A New Approach to the Constant-Round Re-encryption Mix-Net

Myungsun Kim

Department of Mathematics, Gachon University
msunkim@gachon.ac.kr

Abstract. The re-encryption mix-net (RMN) is a basic cryptographic tool that is widely used in the privacy protection domain and requires anonymity support; for example, it is used in electronic voting, web browsing, and location systems. To protect information about the relationship between senders and messages, a number of mix servers in RMNs shuffle and forward a list of input ciphertexts in a cascading manner. The output of the last mix server is decrypted to yield the set of original messages. The main downside of this approach is that the mixing process requires a number of rounds that is linear in the number of mix servers. This implies that a long round delay would cause network latency, which can dominate local computational latencies. To minimize the effect of network latency, RMN protocols with constant round complexity are more desirable.
In this work, we propose a new RMN protocol that runs in $O(1)$ rounds in the number of mix servers and that UC-realizes a hybrid model with access to some functionalities for secure communication and zero-knowledge proof (ZKP). Interestingly, because our protocol does not require a ZKP protocol for a verifiable shuffle, we also achieve a considerable efficiency gain in terms of computation cost. Our main tools are secret sharing and an ElGamal encryption that is extended in the sense that it works on a multiplicative group under field extension. Importantly, this extended ElGamal encryption scheme acquires a new capability: it can efficiently decompose a decrypted message into unique values. We provide a detailed report on the theoretical performance and security analysis of this method.

Keywords: Re-encryption mix-net · ElGamal encryption · Round complexity.

1 Introduction

A mix-net is an interactive protocol that consists of senders, mix servers, and receivers, where receivers can be replaced with mix servers (or any other trustees). The senders that hold the initial inputs wish to hide the relationship between their initial inputs and the final output produced by the receivers at the end of execution.

For this purpose, mix servers transform sets of input messages by a proper encryption scheme and subsequently by the permutation of their output order. Then, the transformed messages are sent through a sequence of mix servers. We graphically show the abstract structure of this method in Figure **??**. As long as the permutation of at least one server remains secret, the system prevents linking message senders and messages. There are many types of mix-nets, depending on the transformation that they perform on the ciphertexts and how this transformation is verified.

Loosely speaking, there have been two basic types of mix-net. The first is known as a re-encryption mix-net (RMN). In this type of mix-net, both inputs and outputs are ciphertexts under a homomorphic encryption that allows re-encryption without knowing the corresponding private key (e.g., ElGamal [**ElG85**] and Paillier [**Pai99**]). Because the output is still in ciphertext, some number of receivers that had generated a public and private key pair have to decrypt it to produce the final output. The second type is known as a decryption mix-net, which was originally designed by Chaum [**Cha81**]. The inputs to this mix-net are ciphertexts produced through interactive encryption under the public keys of individual servers. While processing the inputs, each mix server decrypts the layer corresponding to its own public key in each ciphertext and then permutes the resulting ciphertexts. Unlike RMNs, the final output is produced in cleartext.

Regardless of the type, one structural feature of mix-nets is that mix servers run in a sequential order. This leads to a linear round complexity in the number of mix servers. For a small number of mix servers, the network latency caused by a small round delay may not pose a performance problem with respect to the whole execution time of the top layer. However, as the number of mix servers increases, the network latency ultimately dictates the local computation time. Then, the top-priority efficiency measure becomes the round complexity. For this reason, much research effort has focused on devising round-optimal protocols. (e.g., see [**BMR90**; **KOS03**; **BL18**] in the MPC literature).

In this work, we are primarily interested in obtaining RMNs with round complexity that is constant in the number of mix servers while ensuring security against a static adversary that can corrupt only a minority of mix servers. We note that if the mix servers generate the key pair, the distributed key generation algorithm requires an honest majority of mix servers; however, if the receivers generate the key pair, then we can drop the minority condition.

Our goals. To our knowledge, no constructions of constant-round RMN protocols that are secure in the universal composability (UC) model are known. Motivated by this observation, we aim to build an efficient RMN protocol with $O(1)$ round complexity while guaranteeing UC security [**Can01**].

In this work, we present a suitable solution for achieving this goal by introducing secret sharing and extending ElGamal encryption for our purposes.

Main challenges. A naïve solution to this problem is as follows: Let n_{s} be the number of senders and n_{m} the number of mix servers. Each mix server outputs a re-encrypted and permuted list of n_{s} ciphertexts from the senders. We ask the receivers to decrypt the output lists. Obviously, we can achieve $O(1)$ round efficiency; however, the transmission cost for the senders increases by $O(n_{\mathsf{s}}n_{\mathsf{m}})$ ciphertexts, and the computation cost of the mix servers requires $O(n_{\mathsf{s}}n_{\mathsf{m}})$ exponentiations. In particular, the decryption cost on the receivers' side grows significantly, to $O(n_{\mathsf{s}}n_{\mathsf{m}}n_{\mathsf{r}})$ exponentiations for the number of receivers n_{r}.

In the RMN, a permutation known only to each mix server is a means of hiding the relationship between messages and senders, together with re-encryption. Including at least one honest mix server in the mix chain ensures security in that sense. Delinking the mix chain of the RMN leads to the trivial protocol above. Nevertheless, chaining without permutation is likely to be useless.

1.1 Our Ideas and Results

We present a new approach for constructing a constant-round RMN protocol that is secure in the UC model against a static adversary that can corrupt only a minority of mix servers. Using this approach, we are able to achieve both of our aforementioned goals.

Splitting messages into additive shares. As a first step toward obtaining our result for constant round efficiency, the sender S_i finds each of the additive shares of her message m_i, that is, $m_i = \oplus_{j=1}^{n_\mathsf{m}} a_{ij}$, and chooses a proper random value δ_i. Then, m_{ij} is set to $m_{ij} = a_{ij} \parallel \delta_i$ for each j.

Encoding the messages as a polynomial. Define a message polynomial $m_{ij}(z) = z - m_{ij}$ in a proper polynomial ring. Then, to treat $m_{ij}(z)$ as a plaintext message, we extend the ElGamal encryption on $\mathbb{Z}_p^*$ to one that applies to $(\mathbb{F}_p[z]/\langle\theta(z)\rangle)^*$, where p is a prime and $\theta(z)$ is an irreducible polynomial of degree n. We then define the extended ElGamal encryption scheme on a subgroup of $(\mathbb{F}_p[z]/\langle\theta(z)\rangle)^*$. Now, we compute $e_{ij} = \mathsf{E}(m_{ij}(z))$ with this new ElGamal encryption scheme and send the ciphertexts e_{ij} to the mix servers.

Permuting the ciphertexts with homomorphic multiplication. Let $\boldsymbol{e}_j = (e_{1j}, \ldots, e_{n_\mathsf{s}j})$ be a list of ciphertexts received by the mix server M_j. Then, the mix server computes $e_j = \prod_{i=1}^{n_\mathsf{s}} e_{ij}$ under a multiplicative homomorphism. Here, $e_j = \mathsf{E}\left(\prod_{i=1}^{n_\mathsf{s}}(z - m_{ij})\right) = \mathsf{E}\left(\prod_{i=1}^{n_\mathsf{s}}(z - a_{ij} \parallel \delta_i)\right)$.

Permuting plaintexts by decomposing the plaintext polynomials. Let $\boldsymbol{e} = (e_1, \ldots, e_{n_\mathsf{m}})$ be the output of the mixing stage. The receivers, which are assumed to generate a key pair for our variant of the ElGamal encryption, jointly decrypt the list $\boldsymbol{e}$ into a list of plaintext polynomials $(m_1(z), \ldots, m_{n_\mathsf{m}}(z))$. Finally, we factorize each polynomial into irreducible polynomials and reconstruct the initial messages while not revealing the information about the linkage between the senders and messages.

Putting these steps together, we can construct a basic constant-round RMN protocol without considering security. In what follows, we describe in more detail our approach to solving the problem of designing a constant-round RMN protocol.

1.2 Design Rationale behind the New Approach

Our starting point has two main components:

- The first observation provides a basis for the improvement of round efficiency. First, we write an integer m into a vector $(a_1, \ldots, a_{n_\mathsf{m}})$ such that $m = \oplus_{j=1}^{n_\mathsf{m}} a_j$. Then, the vector is modified to $\boldsymbol{m} = (a_1 \parallel \delta, \ldots, a_{n_\mathsf{m}} \parallel \delta)$ for a unique value δ. Even though the components of $\boldsymbol{m}$ are distributed over the mix servers, we can always reconstruct m. Now, imagine that a group of senders distributes their respective messages m_i to mix servers in this way. Then, regardless of the order in which the mix server permutes the input $(a_{1j} \parallel \delta_1, \ldots, a_{n_\mathsf{m}j} \parallel \delta_{n_\mathsf{m}})$, we can still reconstruct every initial message.
- Next, the second observation enables us to reduce the computation overhead by introducing a new method of shuffling based on a known method. Let $m = \prod_{i=1}^{n_\mathsf{s}} m_i$

for each distinct prime $m_i \in \mathbb{Z}$, and let E be a multiplicative homomorphic encryption. Given a homomorphic product of $\mathsf{E}(m_i)$, i.e., $\prod_{i=1}^{n_\mathsf{s}} \mathsf{E}(m_i)$, the prime factorization of $\mathsf{D}(\prod_{i=1}^{n_\mathsf{s}} \mathsf{E}(m_i)) = \mathsf{D}(\mathsf{E}(m))$ is $\{m_1, \ldots, m_{n_\mathsf{s}}\}$ since $\mathbb{Z}$ is a unique factorization domain. The resulting set can be considered the output of the RMN.

More specifically, we now explain the design rationale behind our technique based on the observations.

