency due to narrow bandwidth

and slow transmission time, our solution is expected to have a better execution time than existing RMN protocols.

### E. RELATED WORK ON RMNS

The re-encryption mix-net was first developed by Park *et al.* in [15] to fix the drawbacks of the decryption mix-net [4] and the hybrid mix-net [16]. See [17], [18] for a comprehensive survey of mix-nets. Since Park *et al.*'s proposal, much research has focused on RMNs. In a classical RMN, a sender  $S_i$  only has to compute a single encryption for all the mix servers as  $\text{ElG.E}(pk, m_i, r_i)$ , where  $\text{ElG.E}(\cdot, \cdot, \cdot)$  is the El Gamal public key cryptosystem under the public key  $pk$  used to encrypt  $m_i$  with a randomizer  $r_i$ . In particular, since no predefined order of mix servers is required for the RMN, any mix server  $M_j$  can compute  $\text{ElG.E}(pk, m_i, r_i + \rho_j)$  on the  $S_i$  input, where  $\rho_j$  is a randomizer of  $M_j$  for re-encryption of the  $S_i$  input. After the list of inputs is re-encrypted and permuted, a mix server  $M_j$  broadcasts the mixed list to the remaining mix servers for further mixing. The mixing stage ends at the last mix server  $M_{n_m}$ , where  $n_m$  is the number of mix servers. Then the group of receivers may perform a joint decryption stage to output a set of permuted messages. We would like to emphasize that in the RMN design, the mix servers do not have to share the private key. Instead, the public and private keys can be those of the receivers, where the public key  $pk$  is known to the senders and the mix servers. This type of design is widely used in applications such as e-voting, where a set of authorities receives the mix-net output.

One of the weaknesses of the classical RMN is that to support multiple receivers, an additional stage to share the private key is needed. This drawback is addressed with an improved variant, called universal re-encryption mix-net, proposed by Golle *et al.* in [10]. In this RMN, the sender  $S_i$  broadcasts two El Gamal ciphertexts, one containing the plaintext message  $m_i$  and the other containing the public key of the receiver used to encrypt  $m_i$ , i.e.,  $\text{ElG.E}(pk, m_i, r_i) \parallel \text{ElG.E}(pk, 1, \gamma_i)$ . The remaining  $n_m - 1$  mix servers repeat the re-encryption operation with different randomizers. However, the receiver should perform an exhaustive search on every output list from the mix servers received for possible plaintext messages encrypted under its public key  $pk$ .

Despite various optimization techniques (e.g. see [55], [56]), the basic RMN protocol is inherently inefficient in its operation. For example, Hébanat *et al.* [60] and Killer *et al.* [61] most recently proposed a quite efficient mix-net scheme; still their round complexity is linear to the number of mix servers. Since Abe observed that unlike the DMN, sequential ordered mixing is not necessary [11] in the case of an RMN, Golle and Juels [12] utilized this observation in the efficient design of an RMN where the mix servers perform mixing in parallel. Hence, the authors called this a parallel mix-net. Such a mix-net enjoys considerable improvement in network latency due to the parallelizing technique. More specifically, each of the mix servers is assigned a random subset of the input list; i.e., each subset contains  $\ell/n_m$  inputs,

where  $\ell$  is the size of the list. Then, the mix servers perform the following steps:

- 1) Each mix server mixes a given subset of size  $\ell/n_m$ .
- 2) The mix servers perform  $\tilde{n}_{n_m}$  rounds of rotations, where  $\tilde{n}_{n_m}$  is a threshold parameter less than  $n_m$ . Here, each rotation involves a modulo operation with  $n_m$ ; thus, the mix server  $M_{j-1}$  transmits its mixed output list to  $M_j$ , while  $M_j$  transfers its mixed output list to  $M_{j+1}$ , and so on.
- 3) After completing  $\tilde{n}_{n_m} - 1$  rounds, each mix server retains a random fraction  $\frac{1/\tilde{n}_{n_m}}{n_m}$  of its outputs and sends equal random portions of the remaining outputs to each of the  $n_m - 1$  mix servers. Thus,  $M_j$  receives  $\frac{\ell}{n_m^2}$  inputs from each of the remaining mix servers, receiving a total of  $\frac{\ell n_m}{n_m^2} = \frac{\ell}{n_m}$  inputs.
- 4) Steps 1 and 2 are repeated. Then, the resulting output from the  $n_m$  mix servers is the final output.

We note that the parallel mix-net requires a total of  $2(\tilde{n}_{n_m} - 1) + 2 = 2\tilde{n}_{n_m}$  rounds of mixing. Consequently, the parallel mix-net has  $O(\tilde{n}_{n_m})$  round complexity.

### 1) THE OUTLINE OF THIS PAPER

The rest of the paper is organized as follows: We present the system model, the security assumptions and the adversarial model of our protocol in Section II. Ideal primitives are provided in Section III, followed by the full construction in Section IV. Section V presents a formal analysis. First, Section V-A provides the asymptotic and concrete performance of our proposal, and then the formal security proof is given in Section V-B. Concluding remarks are given in Section VII.

## II. MODELS, DEFINITIONS, AND TOOLS

This section mainly aims to present cryptographic primitives and related definitions for the secure design of our RMN protocol. We begin by introducing notation used throughout the paper. We also define the system model and the adversarial model that our construction assumes throughout this paper.

*Notation:* Let  $[n]$  denote the set  $\{1, 2, \dots, n\}$ . Letting  $\mathbf{v}$  be a vector with elements  $\mathbf{v} = (v_1, v_2, \dots, v_n)$ , we use the notation  $\mathbf{v}[i]$  to index the  $i$ th element  $v_i$ . We use bold uppercase letters such as  $V$  to denote matrices and sometimes identify a matrix with its ordered set of column vectors. Thus, we mean by  $V = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_\ell)$  an  $\ell \times n$  matrix, and similarly,  $V[i, j]$  means the  $j$ -th element  $v_i[j]$  of the  $i$ -th vector. For any integer  $a$ , we denote by  $|a|$  the length of  $a$  in bits. For a finite set  $R$ , we let  $r \stackrel{\$}{\leftarrow} R$  be an element that is sampled uniformly at random from  $R$ .

We use  $\kappa$  to denote the computational security parameter (e.g.,  $\kappa = 80$ ). A function  $\nu : \mathbb{N} \rightarrow [0, 1]$  is negligible if it tends toward zero faster than  $1/n^\kappa$  for every fixed constant  $\kappa$ . We then use  $\text{poly}(\kappa)$  and  $\text{negl}(\kappa)$  to denote unspecified polynomial and negligible functions in  $\kappa$ , respectively. Let  $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$  and  $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$  be ensembles. Two ensembles  $X$  and  $Y$  are computationally indistinguishable, denoted by  $\equiv$ , if for every probabilistic polynomial-time



(PPT) algorithm  $\mathcal{D}$  and for all  $\kappa \in \mathbb{N}$ , there exists a negligible function  $\text{negl}$  such that

$$|\Pr[\mathcal{D}(X_\kappa, 1^\kappa) = 1] - \Pr[\mathcal{D}(Y_\kappa, 1^\kappa) = 1]| < \text{negl}(\kappa).$$

## A. MODELS

In this section, we formally describe the target system model and the adversary model we consider in this work, as well as our design goals.

### 1) SYSTEM MODEL

We follow the three-party system model for the re-encryption mix-net specified by Haines and Müller [18]. The system model consists of three participants: senders, mix servers, and receivers, assuming a publicly append-only bulletin board (PBB). We note that in [18], the authors refer to receivers as trustees. Each sender, denoted by  $S_{i \in [n_s]}$  with  $n_s \geq 2$ , owns an original input message  $m_i$ . Let  $n_m$  be the number of mix servers, and let  $M_{j \in [n_m]}$ , with  $n_m \geq 2$ , denote a mix server. Receivers are dedicated servers that jointly decrypt a permuted list of original messages.

A *round* is a duration during which mix servers locally perform mix operations without interacting with other mix servers (or senders). At the end of a round, mix servers may exchange a list of encryptions. From a PBB, we assume that there is a broadcast channel whereby users send messages to all other users in a single round.

### 2) THREAT MODEL

We need to restrict the adversary, simply because if all parties in a protocol are corrupted, no protocol is secure. Thus, we limit the power of the adversary in terms of how many parties it can corrupt as well as when they are corrupted.

Our work disallows an adversary from corrupting more than half of the mix servers; i.e., more than half of the players in the mix server type are honest. We consider static corruption; i.e., the adversary selects which party to corrupt before the protocol starts. However, corrupted parties are active adversaries who may refuse to follow the protocol's instructions. Moreover, it is assumed that there is no collusion between receivers and other entities (senders and mix servers).

Informally, our security guarantee is that the adversary cannot identify any individual message as being from a certain sender. Formal security definitions and guarantees are provided in Sections II-D and V-B, respectively. We prove that our protocol is secure in the UC framework, ensuring a safe composition with other UC protocols. However, we note that a malicious adversary colluding with any receiver can disrupt the protocol. We do not attempt to prevent malicious players from causing the protocol to abort.

## B. DEFINITIONS

### 1) HARDNESS ASSUMPTIONS

Our construction relies on the decisional Diffie-Hellman (DDH) assumptions, which are formalized as follows:

*Definition 1:* We say that the DDH problem is hard relative to  $\mathbb{G}_q$  if, for all PPT algorithms  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that as shown in the equation at the bottom of the page, where  $\mathbb{G}_q$  is a group of order  $q$  and the probabilities are taken over the choices of  $g$  and  $a, b, c \in \mathbb{Z}_q^*$ .

An additional assumption we use is the discrete logarithm (DL) problem, formalized as follows:

*Definition 2:* We say that the DL problem is hard relative to  $\mathbb{G}_q$  if, for all PPT algorithms  $\mathcal{A}$ , there exists a negligible function  $\text{negl}(\cdot)$  such that

$$\Pr[\mathcal{A}(\mathbb{G}_q, q, g, g^a) = a] \leq \text{negl}(\kappa),$$

where  $\mathbb{G}_q$  is a group of order  $q$  and the probabilities are taken over the choices of  $g$  and  $a \in \mathbb{Z}_q^*$ .

## C. CRYPTOGRAPHIC TOOLS

Our construction heavily relies on two cryptographic tools: secret sharing and homomorphic encryption. In addition, some zero-knowledge proof protocols are used for the purpose of verifiability.

### 1) SECRET SHARING

In [19], Shamir proposed the first  $(\ell_1, \ell_2)$ -threshold secret sharing scheme, where the  $(\ell_1, \ell_2)$ -threshold means that the original secret  $m$  is split into  $\ell_2$  different shares, and with any  $\ell_1$  shares, the original secret can be reconstructed, while any  $\ell_1 - 1$  shares reveal nothing about the secret. Thus,  $(\ell, \ell)$ -threshold secret sharing technology is suitable for our system model because mixing ciphertexts can be viewed as an outsourced protocol with  $\ell \geq 2$ .

As a popular variant of this method, Blakley, in [20], constructed additive secret sharing, which allows a given secret  $m$  to be decomposed into a sum of  $\ell$  random numbers. In this work, we use an additive  $\ell$ -out-of- $\ell$  secret sharing scheme, denoted as  $\binom{\ell}{\ell}$ -sharing, as follows: To share a value  $m$ , one party, as the dealer, chooses  $\ell$  random values  $\{r_j\}_{j \in [\ell]}$  under the constraint that  $m = \bigoplus_{j=1}^{\ell} r_j$  and distributes  $\ell - 1$  random shares among them to  $\ell - 1$  parties. Clearly, these  $\ell - 1$  random shares do not reveal anything about  $m$ . This approach is much more efficient for computation than Shamir's solution because it does not require expensive polynomial operations such as interpolation and multipoint evaluation. Furthermore, since our construction does not require a series of multiplication on shares, an  $\binom{\ell}{\ell}$ -secret sharing scheme is quite suitable for our purposes.

### 2) THRESHOLD HOMOMORPHIC ENCRYPTION

A public-key encryption (PKE) scheme is a triple of PPT algorithms, denoted by  $(\text{Kg}, \text{E}, \text{D})$ :

- $(pk, sk) \leftarrow \text{Kg}(1^\kappa)$  takes a security parameter  $\kappa \in \mathbb{N}$  as input. It outputs a pair of keys  $(sk, pk)$ . Here, the public key  $pk$  also defines a plaintext space  $M_{pk}$ , a randomness space  $R_{pk}$ , and a ciphertext space  $C_{pk}$ .
- $e \leftarrow \text{E}(pk, m, r)$  takes  $pk$  and a plaintext  $m \in M_{pk}$  as input. It outputs a ciphertext  $e \in C_{pk}$ . As usual, this

process is randomized using a randomizer  $r \in R_{pk}$ ; however, sometimes we simply write  $e \leftarrow E(pk, m)$ , omitting the randomness  $r$ .