Concurrent mixing. Our idea for breaking the $O(n_\mathsf{m})$ round barrier is to introduce additive secret sharing, as mentioned in the first observation. Assume that a mix server M_j receives e_{ij} from each sender S_i, where $e_{ij} = \mathsf{E}(a_{ij} \parallel \delta_i)$ and a_{ij} is an additive share of m_i. Then, we build a list $\boldsymbol{e}_j = (e_{1j}, \ldots, e_{n_\mathsf{s}j})$ and form a new list $\boldsymbol{e}'_j = (e'_{1j}, \ldots, e'_{n_\mathsf{s}j})$, where e'_{ij} is a re-encryption of $e_{\pi_j(i)j}$ for a private permutation π_j. Decrypting the output $(\boldsymbol{e}'_1, \ldots, \boldsymbol{e}'_{n_\mathsf{m}})$ from the mix servers, we can correctly recover $\{m_1, \ldots, m_{n_\mathsf{m}}\}$.

However, this approach has a critical security flaw. When the mix servers decrypt $(\boldsymbol{e}'_1, \ldots, \boldsymbol{e}'_{n_\mathsf{m}})$, they can see which message was sent from which sender because the δ_is are unique. Specifically, on receiving e_{ij} from S_i, the servers record (i, e_{ij}) in a table. While performing decryption, they can uniquely mark all additive shares by using δ_i. Not only did we fail to eliminate the need for a proof of correct shuffling, but because each sender needs to send n_m ciphertexts, we have as much computation and communication overhead as in the naïve solution. This is clearly undesirable.

To fix this problem, we need a way to prevent the mix servers from associating e_{ij} from S_i with δ_i. Our solution takes advantage of the multiplicative homomorphism of ElGamal encryption. Concretely, we make the mix servers homomorphically multiply each ciphertext, i.e., compute $e'_j = \prod_{i=1}^{n_\mathsf{s}} e'_{ij}$. Then, the mix servers can see only the decryption of e'_j. To reconstruct the initial input, we have to decompose $\mathsf{D}(e_j)$ into $(a_{1j} \parallel \delta_1, \ldots, a_{n_\mathsf{s}j} \parallel \delta_{n_\mathsf{s}})$; however, we do not know that it is possible to decompose $\mathsf{D}(e_j)$ into irreducible elements in the plaintext space and that such a decomposition can be performed efficiently in a practical sense.

Thus, we need to find a way to efficiently encode $a_{ij} \parallel \delta_i$ into an irreducible element in the plaintext space. To address this issue, we encode $a_{ij} \parallel \delta_i$ into a polynomial $m(z) = (z - a_{ij} \parallel \delta_i) \in \mathbb{F}_p[x]$. However, in general, ElGamal encryption works on a group of congruent integers (e.g., a subgroup of $\mathbb{Z}_p^*$). We thus need to make ElGamal encryption work on a group of polynomials. To do this, we extend ElGamal encryption so that it works over a multiplicative group of extension fields. In particular, with this extension, we can use efficient algorithms to factorize polynomials over finite fields.

A similar observation appears in [**Nef01**]. Neff utilizes the property that a polynomial $h(z) = \prod_{i=1}^{n_\mathsf{s}}(z - m_{\pi(i)})$ is stable for any permutation π of its roots. However, because our main purpose for using polynomials is the efficiency of factorization, there is a primary difference from the proposal of Neff.

Shuffle proof-free mixing. Consider the Groth shuffle proof in [**Gro10**] based on ElGamal encryption. Since this shuffle proof requires each mix server (as the prover) to compute $2n_\mathsf{s}$ single exponentiations and $4n_\mathsf{s}$ multiexponentiations, approximately $6n_\mathsf{s}n_\mathsf{m}$ exponentiations should be needed for only the shuffle prover. This is not a small amount of computation.

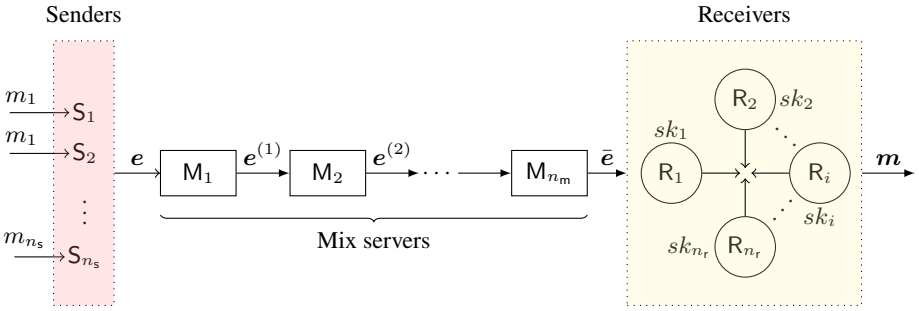

Fig. 1. The classical RMN system model. $\boldsymbol{e}^{(j)}$ is the output of mix server M_j after mixing the list of input ciphertexts $\boldsymbol{e}^{(j-1)}$, where $\boldsymbol{e}^{(0)} = \boldsymbol{e}$ and $\bar{\boldsymbol{e}} = \boldsymbol{e}^{(n_\mathsf{m})}$.

As mentioned previously, we achieve the same effect as permutation by polynomial factorization, and thus, we can save the computational cost of the shuffle proof. However, we need to verify whether the mix servers correctly perform homomorphic multiplication over ciphertexts. The problem is that proofs of the correct multiplication of ciphertexts require the same amount of computation as the shuffle proof in an asymptotic sense. Fortunately, we can circumvent zero-knowledge proofs of correct multiplication by using δ_i. That is, after decrypting the ciphertexts from the mix servers, the sender S_i can check whether her δ_i and every additive share appear correctly on a public board.

1.3 Closely Related Work

The re-encryption mix-net was first developed by Park et al. in [**PIK93**] to address the drawbacks of the decryption mix-net [**Cha81**] and the hybrid mix-net [**GRS96**]. See [**SP06**; **HM20**] 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 $\mathsf{E}_{pk}(m_i, r_i)$, where $\mathsf{E}()$ is the ElGamal public key encryption 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 $\mathsf{E}_{pk}(m_i, r_i + \rho_j)$ on the S_i input, where ρ_j is a randomizer of M_j for the 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 terminates at the mix server $\mathsf{M}_{n_\mathsf{m}}$, where n_m is the number of mix servers. The RMN is graphically illustrated in Fig. **??**; after mixing, the group of receivers may perform a joint decryption stage to output a set of permuted messages.

We 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 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 for sharing the private key is needed. This drawback was addressed

with an improved variant, called the universal re-encryption mix-net, proposed by Golle et al. in [**GJJS04**]. In this RMN, the sender S_i broadcasts two ElGamal 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., $\mathsf{E}_{pk}(m_i, r_i) \parallel \mathsf{E}_{pk}(1, \gamma_i)$. The remaining $n_\mathsf{m} - 1$ 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, the basic RMN protocol is inherently inefficient in its operation. Since Abe observed that unlike the DMN, sequential ordered mixing is not necessary [**Abe99**] in the case of an RMN, Golle and Juels [**GJ04**] utilized this observation in the efficient design of an RMN in which the mix servers perform mixing in parallel. Hence, the authors called this a parallel mix-net. Such a mix-net enjoys a 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 ℓ/n_m inputs, where ℓ is the size of the list. Then, the mix servers perform the following steps:

1. Each mix server mixes a given subset of size ℓ/n_m.
2. The mix servers perform $\tilde{n}_{n_\mathsf{m}}$ rounds of rotations, where $\tilde{n}_{n_\mathsf{m}}$ 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 $\tilde{n}_{n_\mathsf{m}} - 1$ rounds, each mix server retains a random fraction $\frac{1/n_\mathsf{m}}{n_\mathsf{m}}$ of its outputs and sends equal random portions of the remaining outputs to each of the $n_\mathsf{m} - 1$ mix servers. Thus, M_j receives $\frac{\ell}{{n_\mathsf{m}}^2}$ inputs from each of the remaining mix servers, for a total of $\frac{\ell n_\mathsf{m}}{{n_\mathsf{m}}^2} = \frac{\ell}{n_\mathsf{m}}$ 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 $2(\tilde{n}_{n_\mathsf{m}} - 1) + 2 = 2\tilde{n}_{n_\mathsf{m}}$ rounds of mixing. Consequently, the parallel mix-net has $O(\tilde{n}_{n_\mathsf{m}})$ round complexity.

Organization. The rest of the paper is organized as follows: Section **??** presents the notation and definitions used and preliminary background material. Section **??** formally introduces the notion of the ideal mix-net functionality and gives a UC security definition for it. Section **??** describes our main protocol, which has a round cost of $O(1)$ in the number of mix servers while ensuring UC security. Finally, Section **??** contains a performance analysis and proofs of security.

2 Background: Models and Definitions

Before elucidating our main protocol, we introduce the notation and review some definitions and primitives used throughout the paper.

2.1 Notation

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. A null value is denoted by $\perp$. Letting $\boldsymbol{v}$ be an ordered ℓ-tuple (or simply a list) with elements $\boldsymbol{v} = (v_1, v_2, \ldots, v_\ell)$, we use the notation $\boldsymbol{v}[i]$ to index the i-th element v_i. We use bold uppercase letters such as $\boldsymbol{V}$ to denote tables (one may think of them as a database) and sometimes identify a table with its ordered ℓ-tuple. Thus, we mean by $\boldsymbol{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n)$ a table consisting of n ℓ-tuples, and similarly, $\boldsymbol{V}[i, j]$ means the j-th element $\boldsymbol{v}_i[j]$ of the i-th tuple. For any integer a, we denote by $|a|$ the length of a in bits. For a finite set A, we let $a \xleftarrow{\$} A$ be an element that is sampled uniformly at random from A.