- $m \leftarrow D(sk, e)$  takes  $sk$  and  $e \in C_{pk}$  as input. It outputs the plaintext  $m \in M_{pk}$ .

We say that a PKE scheme is correct if for any  $(pk, sk) \leftarrow \text{Kg}(1^\kappa)$  and any  $m \in M_{pk}$ , we have  $m = D(sk, E(pk, m))$ .

We say that a PKE scheme is *homomorphic* for the binary relations  $(\oplus, \otimes)$  if for all  $(pk, sk) \leftarrow \text{Kg}(1^\kappa)$ ,  $(M_{pk}, \oplus)$  forms a group and  $(C_{pk}, \otimes)$  forms a group and if for all  $e_1, e_2 \in C_{pk}$ ,  $D(sk, e_1 \otimes e_2) = D(sk, e_1) \oplus D(sk, e_2)$ . Moreover, given a ciphertext  $e$ , anyone can produce a different ciphertext  $e^*$  that carries the same plaintext as  $e$ . Therefore, given a homomorphic PKE scheme, we can define the rerandomization algorithm as  $\text{Re}(pk, e, r) := e \otimes E(pk, 0; r)$  for the identity  $0 \in M_{pk}$  and  $r \in R_{pk}$ .

Because it is undesirable for only a single party to be able to control the decryption of ciphertexts, a threshold version of a homomorphic PKE scheme needs to be used in a multiparty setting (e.g., [5], [6], [21], [22]). We use a threshold El Gamal cryptosystem due to its computational efficiency. A threshold PKE (TPKE) scheme has a different syntax than the underlying encryption scheme because of the additional requirement. We present the formal syntax of a TPKE scheme, which consists of a quadruple of PPT algorithms.

- $(pk, sk) \leftarrow \text{TKg}(1^\kappa, n)$  takes as input a security parameter  $\kappa$  and the number of parties  $n$ . It outputs a pair  $(pk, sk)$ , where  $pk$  is called the public key and  $sk = (sk_1, \dots, sk_n)$  is a vector of  $n$  private key shares. A party  $R_i$  is given the private key share  $sk_i$  and later uses it to compute a decryption share for a given ciphertext.
- $e \leftarrow E(pk, m, r)$ . This is the same as the underlying encryption algorithm.
- $e_i \leftarrow \text{TD}(pk, sk_i, e)$  takes as input the public key  $pk$ , the ciphertext  $e$ , and one of the  $n$  private key shares  $sk_i \in sk$ . It outputs the decryption share  $e_i$  of the plaintext or a special symbol  $\perp$ .
- $m \leftarrow \text{Agg}(pk, \{e_i\}_{i \in [n]})$  takes as input the public key  $pk$ , the ciphertext  $e$ , and  $n$  decryption shares  $\{e_1, \dots, e_n\}$ . It outputs a plaintext  $m$  or  $\perp$ .

We next formally define the notion of semantic security against chosen plaintext attacks (CPAs) [21]. To simplify the notation, we use  $a \leftarrow A^{\mathcal{O}_1, \mathcal{O}_2, \dots}(b_1, b_2, \dots)$  to denote an algorithm  $A$  that takes as inputs  $b_1, b_2, \dots$ , uses oracles  $\mathcal{O}_1, \mathcal{O}_2, \dots$  in a black-box manner, and outputs  $a$ . For a PPT adversary  $\mathcal{A}$ , we define the advantage function

$$\text{adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{A}, \kappa) := \left| \Pr \left[ b = b' \mid b' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{Kg}}, \mathcal{O}_E}(1^\kappa) \right] - \frac{1}{2} \right|$$

where  $\mathcal{O}_{\text{Kg}}$  samples  $(pk, sk) \leftarrow \text{Kg}(1^\kappa)$  and  $b \xleftarrow{\$} \{0, 1\}$  and outputs  $pk$ , and if  $|m_0| = |m_1|$ , then  $\mathcal{O}_E(m_0, m_1)$  returns  $e_b \leftarrow E(pk, m_b)$ . We say that the homomorphic PKE scheme is *semantically secure* against a CPA attack (IND-CPA) if for all PPT adversaries  $\mathcal{A}$ , the advantage  $\text{adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{A}, \kappa)$  is a negligible function of  $\kappa$ .

#### a: AN INSTANTIATION OF HOMOMORPHIC TPKE

Let  $p$  and  $q$  be large primes such that  $q|(p-1)$ ; let  $\mathbb{G}_q = \langle g \rangle$  be a subgroup of  $\mathbb{Z}_p^*$  of order  $q$  for a generator  $g$ . Because any element  $(1 \neq)g \in \mathbb{G}_q$  generates the group, the discrete logarithm of  $\alpha \in \mathbb{G}_q$  with respect to the base  $g$  is defined as usual. All computations in the remainder of this paper are modulo  $p$  unless otherwise noted.

Now, we describe a way to instantiate a threshold El Gamal PKE scheme.

- $(pk, sk) \leftarrow \text{EIG.TKg}(1^\kappa, n)$ : This produces the public parameter  $\text{pp} = (\mathbb{G}_q, g, p, q)$ , taking  $\kappa$  and  $n$  as input. Each party outputs  $\beta_i = g^{\alpha_i}$  for a random  $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$  and sets  $sk_i$  to  $sk_i = \alpha_i$ . It outputs the public key  $pk = (\text{pp}, \beta)$  and the secret key  $sk = \alpha$ , where  $\beta = \prod_{i=1}^n \beta_i$  and  $\alpha = \sum_{i=1}^n \alpha_i$ .
- $e \leftarrow \text{EIG.E}(pk, m, r)$ . This takes as input a message  $m$  and randomness  $r \xleftarrow{\$} \mathbb{Z}_q$  and outputs the ciphertext  $e = (g^r, m\beta^r)$ .
- $u_i \leftarrow \text{EIG.TD}(pk, sk_i, e)$ . Given a ciphertext  $e = (u, v)$ , a party  $R_i$  publishes her decryption share  $u_i = u^{\alpha_i}$ .
- $m \leftarrow \text{EIG.Agg}(pk, v, \{u_i\}_{i \in [n]})$ . This outputs the plaintext  $m = \frac{v}{\prod_{i \in [n]} u_i} = \frac{m \cdot \beta^r}{g^{r \cdot \sum_{i \in [n]} \alpha_i}}$ .

Since the threshold El Gamal PKE scheme is multiplicatively homomorphic, it has the re-randomization algorithm  $\text{EIG.Re}(pk, e, \gamma) := e \otimes \text{EIG.E}(pk, 1, \gamma)$ , where  $e \otimes \text{EIG.E}(pk, 1, \gamma) = (g^{r+\gamma}, m\beta^{r+\gamma})$  for the identity element  $1 \in \mathbb{G}_q$ .

#### 3) ZERO-KNOWLEDGE PROOFS

Our construction exploits zero-knowledge proofs (ZKPs) to ensure correct behavior. In practice, our protocol can be proven correct by using only so-called  $\Sigma$ -protocols, which only need three rounds of interaction [23], [24]. Unfortunately,  $\Sigma$ -protocols are not known to be zero-knowledge, but they satisfy the weaker property of honest-verifier zero-knowledge. This suffices for our purposes, as we can use the Fiat and Shamir heuristic [25] to make these proofs non-interactive.<sup>1</sup> As a consequence, the obtained proofs are indeed zero-knowledge in the random oracle model and consist of

<sup>1</sup>By a non-interactive proof, we mean that the proof generated by the prover can be verified without further interaction with the prover.

$$\left| \Pr[\mathcal{A}(\mathbb{G}_q, q, g, g^a, g^b, g^c) = 1 \mid \Pr[\mathcal{A}(\mathbb{G}_q, q, g, g^a, g^b, g^{ab}) = 1]] \right| \leq \text{negl}(\kappa)$$

only a single message.<sup>2</sup> For convenience, we report the formal definitions of ZKP, proof of knowledge and  $\Sigma$ -protocols in Appendix A.

Over the decades, there has been much work on securely instantiating various ideal ZKP primitives. In particular, Hazay and Lindell [27], and later Hazay and Nissim [28], described how to transform any ZKP protocol compiled from the  $\Sigma$ -protocol into an efficient protocol that is secure in the UC model. In this section, we thus describe only the  $\Sigma$  protocols that are required to achieve verifiability in our RMN protocol. The detailed specifications of relations and ideal ZKP functionalities will be given in Section III-C.

#### a: SENDER's ZKPs.

When receiving an encryption, the encryption should be checked to determine whether it has been faithfully generated. Given an instance of a PKE scheme with public key  $pk$ , a zero-knowledge proof of plaintext knowledge (ZKPT) allows a prover to prove knowledge of the plaintext  $m$  of ciphertext  $e \in \mathbf{E}(pk, m)$  to a verifier. Known efficient ZKPTs for El Gamal PKE schemes using the  $\Sigma$ -protocol have been demonstrated (e.g., [29], [30], [31]).

#### b: RECEIVER's ZKPs.

On the receiver's side, two things should be confirmed. First, each receiver should show that it owns the private key share  $sk_i$  of the private key  $sk$ . However, the key generation process is not a simple singleton of the ZKP protocol but requires several complicated subprotocols. Thus, we describe distributed key generation (DKG) in terms of the ideal functionality, as shown later in Section III-B (see Functionality 3). Pedersen showed such a DKG protocol for the El Gamal TPKE scheme [32]. A security flaw in Pedersen's DKG protocol was later corrected by Gennaro *et al.* in [33].

The second thing to confirm is strongly connected to the distributed decryption of the El Gamal TPKE scheme. Each receiver should prove that it correctly runs the threshold decryption algorithm without reconstructing the private key. Due to the importance of designing verifiable protocols, there have been many ZKP solutions to this problem (e.g., see [13], [34]).

#### c: MIX SERVER's ZKPs.

A shuffle is a permutation of a re-encryption of the input ciphertexts. By proving that such a shuffle is correct, a prover can verifiably rearrange a vector of ciphertexts

$$\mathbf{e} = (e_1, e_2, \dots, e_\ell)$$

$\hat{\mathbf{e}} = (\text{ElG.Re}(pk, e_{\pi(1)}), \dots, \text{ElG.Re}(pk, e_{\pi(\ell)}))$  without revealing the applied permutation  $\pi$  over  $[\ell]$ . Groth [35] proposed a very efficient way of proving the correctness of a shuffle of El Gamal encryptions in special honest-verifier

<sup>2</sup>The stronger assumption of a random oracle is only made for efficiency reasons. Alternatively, we could employ noninteractive ZKPs in the common random string model [26] to obtain noninteractiveness. In principle, our security proof also works in the standard model by utilizing interactive ZKPs.

zero-knowledge proofs. Indeed, there have been many solutions for this problem [36], [37], [38], [39], [40], [41].

#### D. THREE-PARTY RE-ENCRYPTION MIX-NET

Here, we present the ideal functionality of mix-net. In the ideal model, the type of mix-net does not matter; however, for our purposes, the ideal functionality of mix-net uses three distinct parties: senders, mix servers and receivers.

By  $(P : \text{message})$ , we mean that message was received on a player (e.g., sender)  $P$ 's input port via the environment  $\mathcal{Z}$ .

*Functionality 1 (Three-Party Mix-net):* Let  $n_s, n_m, n_r$  be positive integers and  $t \leq n_r$  be a threshold value. The ideal functionality for a three-party mix-net, denoted by  $\mathcal{F}_{3\text{RMN}}$ , running with senders  $S_1, \dots, S_{n_s}$ , mix servers  $M_1, \dots, M_{n_m}$ , receivers  $R_1, \dots, R_{n_r}$ , and a simulator  $\mathcal{S}$ , proceeds as follows:

- 1) Initialize lists  $L = \emptyset, L_j = \emptyset$  for  $j \in [n_m]$  and three index sets  $\mathcal{I}_S = \mathcal{I}_M = \mathcal{I}_R = \emptyset$ . Let there be a table  $T = \emptyset$  indexed by integers.
- 2) Upon receiving  $(S_i : \text{Send}, m_{ij})$  from  $\mathcal{Z}$ , if  $i \notin \mathcal{I}_S$ , then set  $\mathcal{I}_S = \mathcal{I}_S \cup \{i\}$ , choose a random value  $\delta_i$ , store  $\delta_i$  under index  $i$  in table  $T$ , and set  $L_j = L_j \cup \{m_{ij} \parallel \delta_i\}$ . Then, hand  $(S : S_i, \text{Send})$  to  $\mathcal{Z}$ .
- 3) Upon receiving  $(M_j : \text{Mix})$  from  $\mathcal{Z}$ , if  $j \notin \mathcal{I}_M$ , then set  $\mathcal{I}_M = \mathcal{I}_M \cup \{j\}$ . If  $|\mathcal{I}_M| \geq n_m/2$ , sort the list  $L_j$  lexicographically to build a new list  $L'_j$ , and hand  $(S : M_j, \text{Mix})$  to  $\mathcal{Z}$ .
- 4) Upon receiving  $(R_k : \text{Recover})$ , if  $k \notin \mathcal{I}_R$ , then set  $\mathcal{I}_R = \mathcal{I}_R \cup \{k\}$ . If  $|\mathcal{I}_R| \geq t$ , then restore  $m_i$  by using  $\delta_i$  in the table and  $(L'_1, \dots, L'_{n_m})$ . Then, set  $L = \{m_1, \dots, m_{n_m}\}$  and hand  $(S : R_k, \text{Output}, L), \{(R_k : \text{Output}, L)\}_{k \in [n_r]}$  to  $\mathcal{Z}$ .

In the application of electronic voting,  $m_i$  is the secret ballot of voter  $S_i$ . Thus, the secret message  $m_i$  is known only to  $S_i$ . Whereas in a conventional RMN protocol,  $|L_j| = 1$ , our RMN protocol sets  $|L_j| = n_m$  and  $|e_i| > 1$ , as will be seen.

#### 1) SECURITY OF A THREE-PARTY RMN

We define security using the UC framework. The UC framework is quite general and complex, and it is difficult to describe concisely. For reasons of parsimony, we thus omit a review of the UC security framework. For an in-depth discussion, we refer the reader to [14] and [42].

Roughly, the crux of the UC framework consists of the real (world) model, ideal (world) model, and a set of distinct hybrid models. The respective adversary in each model may corrupt a subset of the participants. In the real model, the parties execute a protocol  $\Pi$  in a certain environment  $\mathcal{Z}$  in which there is an adversary  $\mathcal{A}$  and ideal interaction. In the ideal-world model, all parties send their inputs to an ideal functionality  $\mathcal{F}$  (i.e., a trusted third party) to implement the protocol completely and truly. The parties in the ideal model forward any input to  $\mathcal{F}$ . A protocol in the ideal model is trivial and produces any output from  $\mathcal{F}$  as output. Note that an ideal functionality is considered secure by definition. Furthermore,

for the purpose of a seamless transition from the real model to the ideal model, various hybrid models are needed. Thus, a protocol running in the hybrid model has access to two or more ideal functionalities.

The definition of security relies on the simulation paradigm. Concretely, a protocol is said to securely realize an ideal functionality if for all real-world adversaries in the real model, there is an ideal adversary (i.e., a simulator) in the ideal model that has the same advantage. In particular, because the definition of security allows the secure composition of protocols, given a protocol that is secure in a hybrid model and protocols that securely realize all ideal functionalities in use, we can trivially construct a secure protocol in the real model.

We will describe in the next section all ideal functionalities for our construction.

### III. IDEAL FUNCTIONALITIES FOR THREE-PARTY MIX-NETS

In this section, we formally describe ideal functionalities composed with the main protocol. We inherit some of the ideal functionalities specified by Wikström [43]. However, because our construction is different in the type as well as the system model from Wikström's mix-net, we need to modify some ideal functionalities.

We start by defining the ideal PBB functionality with few modifications.

#### A. PBB FUNCTIONALITY

A PBB plays a particularly important role in our construction because our protocol extensively uses an authenticated broadcast channel. Assuming the setting of an honest majority with passive security, where more than half of the parties are honest, a PBB can be securely instantiated (see [44]).

*Functionality 2 (PBB):* The ideal PBB functionality, denoted by  $\mathcal{F}_{\text{PBB}}$ , runs with parties  $P_1, \dots, P_k$  and a simulator  $\mathcal{S}$ .

- 1)  $\mathcal{F}_{\text{PBB}}$  creates a table indexed on positive integers and initializes an index  $t = 1$ .
- 2) On receiving  $(P_i : \text{Write}, m_i)$  from the environment  $\mathcal{Z}$ ,  $(P_i, m_i)$  is stored in the table at the index  $t$ ,  $(\mathcal{S} : \text{Write}, t, P_i, m_i)$  is sent to  $\mathcal{Z}$ , and  $t = t + 1$  is set.
- 3) On receiving  $(P_j : \text{Read}, t)$  from the environment  $\mathcal{Z}$ , hand  $(\mathcal{S} : P_j, \text{Read}, t, P_i, m_i)$  and  $(P_j : \text{Read}, t, P_i, m_i)$  to  $\mathcal{Z}$  if a tuple  $(P_i, m_i)$  is found in the table at index  $t$ . Otherwise, hand  $(\mathcal{S} : P_j, \text{NoRead}, t)$  and  $(P_j : \text{NoRead}, t)$  to  $\mathcal{Z}$ .

The lemma below states that a PBB can be securely realized, assuming that more than half of the parties are honest.

*Lemma 1 (Goldwasser and Lindell [44]):* There exists a protocol  $\Pi_{\text{PBB}}$  that securely realizes the ideal functionality  $\mathcal{F}_{\text{PBB}}$  in the setting in which more than half of the parties are honest.

#### B. DKG FUNCTIONALITY

We present the ideal functionality for key generation in a distributed manner for the El Gamal TPKE scheme.

*Functionality 3 (DKG):* The ideal DKG functionality for the El Gamal TPKE scheme, denoted by  $\mathcal{F}_{\text{DKG}}$ , running with senders  $\mathcal{S}_1, \dots, \mathcal{S}_{n_s}$ , mix servers  $\mathcal{M}_1, \dots, \mathcal{M}_{n_m}$ , receivers  $\mathcal{R}_1, \dots, \mathcal{R}_{n_r}$ , and a simulator  $\mathcal{S}$ , proceeds as follows:

- 1) Initialize an index set  $\mathcal{I}_k = \emptyset$  for  $k = \{0, 1, \dots, n_r\}$ .
- 2) For each  $k \in [n_r]$ , receive  $(\mathcal{R}_k : \text{SKShares}, \alpha_k, \beta_k)$  such that  $\alpha_k \in \mathbb{Z}_q$  and  $\beta_k = g^{\alpha_k} \in \mathbb{G}_q$ , and set  $\mathcal{I}_0 = \mathcal{I}_0 \cup \{k\}$ .
- 3) Hand  $(\mathcal{S} : \text{PKShares}, \beta_1, \dots, \beta_{n_r})$  to  $\mathcal{Z}$ .
- 4) Hand  $\{(\mathcal{S}_i : \text{PKShares}, \beta_1, \dots, \beta_{n_r})\}_{i \in [n_s]}$  and  $\{(\mathcal{M}_j : \text{PKShares}, \beta_1, \dots, \beta_{n_r})\}_{j \in [n_m]}$  to  $\mathcal{Z}$ .
- 5) Hand  $\{(\mathcal{R}_k : \text{KeyPair}, \alpha_k, \beta_1, \dots, \beta_{n_r})\}_{k \in [n_r]}$  to  $\mathcal{Z}$ .
- 6) If  $(\mathcal{R}_{k'} : \text{Reconst}, \mathcal{R}_k)$  is received from  $\mathcal{Z}$ , set  $\mathcal{I}_k = \mathcal{I}_k \cup \{k'\}$ . If  $|\mathcal{I}_k| \geq \frac{n_r}{2}$ , hand  $(\mathcal{S} : \text{Reconsted}, \mathcal{R}_k, \alpha_k)$  and  $\{(\mathcal{R}_{k'} : \text{Reconsted}, \mathcal{R}_k, \alpha_k)\}_{k \in [n_r]}$  to  $\mathcal{Z}$ , and otherwise, hand  $(\mathcal{S} : \mathcal{R}_{k'}, \text{Reconst}, \mathcal{R}_k)$  to  $\mathcal{Z}$ .

As discussed in Section II-C2, for the El Gamal TPKE cryptosystem, Gennaro *et al.* [33] suggest a secure multi-party protocol for realizing this functionality. Conventionally, we omit the public parameter  $\text{pp}$  during key generation.

#### C. ZKP FUNCTIONALITY

As mentioned above, our protocol requires three idealized ZKP protocols, more precisely, zero-knowledge proof of knowledge (ZPK) protocols. From now on, we use ZPK in place of ZKP, unless explicitly stated otherwise. For consistency of representation, we define the ideal functionality for these ZPK protocols following Canetti *et al.* [45]. Since this functionality takes as a parameter a relation  $\mathcal{R}$ , we present the ideal ZPK functionality and then present three relations for the functionalities used in constructing our RMN protocol.

*Functionality 4 (ZPK):* Let  $\mathcal{L}$  be a language given by a relation  $\mathcal{R}$ . The ideal functionality, denoted by  $\mathcal{F}_{\text{ZPK}}^{\mathcal{R}}$ , for a zero-knowledge proof of knowledge of a witness  $\omega$  to a statement  $x \in \mathcal{L}$ , running with a prover  $P$  and a verifier  $V$ , proceeds as follows:

- 1) The functionality has an empty table.
- 2) On receiving  $(P : \text{Prove}, x, \omega)$  from  $\mathcal{Z}$ , store  $\omega$  in the table under the index  $(P, x)$  and send  $(\mathcal{S} : P, \text{Prove}, x, \mathcal{R}(x, \omega))$  to  $\mathcal{Z}$ . Discard all further messages from  $P$ .
- 3) On receiving  $(V : \text{Verify}, P, x)$ , read  $\omega$  by the tag  $(P, x)$  (the empty string if no witness is found), and hand  $(\mathcal{S} : V, \text{Accept}, P, x, \mathcal{R}(x, \omega))$  and  $(V : \text{Accept}, P, \mathcal{R}(x, \omega))$  to  $\mathcal{Z}$ .

The first relation we describe is a ZPK of the plaintext message  $m$  given a ciphertext message  $e = (u, v)$ , where  $u = g^r$ ,  $v = m\beta^r$ . We formally define the relation  $\mathcal{R}_{\text{PT}}$  below. As mentioned above, well-known examples of idealized ZPK protocols of plaintext messages include [29], [46].

*Definition 3 (Knowledge of Plaintext):* Define a relation  $\mathcal{R}_{\text{PT}} \subset (\mathbb{G}_q)^4 \times \mathbb{Z}_q^*$  as  $((g, \beta, u, v), r) \in \mathcal{R}_{\text{PT}}$  only if  $r = \log_g u$ .



We remark that the pair  $(\beta, v)$  appearing in the definition is not explicitly used, but we keep it for compatibility with previous methods.

The second relation for our purposes is a ZPK protocol of correct decryption, where, given a ciphertext  $e = (u, v)$ , a receiver  $R_i$  broadcasts  $d_i = u^{\alpha_i}$  so that  $\log_g \beta_i = \log_u d_i$ . The formal definition is given below.

*Definition 4 (Knowledge of Correct Decryption):* Define a relation  $\mathcal{R}_{CD} \subset (\mathbb{G}_q)^5 \times \mathbb{Z}_q^*$  as  $\langle (g, \beta_i, u, v, d_i), \alpha_i \rangle \in \mathcal{R}_{CD}$  only if  $\alpha_i = \log_g \beta_i = \log_u d_i$ .

One can view the relation  $\mathcal{R}_{CD}$  as an ideal counterpart to the  $\Sigma$  protocol shown by Cramer *et al.* [31]. In our protocol, the receiver  $R_i$  holds  $e = (u, v)$ ,  $d_i$  and  $\alpha_i$  such that  $d_i = u^{\alpha_i}$ , and  $\beta_i$  is a public component of  $pk$ .

Last, the shuffle relation is defined formally. More specifically, the shuffle relation used in the RMN corresponds to a correct permutation and re-encryption of a list of El Gamal ciphertexts. This relation can be treated as an ideal counterpart to the  $\Sigma$  protocols given by Groth [35], [47]. We continue to define the relation  $\mathcal{R}_{CS}$ .