We use κ to denote the security parameter, which all protocols and ideal functionalities implicitly take as input, throughout the paper. A function $\nu : \mathbb{N} \to [0, 1]$ is negligible if it tends toward zero faster than $1/n^\kappa$ for every fixed constant κ. We then use $\mathsf{poly}(\kappa)$ and $\mathsf{negl}(\kappa)$ to denote unspecified polynomial and negligible functions in κ, respectively. Let $\mathcal{X} = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$ be ensembles. Two ensembles $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable, denoted by $\equiv$, if for every probabilistic polynomial-time (PPT) algorithm D and for all $\kappa \in \mathbb{N}$, there exists a negligible function negl such that

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

2.2 Models: System, Communication and Adversary

System. We assume that our system model consists of a set of n_s senders $\mathsf{S}_1, \ldots, \mathsf{S}_{n_\mathsf{s}}$, a set of n_m mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_\mathsf{m}}$, and a set of n_r receivers $\mathsf{R}_1, \ldots, \mathsf{R}_{n_\mathsf{r}}$, which are all modeled by PPT Turing machines. Here, some number of receivers will jointly decrypt a list of ciphertexts produced by mix servers. However, parties of the receiver type are not requisite entities; they are employed only for simplicity of representation. Thus, on behalf of the receivers, the mix servers can perform joint decryption after completing a mix stage.

Channel. We assume that the parties have access to a broadcast channel (e.g., by using a public, append-only bulletin board) Furthermore, we assume that the parties are equipped with a synchronized clock; thus, computation is carried out in synchronized rounds, and messages are received by their sink parties within a fixed time bound. For simplicity of discussion, we assume a fully synchronous channel, where for the execution of the protocol, messages of a given round are sent by all parties and in particular are delivered to their intended sink parties.

Adversary. We assume that an adversary $\mathcal{A}$ can corrupt at most t of the n_m mix servers in the system for any access threshold $t < n_\mathsf{m}/2$. Likewise, the adversary cannot corrupt a dishonest majority of receivers. We consider a malicious adversary that can dictate that corrupted parties deviate from the protocol specification in any way. We assume that the adversary is also modeled by a PPT Turing machine. In particular, the adversary is restricted to be static and thus needs to choose the corrupted parties at the beginning of the protocol.

2.3 Hardness Assumptions

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

Assumption 1. *We say that a DDH problem is hard relative to $\mathbb{G}_q$ if for all PPT algorithms $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

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

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 by

Assumption 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* $\mathsf{negl}(\cdot)$ *such that*

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

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

2.4 Cryptographic Tools

Secret sharing. In [**Sha79**], Shamir proposed the first (τ, n)-threshold secret sharing scheme, where the (τ, n)-threshold means that the original secret m is split into n different shares and that with any τ shares, the original secret can be reconstructed, while any $\tau - 1$ shares leak nothing about the secret.

A (τ, n)-threshold secret sharing scheme is a pair of PPT algorithms $(\mathsf{Sh}, \mathsf{Rc})$ such that:

- $(m_1, \ldots, m_n) \leftarrow \mathsf{Sh}(\mathsf{pp}, m, n, \tau)$. Taking as input a publicly known parameter pp, the number of parties n, and an access threshold τ, this distributes the secret m among the n parties and outputs an n-tuple of shares $(m_1, \ldots, m_n)$. Assume that a message space $\mathcal{M}$ is implicitly described in pp.
- $m \leftarrow \mathsf{Rc}(\mathsf{pp}, \{m_i\}_{i\in[\tau]})$. Given a τ-tuple of shares, this outputs a message m satisfying the following correctness condition: for all $m \in \mathcal{M}$ and for any subset $\{i_1, \ldots, i_\tau\} \subseteq [n]$ of size τ,

$$\Pr_{(m_i)_{i\in[n]} \leftarrow \mathsf{Sh}(\mathsf{pp}, m, n, \tau)}[\mathsf{Rc}(\mathsf{pp}, \{m_{i_1} \ldots, m_{i_\tau}\}) = m] = 1$$

We can see that the (τ, τ)-threshold secret sharing technology is suitable for our system model because mixing ciphertexts can be viewed as an outsourced protocol with $\tau \geqslant 2$. Hereafter, we use $\binom{n}{\tau}$-sharing for convenience instead of (τ, n)-threshold secret sharing.

Threshold homomorphic encryption. A public-key encryption (PKE) scheme is a triple of PPT algorithms denoted by $\mathtt{PKE} = (\mathsf{Kg}, \mathsf{E}, \mathsf{D})$:

- $(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$ 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 $\mathcal{M}_{pk}$, a randomness space $\mathcal{R}_{pk}$, and a ciphertext space $\mathcal{C}_{pk}$.
- $e \leftarrow \mathsf{E}_{pk}(m, r)$ takes pk and a plaintext $m \in \mathcal{M}_{pk}$ as input. It outputs a ciphertext $e \in \mathcal{C}_{pk}$. As usual, this process is randomized using a randomizer $r \in \mathcal{R}_{pk}$; however, sometimes we simply write $e \leftarrow \mathsf{E}_{pk}(m)$, omitting the randomness r.
- $m \leftarrow \mathsf{D}_{sk}(e)$ takes sk and $e \in \mathcal{C}_{pk}$ as input. It outputs the plaintext $m \in \mathcal{M}_{pk}$.

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

We say that a PKE scheme is *homomorphic* for the binary relations $(*_\mathsf{h}, \times_\mathsf{h})$ if for all $(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$, $(\mathcal{M}_{pk}, *_\mathsf{h})$ and $(\mathcal{C}_{pk}, \times_\mathsf{h})$ each form a group, and for all $e_1, e_2 \in \mathcal{C}_{pk}$, $\mathsf{D}_{sk}(e_1 \times_\mathsf{h} e_2) = \mathsf{D}_{sk}(e_1) *_\mathsf{h} \mathsf{D}(sk, e_2)$. 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 $\mathsf{Re}(pk, e, r) := e \times_\mathsf{h} \mathsf{E}_{pk}(0, r)$ for the identity $0 \in \mathcal{M}_{pk}$ and $r \in \mathcal{R}_{pk}$.

Because it is undesirable for only a single party to be able to control the decryption process of ciphertexts, a threshold version of a homomorphic PKE scheme needs to be used in a multiparty setting (e.g., [**GM84**; **ElG85**; **Pai99**; **FPS00**]). We use a threshold ElGamal 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.

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

We next formally define the notion of semantic security against chosen plaintext attacks (CPAs) [**GM84**]. To simplify the notation, we use $a \leftarrow A^{\mathcal{O}_1, \mathcal{O}_2, \cdots}(b_1, b_2, \ldots)$ to denote an algorithm A that takes as inputs $b_1, b_2, \ldots$, uses oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots$ in a black-box manner and outputs a. For a PPT adversary $\mathcal{A}$, we define the advantage function

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

where $\mathcal{O}_{\mathsf{Kg}}$ samples $(pk, sk) \leftarrow \mathsf{Kg}(1^\kappa)$ and $b \xleftarrow{\$} \{0,1\}$ and outputs pk; if $|m_0| = |m_1|$, then $\mathcal{O}_{\mathsf{E}}(m_0, m_1)$ returns $e_b \leftarrow \mathsf{E}_{pk}(m_b)$. 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 $\mathtt{adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{PKE}}(\mathcal{A}, \kappa)$ is a negligible function of κ.

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 Σ-protocols, which need only three rounds of interaction [**Dam02**; **CDS94**]. Unfortunately, Σ-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-Shamir heuristic [**FS86**] to make these proofs noninteractive.[1] As a consequence, the obtained proofs are indeed zero-knowledge in the random oracle model and consist of only a single message.[2]

A zero-knowledge proof of knowledge is a proof system $\langle P, V \rangle$ for a language $\mathcal{L}$ defined over a relation $\mathcal{R}$, i.e., $\mathcal{L} = \{\mathsf{x} | \exists \omega : (\mathsf{x}, \omega) \in \mathcal{R}\}$, by which a prover P that knows witness ω can prove the validity of a statement, i.e., $\mathsf{x} \in \mathcal{L}$, to a verifier V. Let $(P(\omega), V(z, r))(\mathsf{x})$ 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 ω.

Definition 1 (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 $\mathsf{negl}(\cdot)$ such that the following two conditions hold:*

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

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

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

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

We formally define zero-knowledge and knowledge extraction by following [**Gold04**]. We next provide a definition of a Σ-protocol that constitutes a zero-knowledge proof of a special type.

---

[1] By a noninteractive proof, we mean that the proof generated by the prover can be verified without further interaction with the prover.

[2] 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 [**SCO+01**] to obtain noninteractivity. In principle, our security proof also works intact in the standard model by utilizing interactive ZKPs.