*Definition 5 (Knowledge of Correct Shuffle):* Let  $\Phi_N$  be the set of permutations on  $[N]$  for an integer  $N \geq 2$ . Define a relation  $\mathcal{R}_{CS} \subset (\mathbb{G}_q)^2 \times (\mathbb{G}_q)^{2N} \times (\mathbb{G}_q)^{2N} \times \Phi_N \times (\mathbb{Z}_q^*)^N$  by

$$\langle (g, \beta, \{e_i\}_{i \in [N]}, \{e'_i\}_{i \in [N]}), (\pi, \{r_i\}_{i \in [N]}) \rangle \in \mathcal{R}_{CS}$$

only if  $\forall i \in [N] : (u'_{\pi^{-1}(i)}, v'_{\pi^{-1}(i)}) = (u_i g^{r_i}, v_i \beta^{r_i})$ , where  $e_i = (u_i, v_i)$ ,  $e'_i = (u'_i, v'_i)$  are El Gamal ciphertexts.

In our protocol, each mix server  $M_j$  has a permutation  $\pi \xleftarrow{\$} \Phi_N$  and a set of randomnesses  $\{r_1, \dots, r_N\} \xleftarrow{\$} (\mathbb{Z}_q^*)^N$ .

We use  $\mathcal{F}_{ZPK}^{\mathcal{R}_{PT}}$  to denote an ideal primitive of a ZPK protocol for the relation  $\mathcal{R}_{PT}$ , i.e., zero-knowledge proof of plaintext knowledge. With a similar purpose, we will use two additional pieces of notation  $\mathcal{F}_{ZPK}^{\mathcal{R}_{CD}}$  and  $\mathcal{F}_{ZPK}^{\mathcal{R}_{CS}}$  in later sections.

In what follows, we present our main protocol. Our description consists of an overview and the main protocol.

## IV. OUR CONSTRUCTION

In this section, we develop the main protocol for the three-party re-encryption mix (3RMN). First, we present our basic idea to address the round efficiency problem in 3RMN. We then give all the design details of our 3RMN protocol.

### A. OVERVIEW

For clarity, we first present our basic 3RMN protocol, hiding the details. Similar to existing RMN protocols, our RMN protocol requires a TPKE with the homomorphic property. However, the main tool that allows us to improve round efficiency is secret sharing. As discussed in the introduction, by encrypting shares of the message rather than a plaintext message itself, mix servers can avoid mixing in a cascade manner.

We focus on showing operational differences from existing RMN protocols on each entity.

### 1) KEY GENERATION

In this subprotocol, the public key that senders and mix servers later use is generated as in existing RMN protocols. The main difference is that we explicitly have receivers jointly create the key pair. In an e-voting application, the setup procedure will invoke this protocol.

### 2) MESSAGE ENCRYPTION

Let  $m_i$  be a plaintext message of sender  $S_i$ . Every known RMN protocol encrypts  $m_i$  as  $e_i$  and then sends  $e_i$  to the mix servers. In contrast, our RMN protocol first splits  $m_i$  into a set of additive shares  $\{\llbracket m_i \rrbracket_j\}$ . Then, each  $\{\llbracket m_i \rrbracket_j\}$  is encrypted and sent to the mix servers. This is one of the main characteristics that is different from those of known solutions. We remark that there are additional options for sending the ciphertexts to a set of mix servers regarding the number of ciphertexts and the choice of the target mix server.

### 3) MIXING

Our protocol also performs a shuffle process that consists of the permutation and re-encryption of ciphertexts given by senders. The important difference is that our protocol performs this process locally without interacting with other mix servers. This is the second main characteristic of our proposal.

### 4) MESSAGE DECRYPTION

Receivers jointly decrypt a set of ciphertexts and output a set of senders' input messages. Note that since the shares of the input messages have been encrypted, the receivers need to reconstruct all original plaintexts before producing a permuted list of plaintexts. To ensure the reconstruction is correct, during encryption, each sender should encrypt  $\llbracket m_{ij} \rrbracket \parallel \delta_i$  rather than  $\llbracket m_{ij} \rrbracket$ , where  $\delta_i$  is a unique random value.

## B. THE PROTOCOL

We now present a 3RMN protocol in the  $(\mathcal{F}_{PBB}, \mathcal{F}_{DKG}, \mathcal{F}_{ZPK}^{\mathcal{R}_{PT}}, \mathcal{F}_{ZPK}^{\mathcal{R}_{CD}}, \mathcal{F}_{ZPK}^{\mathcal{R}_{CS}})$ -hybrid model in which the players utilize an ideal PBB, ideal distributed El Gamal key generation, and ideal ZPK systems for three relations  $\mathcal{R}_{PT}$ ,  $\mathcal{R}_{CD}$ , and  $\mathcal{R}_{CS}$ . Our 3RMN is secure as long as a majority of mix servers are honest. Similar to other RMN systems (e.g., [43]), there are no limitations on the number of corrupted senders.

Our 3RMN protocol  $\Pi_{3RMN}$  consists of senders  $S_{i \in [n_s]}$ , a mix server  $M_{j \in [n_m]}$ , and receivers  $R_{k \in [n_r]}$ . We assume that all players agree on the public system parameter  $\mathbf{pp} = (\mathbb{G}_q, g, p, q)$  and that the public key  $pk$  is implicitly regarded as a tuple  $(\mathbf{pp}, \beta, \{\beta_k\}_{n_r})$ . Concretely, our mix-net protocol runs as follows:

**Sender  $S_i$ .** Each sender  $S_i$  performs the following:

- 1) Wait for  $(PKShares, \{\beta_k\}_{k \in [n_r]})$  from  $\mathcal{F}_{DKG}$  and compute the public key  $\beta = \prod_{k \in [n_r]} \beta_k$ .
- 2) Wait for an input  $(Send, \mathbf{m}_i)$ , where  $\mathbf{m}_i \in (\mathbb{G}_q)^{m_i}$ . Then, choose a random value  $\delta_i \in \{0, 1\}^{\text{poly}(\kappa)}$ ; for each

$j \in [n_m]$ , compute  $\llbracket m_{ij} \rrbracket$  such that  $m_i = \bigoplus_{j \in [n_m]} \llbracket m_{ij} \rrbracket$  and set  $m_{ij} = \llbracket m_{ij} \rrbracket \parallel \delta_i$ .

3) Compute  $e_{ij} = (u_{ij}, v_{ij}) = \text{EIG.E}(pk, m_{ij}, r_{ij})$  for each  $r_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$ .

4) Send each of  $\{(\text{Prove}, (g, \beta, u_{ij}, v_{ij}), r_{ij})\}_{j \in [n_m]}$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ .

5) Send  $\{(\text{Write}, (u_{ij}, v_{ij}))\}_{j \in [n_m]}$  to  $\mathcal{F}_{\text{PBB}}$ .

**Mix server  $M_j$ .** Each mix server  $M_j$  performs the following:

1) Wait for  $(\text{PKShares}, \{\beta_k\}_{k \in [n_r]})$  from  $\mathcal{F}_{\text{DKG}}$  and compute the public key  $\beta = \prod_{k \in [n_r]} \beta_k$ .

2) Form a list  $L_j = \{(u_{ij}, v_{ij})\}_{i \in [n_s]}$  based on the set  $\{(u_{i\ell}, v_{i\ell})\}_{i \in [n_s], \ell \in [n_m]}$  on  $\mathcal{F}_{\text{PBB}}$ .

3) For each  $(u_{ij}, v_{ij}) \in L_j, i \in [n_s]$ , do the following:

a) Send  $(\text{Verify}, S_i, (u_{ij}, v_{ij}))$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ .

b) Wait for  $(\text{Accept}, S_i, b_i)$  from  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ , where  $b_i \in \{0, 1\}$ . If any proof fails to be verified at  $M_j$  (i.e.,  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$  outputs 0 to  $M_j$ ), then  $M_j$  aborts.

4) Choose  $\gamma_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$  for each  $i \in [n_s]$  and  $\pi_j \xleftarrow{\$} \Phi_{n_s}$ , and compute  $e'_{ij} = \text{EIG.Re}(pk, e_{ij}, \gamma_{ij})$ , where  $e_{ij} = (u_{ij}, v_{ij}), e'_{ij} = (u'_{ij}, v'_{ij})$ .

5) Build a list  $L'_j = \{(u'_{\pi_j(i)}, v'_{\pi_j(i)})\}_{i \in [n_s]}$ .

6) Send  $(\text{Prove}, M_j, (g, \beta, L_j, L'_j), \pi_j, \{\gamma_{ij}\}_{i \in [n_s]})$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$ .

7) Send  $(\text{Write}, L'_j)$  to  $\mathcal{F}_{\text{PBB}}$  and output  $(\text{Mix}, M_j)$ .

**Receiver  $R_k$ .** Each receiver  $R_k$  performs the following:

1) Choose  $\alpha_k \xleftarrow{\$} \mathbb{Z}_q^*$  and send  $(\text{SKShares}, \alpha_k, \beta_k)$  to  $\mathcal{F}_{\text{DKG}}$ .

2) Wait for  $(\text{KeyPair}, \alpha_k, \beta_1, \dots, \beta_{n_r})$  from  $\mathcal{F}_{\text{DKG}}$  and compute  $\beta = \prod_{k \in [n_r]} \beta_k$ .

3) For each  $j \in [n_m]$ , form a list  $L_j = \{(u_{ij}, v_{ij})\}_{i \in [n_s]}$  from the table of entries on  $\mathcal{F}_{\text{PBB}}$ . For each  $i \in [n_s]$ , do the following:

a) Hand  $(\text{Verify}, S_i, (u_{ij}, v_{ij}))$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ .

b) Wait for  $(\text{Accept}, S_i, b_i)$  from  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ , where  $b_i \in \{0, 1\}$ . If  $b_i = 0$  (i.e., the proof of knowledge of the randomness used in  $u_{ij}$  fails to be verified at  $R_k$ ), then  $R_k$  aborts.

Then, if  $b_i = 1$  for all  $i \in [n_s]$ , set  $L_j^* = L_j$ ; otherwise,  $L_j^* = \perp$ , where  $\perp$  indicates the empty list.

4) Wait until all mix servers have written  $(\text{Forward}, L'_j)$  on  $\mathcal{F}_{\text{PBB}}$  and let  $L'_j = \{(u'_{ij}, v'_{ij})\}_{i \in [n_s]}$ . For each  $j \in [n_m]$ , do the following:

a) Hand  $(\text{Verify}, M_j, (g, \beta, L_j^*, L'_j))$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$ .

b) Wait for a response  $(\text{Accept}, M_j, b_j^*)$  from  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$ , where  $b_j^* \in \{0, 1\}$ .

If  $\forall j \in [n_m] : b_j^* = 1$ , then set  $L_j^* = L'_j$ ; otherwise,  $R_k$  aborts.

5) Wait until all receivers  $R_{k \in [n_r]}$  have written  $(\text{Recover}, \{L_j^*\}_{k \in [n_r]})$  on  $\mathcal{F}_{\text{PBB}}$ , and then for each  $j \in [n_m]$ , do the following:

a) Initialize  $D_j = \emptyset$ . For each  $i \in [n_s]$ , compute a partial decryption  $d_{ij}^* = \text{EIG.TD}(pk, \alpha_i, e_{ij}^*)$ , where  $e_{ij}^* = (u_{ij}^*, v_{ij}^*) \in L_j^*$ ; hand  $(\text{Prove},$

$(g, \beta_k, u_{ij}^*, v_{ij}^*, d_{ij}^*), \alpha_i)$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RCD}}$ ; and form the list  $D_j = D_j \cup \{(d_{ij}^*, e_{ij}^*)\}$ .

b) Hand  $(\text{Write}, D_j)$  to  $\mathcal{F}_{\text{PBB}}$ .

6) For all  $\ell \in [n_r] \setminus \{k\}$ , do the following:

a) Hand  $(\text{Verify}, R_\ell, (g, \beta_\ell, u_{i\ell}^*, v_{i\ell}^*, d_{i\ell}^*))$  to  $\mathcal{F}_{\text{ZPK}}^{\text{RCD}}$  and wait for a response  $(\text{Accept}, R_\ell, b_\ell^*)$  from  $\mathcal{F}_{\text{ZPK}}^{\text{RCD}}$ . If  $b_\ell^* = 1$ , then read the list  $D_\ell$  on a corresponding index  $t_\ell$  from  $\mathcal{F}_{\text{PBB}}$ ; otherwise,  $R_k$  aborts.

7) For each  $D_j = \{(d_{ij}^*, e_{ij}^*)\}_{j \in [n_m], i \in [n_s]}$ , do the following:

a) Compute  $m_{ij} = \text{EIG.Agg}(pk, v_{ij}^*, \{d_{i\ell}^*\}_{\ell \in [n_m]})$ .

8) For an  $n_s \times n_m$  matrix  $M = (m_{ij})$ , examining the padded random values  $\delta_i$  for each entry, recover a set of plaintext messages  $L = \{m_1, \dots, m_{n_s}\}$  and produce  $(\text{Output}, L)$ .

*Remark: One way to compute a random value used to recover a message is to use a pseudorandom function (PRF) parameterized by the message  $m_i$  and a secret key known only to  $S_i$ . For example, consider the advanced encryption standard (AES) in counter mode to instantiate such a PRF. In particular, the possibility that two senders will use the same random value is negligible.*

### C. FURTHER DISCUSSIONS

We explore possible variants of our 3RMN protocol. We expect that these variants may not only contribute to improving transmission costs but also give an additional option for verifiability.

#### 1) REDUCING TRANSMISSION AND COMPUTATION COSTS

As a trade-off of improving round efficiency, our protocol increases computation and transmission costs. The detailed analysis will be discussed in Section V-A below. First,  $n_m$  El Gamal ciphertexts are sent to the mix servers. This results in  $n_m$  invocations of the El Gamal encryption algorithm. Moreover, the receiver receives a list of  $n_s \cdot n_m$  El Gamal ciphertexts and thus performs  $n_s \cdot n_m$  rounds of the partial decryption and aggregation algorithms in total. In some applications, this computational overhead can be a burden.

One way to reduce such additional overhead in computation and communication costs is for the sender  $S_i$  to split  $m_i$  into additive shares less than  $n_m$ , for example,  $\log n_m$  or  $\sqrt{n_m}$ . Say  $\log n_m$  is used. Then, each sender transmits only  $\log n_m$  El Gamal ciphertexts, and thus the overhead on the receiver side reduces to  $n_s \log n_m$  with respect to the computation and communication costs. Because  $n_m \ll n_s$ , such a design choice can be useful in improving the practical performance of our 3RMN protocol.

We have another option for addressing this efficiency problem. Thus far, every sender has created the same number of additive shares from its plaintext message. However, there is an alternative in which each sender can choose a threshold value of additive shares depending on its circumstances. Thus, a sender that is restricted in computational resources can choose a smaller number of additive shares than resource-rich senders.

## 2) VERSATILITY OF RANDOM PADDING $\delta_i$

An additional benefit of our approach is that  $\delta_i$ , which was originally padded to recover the plaintext message  $m_i$ , can ensure the integrity of  $m_i$ . Specifically, after each receiver restores a permutation of plaintext messages,  $\{m_1, \dots, m_{n_s}\}$ , it can record  $\{(m_1, \delta_1), \dots, (m_{n_s}, \delta_{n_s})\}$  on the PBBs. Then, each sender may later check whether her random padding is correctly posted on the PBB. Basically, since the El Gamal encryption scheme is not nonmalleable [48], malicious mix servers can modify the plaintexts embedded in El Gamal ciphertexts.<sup>3</sup> The uniqueness of  $\delta_i$ , which is only known to the sender  $S_i$ , makes it infeasible for mix servers, given a ciphertext, to create another ciphertext with a related plaintext. This does not mean that our protocol can single out misbehaving mix servers. Instead, our technique can be considered a lightweight solution to prevent mix servers from maliciously performing homomorphic operations on ciphertexts.

## V. THEORETICAL EVALUATION

We provide a theoretical analysis of our 3RMN protocol to evaluate its computation, communication, and, more importantly, round efficiency. Then, we analyze the security of our solution.

### A. COMPLEXITY ANALYSIS

We first recall the parameters below. Let  $n_s$  be the number of senders,  $n_m$  be the number of mix servers, and  $n_r$  be the number of receivers. In the computation and communication analysis, we focus on the most expensive operations (i.e., modular exponentiation in  $\mathbb{G}_q$ ) in each procedure. In addition,  $|\mathbb{Z}_q^*|$  represents the bit length of an element in  $\mathbb{Z}_q^*$ , and  $|\mathbb{G}_q|$  is the bit length of an element in  $\mathbb{G}_q$ .

#### 1) ROUND COMPLEXITY

It is quite clear that our 3RMN protocol achieves constant rounds in the number of players. More specifically, each sender broadcasts its El Gamal ciphertexts to mix servers through a PBB. The mix servers output a list of permuted and re-encrypted El Gamal ciphertexts in a broadcast manner to receivers. The group decryption on the receivers' side is somewhat involved. Suppose that receiver  $R_k$  has a list of El Gamal ciphertexts  $E = (e_1, \dots, e_{n_m})$ , where each component  $e_{j \in [n_m]}$  is also a vector of El Gamal ciphertexts. To decrypt  $E$ , first,  $R_k$  partially decrypts  $E$  into  $D$  under its private key share  $\alpha_k$  and broadcasts  $D$  to the set of receivers through the PBB. At the same time,  $R_k$  reads the other receivers' broadcasted El Gamal lists. Then,  $R_k$  recovers a list of plaintext messages  $M = (m_1, \dots, m_{n_m})$  by simply aggregating counterpart El Gamal ciphertexts. Even if, due to transmission delay among the receivers, some receivers may have to wait to read lists of ciphertexts, the group decryption by the receivers incurs only a constant number of rounds (for details, see [49]). Consequently, our protocol has  $O(1)$  round complexity in

<sup>3</sup>Tsiounis and Yung [46, §7] studied a nonmalleable extension of the El Gamal encryption scheme.

the number of players. We note that all defined idealized primitives work in a noninteractive manner, so the round numbers consumed in these protocols are also constant. From our explanation thus far, we therefore obtain the following:

*Proposition 1: The three-party RMN protocol  $\Pi_{3RMN}$  presented in Section IV-B requires  $O(1)$  round complexity in the number of players, assuming a broadcast channel.*

*Remark: We should note that allowing a broadcast channel does not imply that we can obtain all  $n_m$  re-encryptions for a ciphertext at one time. The reason is that each mix server needs to wait for a re-encryption of  $n_m - 1$  ciphertexts from all other mix servers before it can broadcast its re-encryption on the PBB. In conclusion, this type of re-encryption requires  $O(n_m)$  rounds. See the definition of 'round' in Section II-A.*

#### 2) COMPUTATIONAL COMPLEXITY

We next calculate the amounts of computation needed to complete our 3RMN protocol and represent them in terms of asymptotic complexity. We count only the number of modular exponentiations used in the main protocol. In fact, all exponentiations for ZPK subprocedures are asymptotically dominated by those of receivers, as analyzed below.

Each sender  $S_i$  first generates a set of additive shares and encrypts each of them. Let  $n_m$  be the number of additive shares for the message  $m_i$ . Then, since  $n_m$  El Gamal ciphertexts are computed, in total,  $2n_m$  exponentiations in  $\mathbb{G}_q$  are required. The computational cost for executing  $\binom{n_m}{n_m}$ -secret sharing is quite small and is omitted. As the mix server performs only the El Gamal re-encryption algorithm, it computes two exponentiations in  $\mathbb{G}_q$ . Each mix server has an El Gamal ciphertext list of size  $n_s$ . The total number of exponentiations in  $\mathbb{G}_q$  is  $2n_s$ . Similarly, each receiver processes a ciphertext list of size  $n_m n_s$ ; thus, the total number of exponentiations is  $n_m n_s$ .

As a consequence, the total computation amount is  $2n_s n_m + 2n_s n_m + n_r (n_m n_s) = n_s n_m (n_r + 4)$ , and thus our protocol requires in total  $O(n_s n_m n_r)$  computations, omitting the computational costs of computing all idealized primitives, which contribute to only a constant factor in big- $O$  notation.

#### 3) COMMUNICATION COMPLEXITY

We measure the communication costs for our protocol by counting the number of El Gamal ciphertexts communicated among players. We do not count the transmission costs involved in all subprotocols for the same reason as above.

The sender  $S_i$  broadcasts  $n_m$  El Gamal ciphertexts; thus, the total number of transmissions is  $2n_m$  elements in  $\mathbb{G}_q$  (i.e.,  $2n_m |\mathbb{G}_q|$  bits). Each mix server  $M_j$  sends  $n_s$  El Gamal ciphertexts. The total number of transmissions is therefore  $2n_s |\mathbb{G}_q|$  bits. Last, each receiver broadcasts  $n_s n_m$  tuples consisting of an El Gamal ciphertext together with the corresponding partial decryption  $d_{ij}^*$ . Hence, it sends  $3n_s n_m$  elements in  $\mathbb{G}_q$  and thus  $3n_s n_m |\mathbb{G}_q|$  bits in total.

The total communication costs of our protocol amount to  $n_s \cdot (2n_m) + n_m \cdot (2n_s) + n_r (3n_s n_m)$  elements in  $\mathbb{G}_q$ . In conclusion, our protocol requires a communication complexity



of  $4n_{\mathcal{S}}n_{\mathcal{M}} + 3n_{\mathcal{S}}n_{\mathcal{M}}n_{\mathcal{R}} = n_{\mathcal{S}}n_{\mathcal{M}}(3n_{\mathcal{R}} + 4) = O(n_{\mathcal{S}}n_{\mathcal{M}}n_{\mathcal{R}})$  in the number of players.

**B. SECURITY ANALYSIS**

We continue to formally prove the security of our protocol. To do this, we show that for each adversary  $\mathcal{A}$  in the real model, there is a simulator (i.e., an ideal adversary)  $\mathcal{S}$  such that if the environment cannot distinguish between the two different interactions the protocol is secure. That is, the information learned by adversary  $\mathcal{A}$  in the real interaction and the information obtained by the simulator in the ideal interaction are indistinguishable. As a proof, it is only necessary to check whether the developed simulator has the ability to generate messages that are indistinguishable from the real-world interaction messages. Our work depends on the ideal functionalities described in Section III and thus is secure under the simulation paradigm.

*Proposition 2 (Correctness):* Let  $\mathbf{m} = (m_1, \dots, m_{n_{\mathcal{S}}})$  be a list of messages from senders. Suppose that all players participate honestly in the protocol. Then, the protocol  $\Pi_{3\text{RMN}}$  given in Section IV-B correctly outputs a permutation of  $\mathbf{m}$  at the end.

*Proof:* Let  $m_i$  be the message of sender  $\mathcal{S}_i$  for each  $i \in [n_{\mathcal{S}}]$ . Assume that  $\delta_i$  is padding chosen by  $\mathcal{S}_i$ . As specified in the protocol,  $\mathcal{S}_i$  computes  $e_{ij} = \text{EIG.E}(pk, m_{ij})$  so that  $m_i = \bigoplus_{j \in [n_{\mathcal{M}}]} \llbracket m_{ij} \rrbracket$  and  $m_{ij} = \llbracket m_{ij} \rrbracket \parallel \delta_i$ . Each mix server  $\mathcal{M}_j$  outputs a shuffled list  $L'_j = (e_{\pi_j(1)j}, \dots, e_{\pi_j(n_{\mathcal{S}})j})$  from the senders' ciphertexts, where  $\pi_j$  is a private permutation of  $\mathcal{M}_j$ . Here, an El Gamal ciphertext  $e_{\pi_j(i)j}$  carries a plaintext  $m_{\pi_j(i)j} = \llbracket m_{\pi_j(i)j} \rrbracket \parallel \delta_{\pi_j(i)}$ . During the group decryption, each receiver begins with a list of ciphertext lists  $L'_j$  from  $\mathcal{M}_j$ . Say the list is  $(L'_1, \dots, L'_{n_{\mathcal{M}}})$ . This list can be written in matrix form at the end of group decryption as follows:

$$\begin{pmatrix} \llbracket m_{\pi_1(1)1} \rrbracket \parallel \delta_{\pi_1(1)} & \dots & \llbracket m_{\pi_{n_{\mathcal{M}}}(1)n_{\mathcal{M}}} \rrbracket \parallel \delta_{\pi_{n_{\mathcal{M}}}(1)} \\ \llbracket m_{\pi_1(2)1} \rrbracket \parallel \delta_{\pi_1(2)} & \dots & \llbracket m_{\pi_{n_{\mathcal{M}}}(2)n_{\mathcal{M}}} \rrbracket \parallel \delta_{\pi_{n_{\mathcal{M}}}(2)} \\ \vdots & \ddots & \vdots \\ \llbracket m_{\pi_1(n_{\mathcal{S}})1} \rrbracket \parallel \delta_{\pi_1(n_{\mathcal{S}})} & \dots & \llbracket m_{\pi_{n_{\mathcal{M}}}(n_{\mathcal{S}})n_{\mathcal{M}}} \rrbracket \parallel \delta_{\pi_{n_{\mathcal{M}}}(n_{\mathcal{S}})} \end{pmatrix}$$

By assumption, because all players behave honestly, receivers can find all shares with the same padding and thus can recover the set of original messages  $\{m_1, \dots, m_{n_{\mathcal{S}}}\}$ . This completes the proof.  $\square$

*Proposition 3 (Security):* Our protocol  $\Pi_{3\text{RMN}}$  UC-realizes the ideal functionality  $\mathcal{F}_{3\text{RMN}}$  in the  $(\mathcal{F}_{\text{PBB}}, \mathcal{F}_{\text{DKg}}, \mathcal{F}_{\text{ZPK}}^{\text{RPT}}, \mathcal{F}_{\text{ZPK}}^{\text{RCS}}, \mathcal{F}_{\text{ZPK}}^{\text{RCD}})$ -hybrid model with respect to an adversary that statically corrupts fewer than  $\frac{n_{\mathcal{M}}}{2}$  mix servers and fewer than  $\frac{n_{\mathcal{R}}}{2}$  receivers, assuming that the DDH assumption in  $\mathbb{G}_q$  holds.