Definition 2 (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*

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

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

Definition 3 (Knowledge extraction). *Let $\mathcal{R}$ be a binary relation and $\tau : \mathbb{N} \to [0, 1]$. We say that an interactive function V is a knowledge verifier for the relation $\mathcal{R}$ with knowledge error τ if the following two conditions hold:*

- *(Nontriviality) There exists an interactive machine P such that for every $(\mathsf{x}, \omega) \in \mathcal{R}$, all possible interactions of V with P on the common input x and auxiliary input ω are accepted.*
- *(Validity with error τ) There exists a polynomial $\phi(\cdot)$ and a probabilistic oracle machine $\mathcal{M}$ such that for every interactive function P, every $\mathsf{x} \in \mathcal{L}_{\mathcal{R}}$ and every $\omega, \gamma \in \{0, 1\}^*$, every machine $\mathcal{M}$ satisfies the following condition:*
  Denote by $\delta(\mathsf{x}, \omega, \gamma)$ the probability that the interactive machine V accepts on input x when interacting with the prover specified by $P_{\mathsf{x},\omega,\gamma}$. If $\delta(\mathsf{x}, \omega, \gamma) > \mathcal{M}(|\mathsf{x}|)$, then, on input x and with access to oracle $P_{\mathsf{x},\omega,\gamma}$, machine $\mathcal{M}$ outputs a solution $w \in \mathcal{R}(\mathsf{x})$ within an expected number of steps bounded by

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

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

Definition 4 (Σ-protocol). *A protocol Π is a Σ-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 ω to P, where $(\mathsf{x}, \omega) \in \mathcal{R}$, then V always accepts.*
- *(Special soundness) There exists a polynomial-time algorithm $\mathcal{A}$ that, given any x and any pair of accepting transcripts $(a, e, z), (a, e', z')$ on input x, where $e \neq e'$, outputs ω such that $(\mathsf{x}, \omega) \in \mathcal{R}$.*
- *(Special honest-verifier zero knowledge) There exists a PPT algorithm $\mathcal{M}^*$ such that*

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

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

3 Ideal Functionalities

In this section, we formally define an ideal functionality $\mathcal{F}_{\mathsf{MIX}}$ for an RMN and describe other ideal functionalities invoked as submodules. We inherit some of the ideal functionalities specified by Wikström [**Wik04**] with modifications. The differences basically result from the fact that the mix servers in our construction are different from those in Wikström's.

3.1 An Ideal Mix-Net Functionality

Functionality 1 (An ideal mix-net). *Let $n_{\mathsf{s}}, n_{\mathsf{m}}, n_{\mathsf{r}} \in \mathbb{N}$, and let there be a threshold $t \in \mathbb{N}$. The ideal functionality $\mathcal{F}_{\mathsf{MIX}}$ of a mix-net proceeds as follows, running with senders $\mathsf{S}_1, \ldots, \mathsf{S}_{n_{\mathsf{s}}}$, mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_{\mathsf{m}}}$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_{n_{\mathsf{r}}}$, and an ideal adversary $\mathcal{S}$.*

- *Let $\boldsymbol{Q} = (\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{n_{\mathsf{s}}})$. Initialize a table of same-sized tuples $\boldsymbol{Q} \leftarrow \varnothing$ in which each tuple $\boldsymbol{q}_{i \in [n_{\mathsf{s}}]}$ keeps a counter c_i with the setting $c_i \leftarrow 0$. Further initialize a set of index pairs to $\Gamma_{i \in [n_{\mathsf{s}}]} \leftarrow \varnothing$ and index sets to $\Delta \leftarrow \varnothing, \Lambda \leftarrow \varnothing$. Configure a table $\boldsymbol{X} \leftarrow \varnothing$ in which the position of a row is indexed by a counter c, setting $c \leftarrow 0$. Finally, create a set of empty lists $\boldsymbol{L} = (\boldsymbol{l}_1, \ldots, \boldsymbol{l}_{n_{\mathsf{m}}})$.*
- *For each new input message, repeatedly perform the following:*
  - *Upon receiving $(\mathsf{S}_i : \mathtt{Send}, m_{ij})$ from $\mathcal{Z}$, check whether the i-th tuple $\boldsymbol{q}_i = \boldsymbol{Q}[i]$ is full and $(i, j) \in \Gamma_i$. If it is not full and $(i, j) \notin \Gamma_i$, then $c_i \leftarrow c_i + 1, \Gamma_i \leftarrow \Gamma_i \cup \{(i, j)\}$, add $(\mathsf{S}_i : \mathtt{Send}, m_{ij})$ to $\boldsymbol{Q}[i]$ and hand $(\mathcal{S} : \mathsf{S}_i, \mathtt{Set}, (i, c_i))$ to $\mathcal{Z}$. Otherwise, ignore the message.*
  - *Upon receiving $(\mathsf{M}_j : \mathtt{Mix})$ from $\mathcal{Z}$, if for all $i \in [n_{\mathsf{s}}] : c_i \geqslant j$, then set $c \leftarrow c + 1$, insert a tuple $(\mathsf{M}_j : \mathtt{Mix})$ into $\boldsymbol{X}[c]$, and hand $(\mathcal{S} : \mathsf{M}_j, \mathtt{Set}, c)$ to $\mathcal{Z}$.*
  - *Upon receiving $(\mathsf{R}_k : \mathtt{Receive})$ from $\mathcal{Z}$, push a tuple $(\mathsf{R}_k : \mathtt{Receive})$ to $\boldsymbol{X}[c]$, increment c by 1, and hand $(\mathcal{S} : \mathsf{R}_k, \mathtt{Set}, c)$ to $\mathcal{Z}$.*
  - *Upon receiving the message $(\mathcal{S} : \mathtt{Get}, x)$, do the following:*
    - *Parse x into $\langle (i, c_i), c \rangle$.*
    - *Input. Suppose that $c = \perp$ and a record $(\mathsf{S}_i : \mathtt{Send}, m_{ij})$ was found at $\boldsymbol{Q}[i, c_i] = \boldsymbol{q}_i[c_i]$. If $(i, j) \in \Gamma_i$, then single out m_{ij} from the record, choose a random value $\delta_i \in \{0,1\}^{\mathsf{poly}(\kappa)}$, append $(m_{ij} \parallel \delta_i)$ to the list $\boldsymbol{l}_j$, set $\Gamma_i \leftarrow \Gamma_i \backslash \{(i, j)\}$, and hand $(\mathcal{S} : \mathsf{S}_i, \mathtt{Send})$ to $\mathcal{Z}$.*
    - *Mix. Suppose that $(i, c_i) = \perp$ and that $(\mathsf{M}_j : \mathtt{Mix})$ was found at $\boldsymbol{X}[c]$. If $j \notin \Delta$, set $\Delta \leftarrow \Delta \cup \{j\}$ and $|\Delta| \geqslant n_{\mathsf{m}}/2$, sort $\boldsymbol{l}_j$ in lexicographic order to build a new list $\boldsymbol{l}_j^*$, and hand $\langle (\mathcal{S} : \mathsf{M}_j, \mathtt{Forward}), \{\mathsf{M}_j : \mathtt{Forward}, \boldsymbol{l}_j^*\}_{j \in [n_{\mathsf{m}}]} \rangle$ to $\mathcal{Z}$; otherwise, hand $(\mathcal{S} : \mathsf{M}_j, \mathtt{Mix})$ to $\mathcal{Z}$.*
    - *Output. Suppose that $(i, c_i) = \perp$ and that $(\mathsf{R}_k : \mathtt{Receive})$ was found at $\boldsymbol{X}[c]$. Set $\Lambda \leftarrow \Lambda \cup \{k\}$. If $|\Lambda| \geqslant t$, then for each $i \in [n_{\mathsf{s}}], j \in [n_{\mathsf{m}}]$, mark the m_{ij}s so that each m_{ij} has the same suffix value δ_i, and restore m_i from those m_{ij}s. Then, set $\boldsymbol{l}^{\bullet} = \{m_1, \ldots, m_{n_{\mathsf{s}}}\}$ and hand $\langle (\mathcal{S} : \mathsf{R}_k, \mathtt{Recover}, \boldsymbol{l}^{\bullet}), \{(\mathsf{R}_k : \mathtt{Recover}, \boldsymbol{l}^{\bullet})\}_{k \in [n_{\mathsf{r}}]} \rangle$ to $\mathcal{Z}$. Otherwise, hand $(\mathcal{S} : \mathsf{R}_k, \mathtt{Receive})$ to $\mathcal{Z}$.*

3.2 Security Definition

We formally define security via the UC framework [**Can01**]. Similar to the simplified UC framework [**CCL15**], we assume the existence of a default authenticated channel in the real world. This enables the definition of our ideal functionality to be simpler, and it can be removed easily by combining it with an ideal authenticated channel functionality.

Unlike existing work, we consider three types of participants (e.g., see [**HM20**]): senders, mix servers, and receivers. In general, mix-nets consider only two types of participants: the sender type and the mix-server type; however, in the service of mix servers, the step of decrypting a list of shuffled ciphertexts is removed and given to a receiver-type entity. This makes the structure of the mix-net appear simple and clear. We assume a fixed number of parties in the system throughout the paper; i.e., no new party can join after execution begins.

Specifically, each sender, denoted by $\mathsf{S}_{i\in[n_\mathsf{s}]}$ with $n_\mathsf{s} \geqslant 2$, owns an original input message m_i. Let n_m be the number of mix servers, and let $\mathsf{M}_{j\in[n_\mathsf{m}]}$, with $n_\mathsf{m} \geqslant 2$, denote a mix server. Receivers are dedicated servers that jointly decrypt a permuted list of original messages. We should note that a receiver-type party is introduced only for the purpose of notational convenience and not for security reasons.

Let Π_{MIX} be a re-encryption mix-net protocol. Consider an adversarial environment $\mathcal{Z}$. We consider a static corruption model in which there is a fixed set of corrupt parties determined a priori. Informally speaking, it is required that for all PPT adversaries $\mathcal{A}$ that corrupt some subset of the parties and participate in the real execution of the protocol, there exists an ideal adversary $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the view of the environment is the same in both worlds. We describe this model more formally below.

Ideal world. The ideal world is defined by a trusted ideal functionality $\mathcal{F}_{\mathsf{MIX}}$ described in Functionality **??** that interacts with some number of dummy senders $\mathsf{S}_1, \ldots, \mathsf{S}_{n_\mathsf{s}}$, mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_\mathsf{m}}$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_{n_\mathsf{r}}$, and an ideal adversary (the simulator) $\mathcal{S}$ via secure and authenticated channels. The simulator can corrupt a subset of the parties and may fully control them.