Because our proof is in a hybrid model that assumes the existence of the ideal functionalities used in the protocol description (see Section IV-B), we provide a description of possible protocols to implement the functionalities used by our 3RMN protocol (see Section III). For all ideal functionalities, we assume that the public parameter

$\text{pp} = (\mathbb{G}_q, g, p, q)$  used throughout the protocol is publicly known.

We begin with the idealized primitives for the functionality  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ . Recall that Goldwasser and Lindell suggest a protocol to realize the ideal functionality for PBB (see Lemma 1). Furthermore, our proof heavily relies on transformer ZPK protocols compiled from  $\Sigma$ -protocols into UC-secure ideal functionalities in the common reference string model, as given by Hazay and Nissim [28]. Thus, because the proofs of Lemmas 2, 3, 4, and 5 can be easily obtained from [28], we omit the details. For notational convenience, we use  $\text{EIG.}(Kg, E, D, \text{Agg})$  instead of  $(\text{EIG.Kg}, \dots)$ .

*Lemma 2:* Assume that  $\text{EIG.}(Kg, E, D, \text{Agg})$  is the semantically secure El Gamal TPKE scheme. The Schnorr protocol in [29] securely realizes the ideal functionality  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$  in the  $\mathcal{F}_{\text{PBB}}$  hybrid model under the hardness assumption of the DDH problem in  $\mathbb{G}_q$ .

We proceed to describe two other ZPK protocols to prove correct decryption and correct re-encryption shuffles. To begin with, Chaum and Pedersen presented a  $\Sigma$ -protocol to prove knowledge of the equality of discrete logs; accordingly, [28] allows us to have Lemma 3, ensuring a protocol that UC-realizes the ideal functionality for the relation  $\mathcal{R}_{\text{CD}}$ .

*Lemma 3:* Assume that  $\text{EIG.}(Kg, E, D, \text{Agg})$  is the semantically secure El Gamal TPKE scheme. The Chaum-Pedersen protocol in [50] securely realizes the ideal functionality  $\mathcal{F}_{\text{ZPK}}^{\text{RCD}}$  in the  $\mathcal{F}_{\text{PBB}}$ -hybrid model, assuming that the DDH problem is hard in  $\mathbb{G}_q$ .

The second is the idealization of the re-encryption shuffle ZKP protocol. Similar to Lemma 3, there is a securely realized protocol of the ideal functionality  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$  for the relation  $\mathcal{R}_{\text{CS}}$ . For the same reason as above, we have Lemma 4.

*Lemma 4:* Assume that  $\text{EIG.}(Kg, E, D, \text{Agg})$  is the semantically secure El Gamal TPKE scheme. The protocol [35] securely realizes the ideal functionality  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$  for the correct shuffle relation  $\mathcal{R}_{\text{CS}}$  in the  $\mathcal{F}_{\text{PBB}}$ -hybrid model under the assumption that the El Gamal TPKE scheme is semantically secure.

Finally, we need the lemma below for an idealized El Gamal distributed key generation protocol. Examples using such a UC-secure DKG protocol include Wikström [43].

*Lemma 5:* Assume that the DL problem is hard in  $\mathbb{G}_q$ . Then, the protocol in [33] securely realizes the ideal functionality  $\mathcal{F}_{\text{DKg}}$  in the  $\mathcal{F}_{\text{PBB}}$ -hybrid model with respect to the honest majority of receivers.

*Proving Proposition 3 correct:* For better understanding, we first describe a brief overview of the proof and then provide a formal proof by developing a simulator.

Since all messages communicated among players are encrypted by the El Gamal TPKE scheme, which is semantically secure, the adversary cannot distinguish an encryption of 0 by a simulator from a real encryption by an honest player in the real model. Thus, we can simulate the messages of honest players. Furthermore, all messages from



corrupt players can be extracted from the counterpart ideal functionality by the simulator; thus, the simulator learns all input messages of corrupt senders, all private keys of corrupt receivers, and all shuffle information of corrupt mix servers. As a result, we can simulate the messages of corrupt players. Therefore, it is easy to show that the protocol  $\Pi_{3\text{RMN}}$  UC-realizes the ideal functionality  $\mathcal{F}_{3\text{RMN}}$  by combining all these lemmas.

Now, we are ready to formally prove Proposition 3. Recall that all players are assumed to agree on the publicly known protocol parameters  $\text{pp} = (\mathbb{G}_q, g, p, q)$ .

*Proof:* We define the simulator  $\mathcal{S}$ . We fix the PPT environment  $\mathcal{Z}$  and assume a dummy adversary  $\mathcal{A}$ .  $\mathcal{S}$  runs a copy of  $\mathcal{A}$  and simulates the other players for  $\mathcal{A}$ 's benefit, forwarding messages from  $\mathcal{Z}$  to its simulated  $\mathcal{A}$  and back. We note that  $\mathcal{S}$  has access to the randomness used for every ciphertext of the senders and receivers as it is provided to a proper ZPK functionality (e.g.,  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ ). Similarly,  $\mathcal{S}$  knows the private keys for each receiver, as it generates the keys for the honest receivers and the corrupt ones send the private keys to  $\mathcal{F}_{\text{DKG}}$ .

Let  $I$  be the index set of honest senders,  $J$  be the index set of honest mix servers, and  $K$  be the index set of honest receivers. By assumption,  $|J| \geq \frac{n_m}{2}$  and  $|K| \geq \frac{n_r}{2}$ .

We describe  $\mathcal{S}$  by dividing it into two submodules. Roughly,  $\mathcal{S}$  first extracts information for the corrupt players from the corresponding ideal functionalities and then performs simulated computations for the honest players.

#### Extraction.

- 1)  $\mathcal{S}$  instructs every ZPK functionality to respond to any proof attempt by honest players with  $b_* = 1$ , which indicates that the proof was correct.
- 2) During key generation, for each receiver  $R_k$ , if  $k \in K$ , then  $\mathcal{S}$  generates the private key for  $R_k$ ; otherwise, it recovers the private key for the corrupt receiver using its submission to  $\mathcal{F}_{\text{DKG}}$ .
- 3)  $\mathcal{S}$  runs simulated honest senders  $S_{i \in I}$  honestly, but since they are not connected to the environment  $\mathcal{Z}$ , they cannot receive `Send` commands and so take as input  $m_i = 0$ .  $S_{i \in I}$  submits a list of distinct  $n_m$  El Gamal ciphertexts of 0. If any sender is corrupted,  $\mathcal{S}$  submits plaintext messages to  $\mathcal{F}_{3\text{RMN}}$  as follows: After the adversary has aggregated the corrupted senders' submissions, as it hands them to  $\mathcal{F}_{\text{ZPK}}^{\text{RPT}}$ ,  $\mathcal{S}$  learns all plaintext shares  $m_{ij}$  for all  $i \in [n_s] \setminus I$  and sends them to  $\mathcal{F}_{3\text{RMN}}$ . Then, it sends  $(S_i, \text{Send})$  to  $\mathcal{F}_{3\text{RMN}}$  on behalf of the honest senders.
- 4) When  $\mathcal{S}$  receives  $(M_j : \text{Mix})$  commands for honest mix servers  $M_{j \in J}$ ,  $M_j$  proceeds as if it had received `Mix` commands from  $\mathcal{Z}$ .  $M_{j \in J}$  forms and submits a list by multiplying each ciphertext of the received list by an encryption of 0. If  $M_j$  is corrupt,  $\mathcal{S}$  submits a list to  $\mathcal{F}_{3\text{RMN}}$  as follows: For corrupt mix servers, from their submissions  $\mathcal{F}_{\text{ZPK}}^{\text{RCS}}$ ,  $\mathcal{S}$  learns the permuted plaintext messages by extracting the permutation  $\pi_j (j \in [n_m] \setminus J)$  and the randomness and sends them to  $\mathcal{F}_{3\text{RMN}}$ . Then,

it sends  $(\text{Mix}, M_j)$  to  $\mathcal{F}_{3\text{RMN}}$  on behalf of the honest mix servers.

- 5)  $\mathcal{S}$  simulates honest receivers  $R_{k \in K}$  by making them act as if they had received `RECOVER` commands from  $\mathcal{Z}$ . Since  $\mathcal{S}$  has already extracted the private keys of corrupt receivers in Step 2 and so can decrypt  $\{L'_j\}$ , it recovers a set of plaintexts and sends the set to  $\mathcal{F}_{3\text{RMN}}$ . Then, it submits the `Output` command on behalf of the honest receivers.

We continue to describe the computation activities of the simulator.  $\mathcal{S}$  waits for  $\mathcal{F}_{3\text{RMN}}$  to output its result  $\mathbf{m} = \{m_1, \dots, m_{n_s}\}$ , recovers it, and delays any messages from  $\mathcal{F}_{3\text{RMN}}$  to other players. Then, the simulator deviates from running the honest players and ideal functionalities honestly in the following ways:

#### Computations.

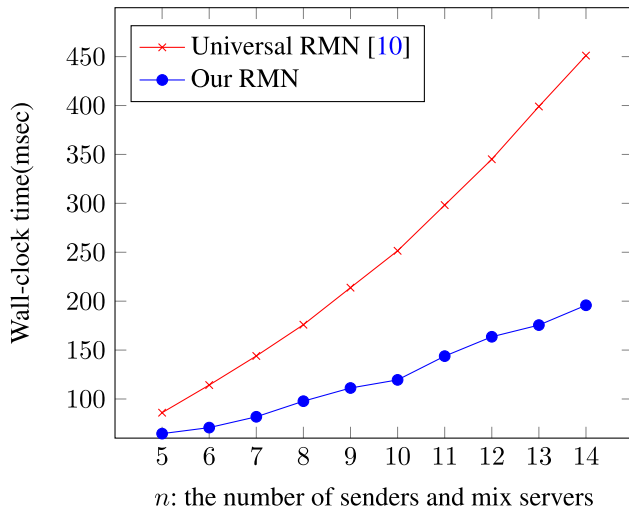
- 6) For honest senders, instead of  $S_{i \in I}$  presenting a list of El Gamal encryptions of additive shares to  $m_i$ , it presents a list of El Gamal encryptions of 0.
- 7) For honest mix servers, instead of  $M_{j \in J}$  presenting a permutation and a re-encryption of El Gamal ciphertexts, it presents an equal-sized list of El Gamal ciphertexts of 0 in lexicographical order.
- 8) For honest receivers  $R_{k \in K}$ , it generates a faked list as follows: For each  $m_{i \in I}$  recovered from  $\mathbf{m}$ , it generates a random padding  $\delta_i^*$  and finds  $m_{ij}$  such that  $m_i = \oplus_j \llbracket m_{ij} \rrbracket$  and  $m_{ij} = \llbracket m_{ij} \rrbracket \parallel \delta_i^*$ ; it combines these messages with those extracted from corrupt receivers into a matrix  $\mathbf{M}^*$ , each column of which is sorted lexicographically. Then, we present  $\mathbf{M}^*$  on behalf of the honest receivers.