Real world. The environment sets the inputs for all parties, including the adversaries, and obtains their outputs in both worlds. However, the environment does not observe any internal interactions. For example, in the ideal world, such interactions take place between the ideal functionality and another entity (a dummy party or the simulator); in the real world, such interactions take place among real parties. Finally, once the execution is over, the environment outputs a bit denoting either the real or ideal world. For ideal functionality $\mathcal{F}$, adversary $\mathcal{A}$, simulator $\mathcal{S}$, environment $\mathcal{Z}$ and a protocol Π, we formally denote the output of $\mathcal{Z}$ by the random variable $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ in the ideal world and $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ in the real world.

Definition 5 (UC-Realizing RMN). *Let $\mathcal{F}_{\mathsf{MIX}}$ be an ideal functionality as described in Functionality* **??***, and let Π_{MIX} be an RMN protocol. We say that Π_{MIX} UC-realizes $\mathcal{F}_{\mathsf{MIX}}$ if for any real-world PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$,*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{MIX}},\mathcal{S},\mathcal{Z}} \equiv \mathsf{REAL}_{\Pi_{\mathsf{MIX}},\mathcal{A},\mathcal{Z}}.$$

Intuitively, for any adversary, there should be a simulator that can simulate its behavior so that no environment can distinguish between these two worlds. Additionally, our definition can capture compositions by considering an $\mathcal{H}$-hybrid model with an ideal functionality $\mathcal{H}$ for the setup.

3.3 Other Ideal Functionalities in a Hybrid Model

In this section, we specify ideal functionalities that capture authenticated point-to-point and broadcast communication links as well as properties of ZKPs. We begin by defining two ideal functionalities for the communication model. We use a slight modification of the protocols defined in [**GL02**; **CK02**; **GL05**; **CSV16**].

Secure communication channel. As a fundamental communication primitive, we first consider a secure point-to-point communication (SC) ideal functionality that delivers messages from a source party to a sink party. We rephrase the ideal functionality given by Canetti [**Can01**].

Functionality 2 (SC). *The ideal SC functionality, denoted by $\mathcal{F}_{\mathsf{SC}}$, running with a source party S, a sink party R and a simulator $\mathcal{S}$, proceeds as follows:*

1. *Upon receiving a message* $(\mathsf{S} : \mathtt{Send}, m, \mathsf{R})$ *from* $\mathcal{Z}$*, hand* $(\mathcal{S} : \mathsf{S}, \mathtt{Send}, \mathsf{R}, |m|)$ *to* $\mathcal{Z}$.
2. *If* $\mathcal{S}$ *gives the appropriate instruction, hand* $(\mathsf{S} : \mathtt{Send}, m, \mathsf{R})$ *to* $\mathcal{Z}$.

Broadcast channel. The second communication primitive is a secure broadcast (BC) functionality that ensures that the same message is delivered to all honest parties that do not abort. This functionality can also be UC-realized by using the method of Goldwasser and Lindell in [**GL02**; **GL05**]. Lemma **??** states that a secure BC can be securely realized, assuming that more than half of the parties are honest.

Functionality 3 (BC). *The ideal BC functionality, denoted by $\mathcal{F}_{\mathsf{BC}}$, running with parties $\mathsf{S}, \mathsf{R}_1 \ldots, \mathsf{R}_{n_\mathsf{r}}$ and a simulator $\mathcal{S}$, proceeds as follows.*

1. *Upon receiving a message* $(\mathsf{S} : \mathtt{Broadcast}, m)$ *from* $\mathcal{Z}$*, hand* $(\mathcal{S} : \mathsf{S}, \mathtt{Broadcast}, |m|, \{\mathsf{R}_k\}_{k \in n_\mathsf{r}})$ *to* $\mathcal{Z}$*, and if* $\mathcal{S}$ *gives the appropriate instruction, then hand* $(\mathsf{S} : \mathtt{Broadcast}, m, \{\mathsf{R}_k\}_{k \in [n_\mathsf{r}]})$ *to* $\mathcal{Z}$.

Lemma 1 ([GL05]). *There exists a protocol Π_{BC} that securely realizes the ideal functionality $\mathcal{F}_{\mathsf{BC}}$ in a setting in which more than half of the parties are honest.*

Distributed key generation (DKG) functionality. We present the ideal functionality for key generation in a distributed manner for the ElGamal TPKE scheme.

Functionality 4 (DKG). *The ideal DKG functionality for the ElGamal TPKE scheme, denoted by $\mathcal{F}_{\mathsf{DKg}}$, running with senders $\mathsf{S}_1, \ldots, \mathsf{S}_{n_\mathsf{s}}$, mix servers $\mathsf{M}_1, \ldots, \mathsf{M}_{n_\mathsf{m}}$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_{n_\mathsf{r}}$, and a simulator $\mathcal{S}$, proceeds as follows.*

1. *Initialize two index sets* $\Lambda = \emptyset$ *and* $\Lambda_{k' \in [n_\mathsf{r}]} = \emptyset$.
2. *For each* $k \in [n_\mathsf{r}]$, *wait for a message* $(\mathsf{R}_k : \mathtt{SKShares}, \alpha_k, \beta_k)$ *such that* $\alpha_k \in \mathbb{Z}_q^*$ *and* $\beta_k = g^{\alpha_k} \in \mathbb{G}_q$, *and set* $\Lambda = \Lambda \cup \{k\}$.
3. *Hand* $(\mathcal{S} : \mathtt{PKShares}, \beta_1, \ldots, \beta_{n_\mathsf{r}})$ *to* $\mathcal{Z}$.
4. *Hand* $\{(\mathsf{S}_i : \mathtt{PKShares}, \beta_1, \ldots, \beta_{n_\mathsf{r}})\}_{i \in [n_\mathsf{s}]}$ *and* $\{(\mathsf{M}_j : \mathtt{PKShares}, \beta_1, \ldots, \beta_{n_\mathsf{r}})\}_{j \in [n_\mathsf{m}]}$ *to* $\mathcal{Z}$. *Then, hand* $\{(\mathsf{R}_k : \mathtt{KeyPair}, \alpha_k, \beta_1, \ldots, \beta_{n_\mathsf{r}})\}_{k \in [n_\mathsf{r}]}$ *to* $\mathcal{Z}$.
5. *If* $(\mathsf{R}_k : \mathtt{Reconst}, \mathsf{R}_{k'})$ *is received from* $\mathcal{Z}$, *set* $\Lambda_{k'} = \Lambda_{k'} \cup \{k\}$. *If* $|\Lambda_{k'}| \geqslant \frac{n_\mathsf{r}}{2}$, *hand* $(\mathcal{S} : \mathtt{Reconsted}, \mathsf{R}_{k'}, \alpha_{k'})$ *and* $\{(\mathsf{R}_k : \mathtt{Reconsted}, \mathsf{R}_{k'}, \alpha_{k'})\}_{k \in [n_\mathsf{r}]}$ *to* $\mathcal{Z}$, *and otherwise, hand* $(\mathcal{S} : \mathsf{R}_k, \mathtt{Reconst}, \mathsf{R}_{k'})$ *to* $\mathcal{Z}$.

For the ElGamal TPKE cryptosystem, Gennaro et al. [**GJKR07**] suggested a secure multiparty protocol for realizing this functionality. For a concrete UC-secure DKG instantiation, see [**Wik04c**]. As usual, we omit the public parameter $\mathsf{pp} = (\mathbb{G}_q, g, p, q)$ during key generation.

Zero-knowledge proof of knowledge (ZPK) 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. [**CLOS02**]. Since this functionality takes as a parameter a relation $\mathcal{R}$, we present the ideal ZPK functionality, followed by three relations for the functionalities used in constructing our RMN protocol.

Functionality 5 (ZPK). *Let* $\mathcal{L}$ *be a language given by a relation* $\mathcal{R}$. *The ideal functionality, denoted by* $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_\mathcal{L}}$, *for a zero-knowledge proof of knowledge of a witness* ω *to a statement* $\mathsf{x} \in \mathcal{L}$, *running with a prover* P *and a verifier* V, *proceeds as follows:*

1. *The functionality holds an empty table.*
2. *On receiving* $(P : \mathtt{Prove}, \mathsf{x}, \omega)$ *from* $\mathcal{Z}$, *store* ω *in the table under the index* (P, x) *and hand* $(\mathcal{S} : P, \mathtt{Prove}, \mathsf{x}, \mathcal{R}(\mathsf{x}, \omega))$ *to* $\mathcal{Z}$. *Discard all further messages from* P.
3. *On receiving* $(V : \mathtt{Verify}, P, \mathsf{x})$, *read* ω *by the tag* (P, x) *(the empty string if no witness is found), and hand* $(\mathcal{S} : V, \mathtt{Accept}, P, \mathsf{x}, \mathcal{R}(\mathsf{x}, \omega))$ *and* $(V : \mathtt{Accept}, P, \mathcal{R}(\mathsf{x}, \omega))$ *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 $\mathcal{R}_{\mathrm{PT}}$ as follows. As mentioned above, well-known examples of idealized ZPK protocols of plaintext messages include [**Sch91**; **TY98**].

Definition 6 (Knowledge of Plaintext). *Define a relation* $\mathcal{R}_{\mathrm{PT}} \subset (\mathbb{G}_q)^4 \times \mathbb{Z}_q^*$ *as* $\langle (g, \beta, u, v), r \rangle \in \mathcal{R}_{\mathrm{PT}}$ *only if* $r = \log_g u$.

We remark that the pair (β, v) appearing in the definition is not explicitly used, but we retain it for compatibility with legacy methods.

The second relation for our purpose is a ZPK protocol for the knowledge of equality of discrete logarithms, where given a ciphertext (u, v), a prover presents a ciphertext (u', v') and a proof that there exists some r such that $\log_u u' = \log_\beta v'/v = r$. The formal definition is given below.

Definition 7 (Knowledge of the Equality of Discrete Logs). *Define a relation* $\mathcal{R}_{\mathrm{EDL}} \subset (\mathbb{G}_q)^6 \times \mathbb{Z}_q^*$ *as* $\langle (g, \beta, (u, v), (u', v')), r \rangle \in \mathcal{R}_{\mathrm{EDL}}$ *only if* $u' = u^r \wedge v' = v\beta^r$.