It is quite clear that  $\mathcal{S}$  runs in polynomial time with respect to the input size. We need to show that  $\mathcal{Z}$  cannot distinguish with non-negligible probability between the interaction with  $\mathcal{A}$  in the  $(\mathcal{F}_{\text{PBB}}, \mathcal{F}_{\text{DKG}}, \mathcal{F}_{\text{ZPK}}^{\text{RPT}}, \mathcal{F}_{\text{ZPK}}^{\text{RCS}}, \mathcal{F}_{\text{ZPK}}^{\text{RCD}})$ -hybrid execution and the interaction with  $\mathcal{S}$  in the ideal execution where  $\mathcal{S}$  has access to only  $\mathcal{F}_{3\text{RMN}}$ . At this point, the remaining work is to define sequences of hybrid games that each run  $\mathcal{Z}$  and produce the output of  $\mathcal{Z}$ , as well as to prove that there is no PPT algorithm  $\mathcal{Z}$  that can distinguish between each adjacent pair of hybrid games with nonnegligible probability. We omit this part of the proof since completing hybrid games is extremely tedious and not particularly enlightening.  $\square$

## VI. EXPERIMENTAL EVALUATION

The main goal of this section is to empirically validate the efficiency of our proposal. For this purpose, we have implemented it in C++ together with Golle *et al.* protocol [10] (see §I-E). We herein refer to our implementation as “Our RMN” and to our implementation of Golle *et al.*'s protocol as “Universal RMN”, where the box colors reference the corresponding plot colors in Figures 3 and 4.

Our micro-benchmark tests show that our RMN protocol runs approximately  $2.3\times$  faster than our competitor.



**FIGURE 3.** Overall execution times in milli-seconds. Here we excluded the key generation time in counting the total execution. The number of players in sender and mix server types is taken the same as  $n \in \{5, 6, \dots, 14\}$  but with the prefixed number of receivers ( $n_r = 5$  for our protocol and  $n_r = 1$  for universal RMN protocol).

#### A. EXPERIMENTAL SETUP

We used MacBook Pro running macOS Big Sur (11.5.1), equipped with 16GB of 2667MHz DDR4 RAM and 2.6GHz 6-core Intel Core i7, Turbo Boost up to 4.5GHz, with 12MB shared L3 cache. Our code was compiled using Clang-12.0.1 with options `-std = c++17 -fPIC -O3` enabled. Our implementations leverage the OpenSSL library (1.1.1m) [57]. All experiments use a random prime  $p$  of 2048-bit length such that  $p = 2q + 1$  for a prime  $q$ . All network operations are implemented using the Boost Asio library [58], which is a cross platform C++ library for network and low-level I/O programming.

Each of our experiments measures the wall-clock running times for senders, mix servers, and receivers on a single CPU core. We repeated all experiments for 10 trials, and report the average execution timings across the 10 trials for each type of entities, varying the number of participating players. We set the number of players in senders and mix servers entity types to be the same in the range between 5 to 14 but for simplicity of discussion, we fix  $n_r = 5$  for our protocol and  $n_r = 1$  for Universal RMN protocol. In practice, the number of senders will be much greater than the number of mix servers, but it is sufficient for the purpose of examining how much the total execution time is improved. Furthermore, because Universal RMN protocol does not consider verifiability in their mix-net, for a fair comparison, we also exclude the ZPK protocols discussed in Sections II-C3 and III.

#### B. OUR RMN VS. UNIVERSAL RMN

Our first collection of experiments measures the running time for key generation. As mentioned earlier on, the number of receivers in both experiments is fixed by  $n_r = 5$  which takes an average of 4.72msec. This means that we run the key generation for each experiment and use the newly generated system parameters to execute our and Universal RMN proto-

cols. However, we do not consider the key generation time in counting the total execution time.

Our next set of experiments evaluates the execution time as the number of players increases in the range from 5 to 14. When the total number of senders and mix servers is greater than about 30, our testbed suffered from thermal throttling<sup>4</sup> and thus, we set a maximum of senders and mix servers  $n_s = n_m = 14$ .

The plot of our findings is located in Figure 3. We decompose the total execution time into the running time of each type of entities. We present running time plots for respective entity in Figures 4(a), 4(b), and 4(c). The detailed evaluation about their performance will be provided in the next section. Our finding is consistent with our expectation: Our RMN is faster than Universal RMN protocol for improved round complexity. Our RMN and Universal RMN protocol show comparable performance in the execution time until increasing to  $n_s = n_m = 9$  (i.e., the performance difference is less than  $2\times$ ), whereafter our RMN protocol reigns supreme considerably. More concretely, Our RMN protocol runs about  $2\times$  faster than Universal RMN protocol.

#### C. IMPACT OF ROUND EFFICIENCY

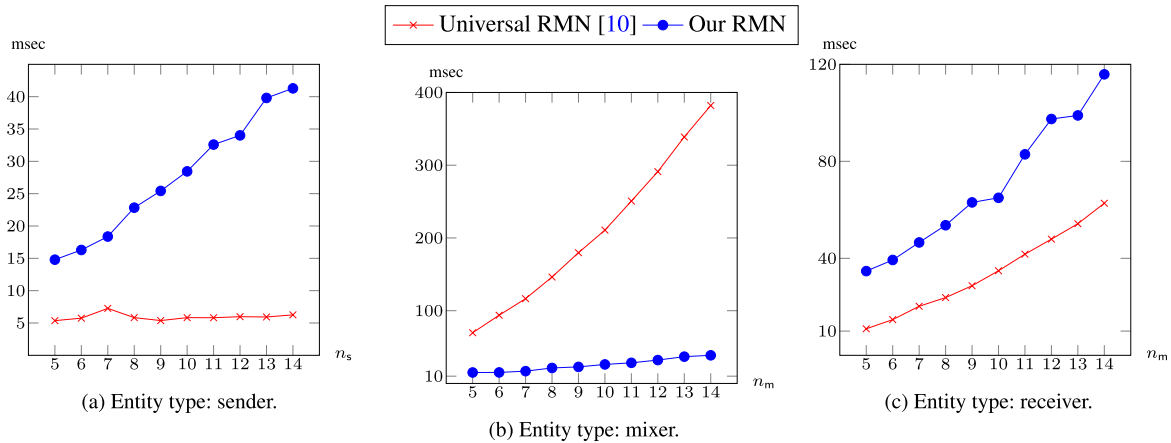
In order to highlight the impact of network delay in Universal RMN protocol which requires the linear round complexity, we now decompose the total execution time into the running times of each type of entities (i.e., sender, mix server, and receiver) and compare to our RMN protocol with respect to each sub-total running time.

##### 1) SENDERS' PERFORMANCE

We graphically compare our RMN and Universal RMN protocols in Figure 4(a). As you will see, our protocol shows a trend that the running time of the senders increasingly grows whereas Universal RMN protocol keeps almost constant in the running time. Specifically, in Universal RMN protocol each sender just generates a pair of ElGamal ciphertexts. Thus increasing the number of senders (and mix servers) does not affect their running time. On the other hand, our protocol requires that each sender generates a list of  $n_m$  ElGamal ciphertexts and thus as the number of mix servers  $n_m$  increases, the running time of senders also increases. It is worthwhile to note that the running time of senders in our protocol may look like to rely on the number of senders, but as mentioned above, in fact, relies on that of mix servers. This is caused by taking the number of senders as the same as that of mix servers.

One thing to give an attention to, in interpreting the plot in Figure 4(a), is that the maximum value of the y-axis is a quite small quantity compared to the total execution time.

<sup>4</sup> We used the app known as Macs fan control (1.5.12) to measure the CPU temperature and during the experiments, used a UNIX command to detect MacBook Pro overheating, `pmset -g thermlog`. Indeed in setting  $n_s = 20 = n_m$ , we observed that the CPU temperature drastically soared to 90 Cellcius degree.



**FIGURE 4.** Comparisons of execution times for each entity type. In (c), our protocol fixes  $n_r$  to 5 while universal RMN protocol sets  $n_r$  to 1.

This means that the running time of senders has little effect on the total running time of both RMN protocols.

In what follows, we compare the performance between our mix servers and our competitor's mix servers.

## 2) MIX SERVERS' PERFORMANCE

This is the heart of our experimental benchmarking test. As shown in Figure 4(b), we can find that the running time of mix servers in both protocol is reversed, viz., the blue line (our plot) is quite below the red line (Universal RMN's plot). We can see from this that regarding the performance of mix servers, the running time of our protocol is much faster than that of Universal RMN protocol. In both protocols, the most expensive computation of mix servers is re-encrypting a list of ElGamal ciphertexts. Mix servers in our protocol only need to locally re-encrypt  $n_s$  ElGamal ciphertexts without any interaction with other mix servers; while in Universal RMN protocol, each of mix servers needs to wait for its all predecessors in a certain order to complete their computation. This is the primary reason that our protocol outperforms Universal RMN protocol.

In particular, the maximum value of the y-axis in Figure 4(b) is much larger than those of Figures 4(b) and 4(c). Therefore, we can conclude that the running time saved by removing the interactions between mix servers is obviously meaningful. In conclusion, although senders and receivers in our protocol run slower than those of Universal RMN protocol, gains by reducing network delay by improving round efficiency make up the losses in their running time.

## 3) RECEIVERS' PERFORMANCE

Receivers in our protocol need to jointly decrypt  $n_m$  times as many ElGamal ciphertexts as Universal RMN protocol. More precisely, receivers in our protocol get  $n_m$  lists of  $n_s$  ciphertexts while those in Universal RMN protocol just get a list of  $n_s$  ElGamal ciphertexts. Recall that in our protocol a message  $m_i$  is split into  $n_m$  shares and then  $n_m$  ElGamal ciphertexts are given to each mix server. This makes the performance gap between ours and Universal RMN protocol, which is depicted in Figure 4(c).

As discussed above, Figure 3 shows that improving round efficiency can compensate for even these additional computation overhead caused by receivers in our protocol.

## VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a UC-secure round-efficient re-encryption mix-net in a three-party setting. Our main tool for achieving UC security is a set of idealized primitives, such as well-studied ZPK toolkits. We applied additive secret sharing to the existing RMN to improve round complexity in an asymptotic sense. Furthermore, we performed a theoretical analysis in terms of computation, communication, and round costs. Finally, we provided a detailed security proof in the UC security model.

**Future work.** In our future work, we will construct and implement a concrete e-voting system based on our 3RMN protocol. Towards this end, we aim to provide a proof-of-concept implementation of our solution in a modern PC environment.

On the theoretical side, we aim to revise our current protocol so that its computation and communication costs are reduced to at least quasi-linear in the number of parties without sacrificing the round efficiency gain. To achieve this goal, we consider two different approaches: one is a batch technique widely used in the design of efficient cryptographic protocols (e.g., [51], [52]). The other is a round compression compiler. Most recently, Ananth *et al.* [53] introduced the notion of efficiency-preserving round compression compilers.

## APPENDIX A OMITTED DEFINITIONS

We formally define zero-knowledge and knowledge extraction by following [54]. We proceed to provide a definition of a  $\Sigma$ -protocol that constitutes a zero-knowledge proof of a special type.

Zero-knowledge proof of knowledge is a proof system  $\langle P, V \rangle$  for a language  $\mathcal{L}$  defined over relation  $\mathcal{R}$  i.e.,  $\mathcal{L} = \{x \mid \exists \omega : (x, \omega) \in \mathcal{R}\}$ , by which a prover  $P$ , knowing witness  $\omega$ , can prove the validity of a statement, i.e.,  $x \in \mathcal{L}$ ,

to a verifier  $V$ . Let  $(P(\omega), V(z, r))(x)$  be the output of  $V$  in interacting with  $P$  on the common public statement  $x$ . The verifier holds the auxiliary input  $z$  and the random tape  $r$ , whereas  $P$  owns the private witness  $\omega$ .

**Definition 6 (Interactive proof system):** A pair of PPT interactive machines  $(P, V)$  is called an interactive proof system for a language  $\mathcal{L}$  if there exists a negligible function  $\text{negl}(\cdot)$  such that the following two conditions hold:

- (Completeness) An honest prover can always convince the honest verifier of a valid statement  $x \in \mathcal{L}$ . Formally, for every  $(x, \omega) \in \mathcal{R}$ ,

$$\Pr[(P(\omega), V)(x) = 1] \geq 1 - \text{negl}(|x|).$$

- (Soundness) A dishonest prover is unable to make a valid proof for an invalid statement  $x \notin \mathcal{L}$  with a high probability. That is, for all  $(x, \omega) \notin \mathcal{R}$  and for all dishonest PPT prover  $P^*$ ,

$$\Pr[(P^*(\omega), V)(x) = 1] \leq \text{negl}(|x|).$$

**Definition 7 (Zero-knowledge):** Let  $(P, V)$  be an interactive proof system for a language  $\mathcal{L}$ . We say that  $(P, V)$  is computational zero-knowledge if for every PPT interactive machine  $V^*$ , there exists a PPT algorithm  $\mathcal{S}$  such that

$$\{(P(\omega), V^*(z, r))(x)\}_{x \in \mathcal{L}} \equiv \{\mathcal{S}(x)\}_{x \in \mathcal{L}},$$

where the left term denotes the output of  $V^*$  after it interacts with  $P$  on the common input  $x$  and the right term denotes the output of  $\mathcal{S}$  on  $x$ .

**Definition 8 (Knowledge extraction):** Let  $\mathcal{R}$  be a binary relation and  $\tau : \mathbb{N} \rightarrow [0, 1]$ . We say that an interactive function  $V$  is a knowledge verifier for the relation  $\mathcal{R}$  with knowledge error  $\tau$  if the following two conditions hold:

- (Nontriviality) There exists an interactive machine  $P$  such that for every  $(x, \omega) \in \mathcal{R}$ , all possible interactions of  $V$  with  $P$  on the common input  $x$  and auxiliary input  $\omega$  are accepted.
- (Validity with error  $\tau$ ) There exists a polynomial  $\phi(\cdot)$  and a probabilistic oracle machine  $\mathcal{M}$  such that for every interactive function  $P$ , every  $x \in \mathcal{L}_{\mathcal{R}}$  and every  $\omega, \gamma \in \{0, 1\}^*$ , every machine  $\mathcal{M}$  satisfies the following condition:

Denote by  $\delta(x, \omega, \gamma)$  the probability that the interactive machine  $V$  accepts on input  $x$  when interacting with the prover specified by  $P_{x, \omega, \gamma}$ . If  $\delta(x, \omega, \gamma) > \mathcal{M}(|x|)$ , then, on input  $x$  and with access to oracle  $P_{x, \omega, \gamma}$ , machine  $\mathcal{M}$  outputs a solution  $w \in \mathcal{R}(x)$  within an expected number of steps bounded by

$$\frac{\phi(|x|)}{\delta(x, \omega, \gamma) - \tau(|x|)}.$$

The oracle machine  $\mathcal{M}$  is called a universal knowledge extractor.

**Definition 9 ( $\Sigma$ -protocol):** A protocol  $\Pi$  is a  $\Sigma$ -protocol for relation  $\mathcal{R}$  if it is a 3-round public-coin protocol and the following requirements hold:

- (Completeness) If  $P$  and  $V$  follow the protocol on input  $x$  and private input  $\omega$  to  $P$ , where  $(x, \omega) \in \mathcal{R}$ , then  $V$  always accepts.
- (Special soundness) There exists a polynomial-time algorithm  $A$  that, given any  $x$  and any pair of accepting transcripts  $(a, e, z)$ ,  $(a, e', z')$  on input  $x$ , where  $e \neq e'$ , outputs  $\omega$  such that  $(x, \omega) \in \mathcal{R}$ .
- (Special honest-verifier zero knowledge) There exists a PPT algorithm  $\mathcal{M}^*$  such that

$$\{(P(x, \omega), V(x, e))\}_{x \in \mathcal{L}_{\mathcal{R}}} \equiv \{\mathcal{M}(x, e)\}_{x \in \mathcal{L}_{\mathcal{R}}},$$

where  $\mathcal{M}(x, e)$  denotes the output of  $\mathcal{M}$  for input  $x$  and  $e$  and  $(P(x, \omega), V(x, e))$  denotes the output transcript of an execution between  $P$  and  $V$ , where  $P$  has input  $(x, \omega)$ ,  $V$  has input  $x$ , and  $V$ 's random tape (determining its query) equals  $e$ .

## ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive and detailed feedback.

## REFERENCES

- [1] M. Jakobsson, A. Juels, and R. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *Proc. USENIX Secur. Symp.*, Aug. 2002, pp. 339–353.
- [2] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *Proc. 19th Int. Conf. Data Eng.*, 2003, pp. 2–15.
- [3] M. Reed, P. Syverson, and D. Goldschlag, "Protocols using anonymous connections: Mobile applications," in *Security Protocols* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1997, pp. 13–23.
- [4] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–88, 1981.
- [5] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Santa Barbara, CA, USA: Springer, 1984, pp. 10–18.
- [6] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 1999, pp. 223–238.
- [7] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proc. 22nd Annu. ACM Symp. Theory Comput. (STOC)*, 1990, pp. 503–513.
- [8] J. Katz, R. Ostrovsky, and A. Smith, "Round efficiency of multi-party computation with a dishonest majority," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, May 2003, pp. 578–595.
- [9] F. Benhamouda and H. Lin, "k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2018, pp. 500–532.
- [10] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal re-encryption for mixnets," in *Topics in Cryptology—CT-RSA* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2004, pp. 163–178.
- [11] M. Abe, "Mix-networks on permutation networks," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1999, pp. 258–273.
- [12] P. Golle and A. Juels, "Parallel mixing," in *Proc. 11th ACM Conf. Comput. Commun. Secur.*, 2004, pp. 220–226.
- [13] R. Cramer, R. Gennaro, and B. Schoenmakers, "A secure and optimally efficient multi-authority election scheme," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1997, pp. 103–118.
- [14] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 136–145.
- [15] C. Park, K. Itoh, and K. Kurosawa, "Efficient anonymous channel and all/nothing election scheme," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1993, pp. 248–259.



- [16] D. Goldschlag, M. Reed, and P. Syverson, "Hiding routing information," in *Information Hiding* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1996, pp. 137–150.
- [17] K. Sampigethaya and R. Poovendran, "A survey on mix networks and their secure applications," *Proc. IEEE*, vol. 94, no. 12, pp. 2142–2181, Dec. 2006.
- [18] T. Haines and J. Müller, "SoK: Techniques for verifiable mix nets," in *Proc. IEEE 33rd Comput. Secur. Found. Symp. (CSF)*, Jun. 2020, pp. 49–64.
- [19] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
- [20] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. Int. Workshop Manag. Requirements Knowl. (MARK)*, Jun. 1979, pp. 313–318.
- [21] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.
- [22] P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting of lotteries," in *Financial Cryptography* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2000, pp. 90–104.
- [23] I. Damgård, "On  $\Sigma$ -protocols," Dept. Comput. Sci., Univ. Aarhus, Aarhus, Denmark, Tech. Rep., 2002.
- [24] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, Jan. 1994, pp. 174–187.
- [25] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1986, pp. 186–194.
- [26] A. De Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai, "Robust non-interactive zero knowledge," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2001, pp. 566–598.
- [27] C. Hazay and Y. Lindell, *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Berlin, Germany: Springer, 2010.
- [28] C. Hazay and K. Nissim, "Efficient set operations in the presence of malicious adversaries," *J. Cryptol.*, vol. 25, no. 3, pp. 383–433, Jul. 2012.
- [29] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, 1991.
- [30] C. Schnorr and M. Jakobsson, "Security of signed ElGamal encryption," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2000, pp. 73–89.
- [31] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2001, pp. 280–299.
- [32] T. Pedersen, "A threshold cryptosystem without a trusted party (extended abstract)," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1991, pp. 522–526.
- [33] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007.
- [34] T. Pedersen, "Distributed provers and verifiable secret sharing based on the discrete logarithm problem," Ph.D. dissertation, Dept. Comput. Sci., Aarhus Univ., Aarhus, Denmark, 1992.
- [35] J. Groth, "A verifiable secret shuffle of homomorphic encryptions," in *Public Key Cryptography*. Berlin, Germany: Springer, Jan. 2003, pp. 145–160.
- [36] C. A. Neff, "A verifiable secret shuffle and its application to e-voting," in *Proc. 8th ACM Conf. Comput. Secur. (CCS)*, 2001, pp. 116–125.
- [37] J. Furukawa and K. Sako, "An efficient scheme for proving a shuffle," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2001, pp. 368–387.
- [38] D. Wikström, "A sender verifiable mix-net and a new proof of a shuffle," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2005, pp. 273–292.
- [39] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2012, pp. 263–280.
- [40] P. Fauzi and H. Lipmaa, "Efficient culpably sound NIZK shuffle argument without random oracles," in *Topics in Cryptology—CT-RSA* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, Feb. 2016, pp. 200–216.
- [41] P. Fauzi, H. Lipmaa, J. Siim, and M. Zajac, "An efficient pairing-based shuffle argument," in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2017, pp. 97–127.
- [42] R. Canetti, "Security and composition of multiparty cryptographic protocols," *J. Cryptol.*, vol. 13, no. 1, pp. 143–202, Jan. 2000.
- [43] D. Wikström, "A universally composable mix-net," in *Proc. Theory Cryptogr. Conf. (TCC)*, 2004, pp. 317–335.
- [44] S. Goldwasser and Y. Lindell, "Secure computation without agreement," in *Distributed Computing* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2002, pp. 17–32.
- [45] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proc. 34th Annu. ACM Symp. Theory Comput. (STOC)*, 2002, pp. 494–503.
- [46] Y. Tsiounis and M. Yung, "On the security of ElGamal based encryption," in *Public Key Cryptography* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1998, pp. 117–134.
- [47] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2005, pp. 467–482.
- [48] D. Dolev, C. Dwork, and M. Naor, "Non-malleable cryptography," in *Proc. 23rd Annu. ACM Symp. Theory Comput. (STOC)*, 1991, pp. 542–552.
- [49] F. Brant, "Efficient cryptographic protocol design based on distributed El Gamal encryption," in *Information Security and Cryptology—(ICISC)* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2005, pp. 32–47.
- [50] D. Chaum and T. Pedersen, "Wallet databases with observers," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1992, pp. 89–105.
- [51] J. Camenisch, S. Hohenberger, and M. O. Pedersen, "Batch verification of short signatures," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2007, pp. 246–263.
- [52] I. Kaslasi, R. D. Rothblum, and P. N. Vasudevan, "Public-coin statistical zero-knowledge batch verification against malicious verifiers," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2021, pp. 219–246.
- [53] P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain, "Towards efficiency-preserving round compression in MPC—Do fewer rounds mean more computation?" in *Advances in Cryptology—ASIACRYPT* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2020, pp. 181–212.
- [54] O. Goldreich, *The Foundations of Cryptography: Volume 1—Basic Tools*. Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [55] A. Kumar and S. K. Gupta, "Synchronization-aware task allocation techniques for preemption control to reduce blocking time in multiprocessor real-time system," *Int. J. Embedded Real-Time Commun. Syst.*, vol. 11, no. 4, pp. 60–79, Oct. 2020.
- [56] A. Kumar and S. K. Gupta, "Reliable energy-aware scheduling algorithm with multi-level budget for real-time embedded system," *Int. J. Embedded Real-Time Commun. Syst.*, vol. 12, no. 4, pp. 55–76, Oct. 2021.
- [57] The OpenSSL Project, *OpenSSL: The Open Source Toolkit for SSL/TLS*. Accessed: Aug. 1, 2022. [Online]. Available: <https://www.openssl.org>
- [58] C. Kohlhoff, *Boost.asio—1.53*. Accessed: Aug. 1, 2022. [Online]. Available: [http://www.boost.org/doc/libs/1\\_53\\_0/doc/html/boost\\_asio.html](http://www.boost.org/doc/libs/1_53_0/doc/html/boost_asio.html)
- [59] N. Borisov, "An analysis of parallel mixing with attacker-controlled inputs," in *Privacy Enhancing Technologies* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2005, pp. 12–25.
- [60] C. Héban, D. H. Phan, and D. Pointcheval, "Linearly-homomorphic signatures and scalable mix-nets," in *Public-Key Cryptography* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2020, pp. 597–627.
- [61] C. Killer, M. Eck, B. Rodrigues, J. Von Der Assen, R. Staubli, and B. Stiller, "ProvoMN: Decentralized, mix-net-based, and receipt-free voting system," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2022, pp. 1–9.



**MYUNGSUN KIM** received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 1994, the M.S. degree in computer science and engineering from Information and Communications University (ICU), Daejeon, in 2002, and the Ph.D. degree in mathematics from Seoul National University (SNU), Seoul, in 2012. He was worked with the Department of Information Security, University of Suwon. He is currently an Assistant Professor with the Department of Mathematics, Gachon University. His research interests include efficient constructions of cryptographic algorithms and their practical applications to real-world solutions.