One can view the relation $\mathcal{R}_{\mathrm{EDL}}$ as an ideal correspondence to the Σ protocol shown by Cramer et al. [**CDN01**]. As a simple variant, we can consider a ZPK of correct decryption by which a receiver can prove the knowledge of the secret key share α_i. Specifically, we define a relation $\mathcal{R}_{\mathrm{CD}} \subset (\mathbb{G}_q)^5 \times \mathbb{Z}_q^*$ as $\langle (g, \beta_i, (u, v), d_i), \alpha_i \rangle \in \mathcal{R}_{\mathrm{CD}}$ only if $\log_g \beta_i = \log_u d_i = \alpha_i$. In our protocol, the receiver R_i holds $e = (u, v), d_i, \alpha_i$ is such that $d_i = u^{\alpha_i}$, and β_i is a public component of pk.

In describing our main protocol, we will use $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$ to denote an ideal primitive of a ZPK protocol for the relation $\mathcal{R}_{\mathrm{PT}}$, i.e., a zero-knowledge proof of knowledge of the plaintext. With a similar purpose, we will use the additional notation $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$ in a later section.

4 Cryptographic Tools and Extensions

Secret sharing and multiplicative homomorphic encryption (MHE) are two main cryptographic primitives for solving the performance problem in RMNs. We concretely instantiate these primitives and provide some further discussion related to our MHE variant.

4.1 Secret Sharing

As a popular variant method, Blakley in [**Bla79**] constructed additive secret sharing, which allows a given secret m to be decomposed into the sum of τ random numbers. In this work, we use an additive $\binom{\tau}{\tau}$-sharing scheme as follows:

- $(r_1, \ldots, r_{\tau-1}, m_\tau) \leftarrow \mathsf{Sh}(m, \tau, \tau)$. To share a value m, a party, as the dealer, chooses $\{r_k\}_{k\in[\tau-1]} \xleftarrow{\$} (\{0,1\}^{\mathsf{poly}(\kappa)})^{\tau-1}$, sets $m_\tau = m \oplus \bigoplus_{k=1}^{\tau-1} r_k$, and outputs $(r_1, \ldots, r_{\tau-1}, m_\tau)$.
- $m \leftarrow \mathsf{Rc}(\{m_k\}_{k\in[\tau]})$. This takes as input $(m_1, \ldots, m_\tau) \in (\{0,1\}^{\mathsf{poly}(\kappa)})^\tau$ and outputs $\bigoplus_{k=1}^{\tau} m_i$.

This approach is much more efficient for computation than Shamir's solution, and in particular, since our construction does not require a series of multiplications on the shares, a $\binom{\tau}{\tau}$-sharing scheme is quite suitable for our purpose.

4.2 ElGamal Variant with Fast Decomposition

In addition to secret sharing, our protocol makes use of several cryptographic tools, such as ZKP protocols. A primary tool is ElGamal encryption [**ElG85**], and our RMN protocol takes advantage of several nice properties, including distributed key generation, re-encryption, and multiplicative homomorphisms. However, it is not sufficient to solve the problem at hand with only these capabilities. Thus, we need to upgrade ElGamal encryption so that it acquires an extra property, called *decomposition*.

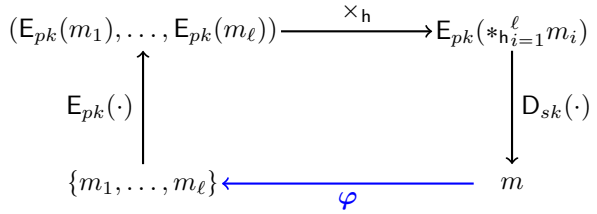

Fig. 2. An abstract relationship between ElGamal processes for our main protocol. Here, we denote by φ a deterministic algorithm Dcom that is added to ElGamal encryption, and $m = *_{\mathsf{h}} {}_{i=1}^{\ell} m_i$.

In what follows, we develop an algorithm that takes as an input a homomorphic product of ℓ ElGamal ciphertexts and outputs a set of original plaintexts by decrypting the product. We call this a decomposition algorithm and denote it as Dcom. Intuitively, the decomposition algorithm aims to restore every original plaintext by factorizing the value being decrypted from a resulting ciphertext by homomorphically multiplying two or more ElGamal ciphertexts. We wish to extend the ElGamal encryption scheme to obtain this efficient decomposition algorithm. In this manner, the extended ElGamal encryption scheme provides the advantages shown in Figure **??**. Then, the resulting extended ElGamal variant has two key properties. Letting $e_1, \ldots, e_\ell$ be ElGamal ciphertexts, the first property is that homomorphic multiplication $\times_{\mathsf{h}} {}_{i=1}^{\ell} e_i$ is inherited from the original ElGamal encryption scheme. More importantly, given a decrypted message $m = *_{\mathsf{h}} {}_{i=1}^{n} m_i$, one can identify from m all original values $m_1, m_2, \ldots, m_\ell$ uniquely up to rearrangement of those values.

A similar technique was given in [**HKK+11**; **KKC13**] with different motivations, such as compressing ciphertexts and set operations, respectively.

The description. The main idea for obtaining an efficient decomposition is to switch a working DDH group from $\mathbb{Z}_p^*$ to one of $\mathbb{F}_{p^n}^*$, where p is a prime. In principle, a decomposition algorithm can be built in the context of the standard ElGamal encryption in the sense of operating in $\mathbb{Z}_p^*$. That is, we encode a plaintext message into an irreducible element in the DDH group, treating it as $\mathbb{Z}$. Then, given a ciphertext through a homomorphic product of ciphertexts whose plaintexts were encoded in this way, after decrypting the ciphertext, one needs to factorize a product of irreducible elements. However, integer factorization may run inefficiently; moreover, a large amount of space is needed to keep the dictionary that may be needed for decoding. On the other hand, because an ElGamal variant in the setting of the multiplicative group $\mathbb{F}_{p^n}^*$ encodes a message m_i into a message polynomial $m_i(z) = (z - m_i) \in \mathbb{F}_p[z]$, one can factorize $m(z) = \prod_{i=1}^{\ell} m_i(z) \in \mathbb{F}_p[z]$ into $\{m_i(z)\}_{i \in [\ell]}$ much more efficiently. More precisely, one can decompose a monic square-free univariate polynomial $f \in \mathbb{F}_p[z]$ of degree ℓ within time $\tilde{O}(\ell^2 \log p \log^{-1} \varepsilon)$ with a small failure probability ε, where $f = \tilde{O}(\ell)$ means that $f = O(\ell(\log \ell)^\eta)$ for constant η [**GG13**]; see also the survey [**GP01**; **HL22**] for a summary of this field.

We are ready to provide a detailed description of the ElGamal variant that allows efficient decomposition.

For reasons of space, we do not offer a syntax for this class of ElGamal variants. Instead, because our main protocol requires distributing a secret key among n_{r} receivers, we describe the ElGamal encryption as operating in $\mathbb{F}^*_{p^n}$,[3] following the syntax of TPKE given in Section **??**. ElGamal encryption in the setting of a cyclic subgroup $\mathbb{G}_q$ of $(\mathbb{F}_p[z]/\langle\theta(z)\rangle)^*$ consists of six PPT algorithms as follows:

Construction 1 (ElGamal in extension fields). *Let κ be the security parameter, and let $\bar{g}(z)$ be a generator of the multiplicative group $(\mathbb{F}_p[z]/\langle\theta(z)\rangle)^*$, where $\theta(z)$ is an irreducible polynomial of degree n.*

- $\mathsf{pp} \leftarrow \mathtt{ElG.Setup}(1^\kappa)$. *This constructs a DDH group $\mathbb{G}_q$ of order q by iteratively using $g(z) = \bar{g}(z)^{\frac{p^n-1}{q}}$, where $p^n - 1 = sq$ for another prime q and an even s, and outputs a publicly known group and protocol parameters as $\mathsf{pp} = (\mathbb{G}_q, g(z), \theta(z), p, q)$.*
- $(pk, sk) \leftarrow \mathtt{ElG.TKg}(\mathsf{pp}, 1^\kappa, n_{\mathsf{r}})$: *This takes as input the public parameter $\mathsf{pp} = (\mathbb{G}_q, g(z), \theta(z), p, q)$, a security parameter κ and the number of receivers n_{r} as input. Each party outputs $\beta(z) = g(z)^{\alpha_i}$ for a random $\alpha_i \xleftarrow{\$} \mathbb{Z}_q$ and sets sk_i to $sk_i = \alpha_i$. It outputs the public key $pk = (\mathsf{pp}, \beta(z))$ and the secret key $sk = \alpha$, where $\beta(z) = \prod_{i=1}^{n_{\mathsf{r}}} \beta_i(z)$ and $\alpha = \sum_{i=1}^{n_{\mathsf{r}}} \alpha_i$.*
- $e \leftarrow \mathtt{ElG.E}_{pk}(m, r)$. *This takes as input a plaintext message $m \in \mathbb{F}_p$ and randomness $r \xleftarrow{\$} \mathbb{Z}_q$, writes m to $m(z) = (z - m) \in \mathbb{F}_p[z]$,[4] and outputs the ciphertext message $e = (g(z)^r, m(z)\beta(z)^r)$.*
- $u_i(z) \leftarrow \mathtt{ElG.TD}_{pk}(sk_i, e)$. *Given a ciphertext $e = (u(z), v(z))$, a party R_i publishes her decryption share $u_i(z) = u(z)^{\alpha_i}$.*
- $m \leftarrow \mathtt{ElG.TAg}_{pk}(v(z), \{u_i(z)\}_{i\in[n_{\mathsf{r}}]})$. *This computes the plaintext $m(z) = \frac{v(z)}{\prod_{i\in[n_{\mathsf{r}}]} u_i(z)} = \frac{m(z)\beta(z)^r}{g(z)^{r\cdot\sum_{i\in[n_{\mathsf{r}}]}\alpha_i}}$ and outputs the message m.*
- $\{m_1, \ldots, m_\ell\} \leftarrow \mathtt{ElG.Dcom}_{pk}(m(z))$. *This takes as input a message polynomial $m(z) \in \mathbb{F}_p[z]$ of degree ℓ, decomposes it into a set of monic irreducible polynomials $\{m_i(z)\}_{i\in[\ell]} \in (\mathbb{G}_q)^\ell$, and outputs a set of plaintext messages $\{m_i\}_{i\in[\ell]} \in (\mathbb{F}_p)^\ell$.*

Proposition **??** ensures that the ElGamal encryption scheme in the setting of a finite cyclic multiplicative subgroup $\mathbb{G}_q$ of $(\mathbb{F}_p[z]/\langle\theta(z)\rangle)^*$ is secure against the IND-CPA attack. As discussed in [**Dam91**; **MOV96**], it is easy to prove the security of this ElGamal variant by following one of the ElGamal encryption schemes working in $\mathbb{Z}^*_p$. All the above computations are performed modulo $\theta(z)$, which we have omitted for brevity.

Proposition 1. *The ElGamal encryption scheme in Construction* **??** *is semantically secure, provided that the DDH assumption holds in $\mathbb{G}_q$.*

More information regarding the distributed key generation (DKG). Despite the simplicity of the DKG procedure above, the complete process is quite involved. The first

[3] In fact, Menezes et al. [**MOV96**] provided a generalized ElGamal encryption scheme that implies our description. For completeness, we provide a full-fledged description.

[4] At this point, it is not clear that $m \in \mathbb{F}_p$; we will justify this later.

design was proposed by Pedersen in [**Ped91**]; however, Gennaro et al. [**GJKR07**] later observed a flaw in the distribution of generated keys and suggested a new DKG construction for DL-based cryptosystems, including ElGamal encryption. Because our variant is the same as conventional ElGamal encryption except that polynomial arithmetic is used in $\mathbb{F}_p[z]$ rather than congruent integer arithmetic, we can directly apply Gennaro et al.'s DKG protocol to the ElGamal variant in extension fields. For this reason, we will use it without further specification, while hiding the details of the DKG in the ElGamal variant. We remark that as in [**GJKR07**], the security of this method holds only for a static adversary that can corrupt at most t parties for $t < n_r/2$.

Compatibility with legacy ZKPs. The second issue that arises because of our modifications is whether the ElGamal variant allows us to reuse existing ZKP protocols, including proof of plaintext knowledge and proof of correct decryption. As mentioned above, since the base is a polynomial $g(z)$ but the exponents are still in $\mathbb{Z}_q^*$, in principle, our modifications do not need to alter ElGamal-related ZKP protocols.

5 Our RMN Protocol Design

Thus far, we have described building blocks to develop a constant-round RMN protocol. In this section, we present the main protocol based on these building blocks. The organization of this section is as follows. First, we clarify the communication model and review some functionalities for ZPK. Next, we provide the details of our RMN protocol and discuss a variant of our protocol. In the next section, we aim to show that our construction UC-realizes the ideal mix-net functionality in a hybrid model.

We assume that the protocol accesses two secure communication primitives as the ideal functionalities specified in Sections **??**. Furthermore, we need to provide our protocol with the following functionalities for ZPK, which output 1 if the proofs are verified and output 0 otherwise. The reason we restate the functionalities is to maintain consistency with the newly developed ElGamal variant.

- ElGamal plaintext knowledge ($\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$). This writes a ciphertext with $e(z) = (u(z), v(z))$. For input $(g(z), \beta(z), e(z))$ from the verifier, the prover inputs r such that $u(z) = g(z)^r$.
- Correct decryption of an ElGamal ciphertext ($\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$). For an input $(g(z), \beta_i(z), e(z), d_i(z))$ from the verifier, where $e(z)$ is the same as above, the prover inputs α_i such that $\beta_i(z) = g(z)^{\alpha_i}$ and $d_i(z) = u(z)^{\alpha_i}$.
- Secure ElGamal initialization ($\mathcal{F}_{\mathsf{DKg}}$). For an input $(\alpha_1, \ldots, \alpha_{n_r})$, a secret sharing of a random, uniformly distributed value $\alpha \in \mathbb{Z}_q^*$ is generated such that $\alpha = \sum_{k \in [n_r]} \alpha_i$, and the value $\beta(z) = g(z)^\alpha$ is publicly open. It is clear that $\beta(z)$ is therefore uniformly distributed in $\mathbb{G}_q = \langle g(z) \rangle$.

5.1 Protocol Details

We are now ready to present the details of our RMN protocol. For simplicity, we assume that the publicly known group description and protocol parameters $\mathsf{pp} = (\mathbb{G}_q, g(z), \theta(z), p, q)$ are given to all parties, and hence we omit the setup algorithm. In

addition, let $\mathbb{H}$ be the hybrid model consisting of functionalities $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\mathsf{BC}}$, $\mathcal{F}_{\mathsf{DKg}}$, $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$, and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$.

Construction 2 (Our RMN Protocol). *The RMN protocol Π_{MIX} is composed of senders $\mathsf{S}_{i\in[n_\mathsf{s}]}$, mix servers $\mathsf{M}_{j\in[n_\mathsf{m}]}$, and receivers $\mathsf{R}_{k\in[n_\mathsf{r}]}$ and runs in the $\mathbb{H}$-hybrid model.*

- *Inputs: Each message m_i from owner S_i*
- *Outputs: At the end of execution, any set of t or more receivers outputs a set of messages $\{m_1, \ldots, m_{n_\mathsf{s}}\}$*
- *The protocol:*
  1. *Key generation. Each receiver R_k does the following:*
     (a) *Choose $\alpha_k \xleftarrow{\$} \mathbb{Z}_q^*$ and hand $(\mathsf{R}_k : \mathtt{SKShares}, \alpha_k, g(z)^{\alpha_k})$ to $\mathcal{Z}$.*
     (b) *Wait for a message $(\mathsf{R}_k : \mathtt{KeyPair}, \alpha_k, \beta_1(z), \ldots, \beta_{n_\mathsf{r}}(z))$ from $\mathcal{Z}$ and set $\beta(z) = \prod_{k\in[n_\mathsf{r}]} \beta_k(z) = g(z)^{\sum_k \alpha_k}$.*
  2. *Message input. Each sender S_i proceeds as follows:*
     (a) *Wait for a message $(\mathsf{S}_i : \mathtt{PKShares}, \beta_1(z), \ldots, \beta_{n_\mathsf{r}}(z))$ from $\mathcal{F}_{\mathsf{DKg}}$ and set $\beta(z) = \prod_{k\in[n_\mathsf{r}]} \beta_k(z) = g(z)^{\sum_k \alpha_k}$.*
     (b) *Wait for a message $(\mathsf{S}_i : \mathtt{Send}_{ij})$ for $j \in [n_\mathsf{m}]$ from $\mathcal{Z}$. Then, choose a random value $\delta_i \in \{0,1\}^{\mathsf{poly}(\kappa)}$ and a randomizer $r_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$. By running $(m_{i1}, \ldots, m_{in_\mathsf{m}}) \leftarrow \mathsf{Sh}(m_i, n_\mathsf{m}, n_\mathsf{m})$, compute $e_{ij} \leftarrow \mathtt{ElG}.\mathsf{E}_{pk}(m_{ij} \parallel \delta_i, r_{ij})$.*
     (c) *Hand $(\mathsf{S}_i : \mathtt{Prove}, \langle (g(z), \beta(z), e_{ij}), r_{ij}\rangle)$ to $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$.*
     (d) *Hand $(\mathsf{S}_i : \mathtt{Send}, e_{ij}, \mathsf{M}_j)$ to $\mathcal{F}_{\mathsf{SC}}$.*
  3. *Ciphertext mixing. Each mix server M_j proceeds as follows:*
     (a) *Wait for a message $(\mathsf{S}_i : \mathtt{PKShares}, \beta_1(z), \ldots, \beta_{n_\mathsf{r}}(z))$ from $\mathcal{F}_{\mathsf{DKg}}$ and set $\beta(z) = \prod_{k\in[n_\mathsf{r}]} \beta_k(z) = g(z)^{\sum_k \alpha_k}$.*
     (b) *Wait until $(\mathsf{S}_i : \mathtt{Send}, e_{ij}, \mathsf{M}_j)$ has been received from $\mathcal{F}_{\mathsf{SC}}$ for all senders; then, do the following:*
        i. *Hand $(\mathsf{M}_j : \mathtt{Verify}, \mathsf{S}_i, (g(z), \beta(z), e_{ij}))$ to $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$.*
        ii. *Wait for $(\mathsf{M}_j : \mathtt{Accept}, \mathsf{S}_i, b_{j_1})$ from $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$. Then, if $b_{j_1} = 0$, abort.*
     (c) *Wait for a message $(\mathsf{M}_j : \mathtt{Mix})$ from $\mathcal{Z}$; then, do the following:*
        i. *Compute $e'_j = \times_{\mathsf{h}_{i=1}}^{n_\mathsf{s}} e_{ij}$, choose $\gamma_j \xleftarrow{\$} \mathbb{Z}_q^*$, and compute $e_j \leftarrow \mathtt{ElG}.\mathsf{Re}_{pk}(e'_j, \gamma_j)$.*
        ii. *Hand $(\mathsf{M}_j : \mathtt{Prove}, \langle (g(z), \beta(z), e_j, e'_j), \gamma_j\rangle)$ to $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$.*
        iii. *Hand $(\mathsf{M}_j : \mathtt{Broadcast}, (e'_j, e_j), \{\mathsf{R}_k\}_{k\in[n_\mathsf{r}]})$ to $\mathcal{F}_{\mathsf{BC}}$.*
  4. *Message output. Each receiver R_k performs the following:*
     (a) *Wait for a message $(\mathsf{M}_j : \mathtt{Broadcast}, e_j, \{\mathsf{R}_k\}_{k\in[n_\mathsf{r}]})$ from $\mathcal{Z}$; then, do the following:*
        i. *Hand $(\mathsf{R}_k : \mathtt{Verify}, (g(z), \beta(z), e_j, e'_j))$ to $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$.*
        ii. *Wait for a message $(\mathsf{R}_k : \mathtt{Accept}, \mathsf{M}_j, b_{k_2})$ from $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$. If $b_{k_2} = 0$, abort.*
     (b) *Wait for a message $(\mathsf{R}_k : \mathtt{Recover})$ from $\mathcal{Z}$, and if more than t distinct $\mathtt{Recover}$ messages are received, do the following:*
        i. *Parse e_j into $(u_j(z), v_j(z))$ for all $j \in [n_\mathsf{m}]$.*

ii. *For each j, compute $u_j^\bullet(z) \leftarrow \mathtt{ElG}.\mathsf{TD}(sk_i, e_j)$ and set $v_j^\bullet(z) = v_j(z)$.*
iii. *Hand* $(\mathsf{R}_k : \mathtt{Prove}, \langle (g(z), \beta_i(z), (u_i^\bullet(z), v_j^\bullet(z))), \alpha_i \rangle)$ *to* $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$.
iv. *Hand* $(\mathsf{R}_k : \mathtt{Broadcast}, u_j^\bullet(z), \{\mathsf{R}_{k'}\}_{k' \in [n_\mathsf{r}] \setminus \{k\}})$ *to* $\mathcal{F}_{\mathsf{BC}}$.
v. *Wait until at least t different $u_j^\bullet(z)$ are received from $\mathcal{Z}$; then, compute $m_j(z) \leftarrow \mathtt{ElG}.\mathsf{TAg}_{pk}(\{u_j^\bullet(z)\})$ and $\{m_{1j} \parallel \delta_1, \ldots, m_{n_\mathsf{s} j} \parallel \delta_{n_\mathsf{s}}\} \leftarrow \mathtt{ElG}.\mathsf{Dcom}(m_j(z))$. Then, for each $j \in [n_\mathsf{m}]$, calculate a set of plaintexts $\boldsymbol{m} = \{m_1 \parallel \delta_1, \ldots, m_{n_\mathsf{s}} \parallel \delta_{n_\mathsf{s}}\}$,[5] identifying every n_s-tuple whose elements have the same δ_i and applying $m_i \leftarrow \mathsf{Rc}(\{m_{ij}\})$.*
vi. *Output* $(\mathsf{R}_k : \mathtt{Recover}, \boldsymbol{m})$

5.2 Discussion

We have described the RMN protocol in a clear way to prove its security. Before reporting the details of the theoretical analysis, we would like to give some further discussions regarding our protocol. These include the following:

- Generation of δ_i. One problem that the value δ_i may pose is that two different senders could choose the same δ_i. One possible way to reduce the chance of this is to use a keyed pseudorandom function (PRF) parameterized by the message m_i and a secret key known only to each S_i. For example, consider AES in counter mode to instantiate such a keyed PRF.
  More importantly, δ_i enables us to have $m_{ij}(z) = (z - m_{ij} \parallel \delta_i) \in \mathbb{F}_p[z]$ by repeatedly checking whether $m_{ij}(z)^q = 1 \mod \theta(z)$. Since we can choose a small number s such that $p^n - 1 = sq$ by the Bateman-Horn conjecture [**BH62**], we can efficiently find such a δ_i.
- The length of δ_i. Essentially, the length of δ_i should be sufficiently large compared with $|n_\mathsf{s}|$ to avoid a collision. For the sake of performance, one may take a relatively small prime p, but this increases the probability of collisions. We should therefore take care in choosing the protocol parameters n, p, q. For the details of the parameter choice, see Section **??**.
- Usefulness of δ_i. The value δ_i can be used to prevent malicious mix servers from manipulating some ciphertexts. Assuming at least one honest mix server, malicious servers cannot make n_m distinct messages for the same δ appear at every mix server.

6 Analysis

Our RMN protocol is designed to produce a set of permuted messages across the mix servers while maintaining strong provable security. As we will show, our protocol is secure against an active adversary that can control a minority of dishonest mix servers. This section shows that our protocol satisfies the security requirement. We begin by studying the asymptotic efficiency of our protocol, mainly focusing on the round complexity.

[5] We should note that $\boldsymbol{m}$ is a set, and thus there is no information about the order of the messages.

6.1 Parameters

In our RMN protocol, ElGamal encryption is the fundamental primitive for security; however, for our purpose, we make it operate on a multiplicative group of $\mathbb{F}_{p^n}$. Therefore, we need to generate the parameters so that our ElGamal variant is secure to several known attacks, including index calculus methods. In taking two primes p, q such that $p^n - 1 = sq$ for some small even s, we should consider the following. First, if p is a relatively small prime, the function field sieve [**Adl94**] on a finite field of small characteristic can efficiently extract discrete logs. Thus, we have to take a prime p such that $\mathbb{F}_p$ has a medium characteristic. For such a prime p, we then take a large prime q such that the size of the DDH group $\mathbb{G}_q$ ensures a proper security level (e.g., a 3072-bit prime for 128-bit security).

6.2 Asymptotic Costs

We quantify the costs for computation and transmission by counting the number of exponentiations used in the whole execution and the total number of $\mathbb{G}_q$ elements exchanged between parties, respectively. We exclude the costs for idealized primitives such as ZPK protocols.

Computational complexity. Let us use Exp to denote an exponentiation time, and let ℓ_q be the bit length of q. The sender computes n_m ElGamal ciphertexts for n_m additive shares of its message with $2n_\mathsf{m}\mathsf{Exp}(\ell_q)$ operations in $\mathbb{F}_p[z]$, re-encryption of the mix server requires $2\mathsf{Exp}(\ell_q)$, and the receiver uses $n_\mathsf{m}\mathsf{Exp}(\ell_q)$ operations in $\mathbb{F}_p[z]$ for threshold decryption as well as one $\mathsf{Exp}(\ell_q)$ operation for key generation. In total, our protocol incurs $O(n_\mathsf{m}(n_\mathsf{s} + n_\mathsf{r})\mathsf{Exp}(\ell_q))$ operations in $\mathbb{F}_p[z]$.

In comparison, our protocol requires $O(n_\mathsf{m})$ exponentiations for the sender. In contrast, in most existing RMN protocols, the sender computes a single ElGamal ciphertext. However, the increase in the computation overhead occurs only on the sender's side, and in practice, the number of mix servers is often much smaller than the number of senders.

Transmission complexity. Likewise, the sender needs to transmit more ElGamal ciphertexts than in previous RMN solutions. That is, each sender sends $2n_\mathsf{m}$ elements in $\mathbb{G}_q$ elements, and thus, the total communication cost for the sender amounts to $2n_\mathsf{s}n_\mathsf{m}$ elements in $\mathbb{G}_q$ elements. Since the mix server sends a single ElGamal ciphertext and the receiver needs to exchange n_m elements in $\mathbb{G}_q$, their total communication cost is $n_\mathsf{m} + n_\mathsf{r}n_\mathsf{m}$ elements in $\mathbb{G}_q$. As a result, our protocol has $O(n_\mathsf{m}(n_\mathsf{s} + n_\mathsf{r}))$ communication complexity in terms of the elements in $\mathbb{G}_q$.

Round complexity. Finally, we examine the round efficiency of our protocol. In our protocol, after performing many local computations, such as homomorphic multiplication and re-encryption, the mix servers broadcast a single ElGamal ciphertext to the receivers through the ideal functionality of BC. Because the UC-secure broadcast [**GL05**] used in our protocol runs in constant rounds, the round complexity of our protocol is also constant for the mix servers. Moreover, all other idealized primitives have constant rounds, and the senders can complete transmission using only a single round via an idealized broadcast channel. Assuming a broadcast channel, decryption can be performed in a

constant number of rounds [**GJKR07**]. Note that over a secure point-to-point communication channel, the sender can send n_{m} ElGamal ciphertexts in parallel in a single round.

From our discussion thus far, we obtain the following:

Proposition 2. *Our RMN protocol given in Construction* **??** *runs in $O(1)$ rounds for all parties with $O(n_{\mathsf{m}}(n_{\mathsf{s}} + n_{\mathsf{r}}))$ computation and communication complexities in total.*

6.3 Security Proof

We prove that our protocol is secure in the UC framework. As mentioned in the Introduction, an RMN protocol may serve as a subprotocol of a high-layer application (e.g., e-voting), and thus we need a strong notion of security. Our proof is carried out in a hybrid model that assumes the existence of some ideal functionalities used in the protocol description (see Section**??**). As mentioned in Section **??**, each functionality can be instantiated using any protocol if the protocol can be proven to UC-realize the functionality.

An ideal functionality $\mathcal{F}_{\mathsf{MIX}}$ for RMN is specified in Section **??**. Now, we will show that our RMN protocol UC-realizes $\mathcal{F}_{\mathsf{MIX}}$ against a dishonest minority of static corruptions for the mix servers. We should note that we allow the static adversary to corrupt a dishonest majority of the senders. Recall that we use $\mathbb{H}$ to denote the hybrid model including the ideal functionalities $\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{DKg}}, \mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$, and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$. The main claim for the protocol is given in Proposition **??**.

Proposition 3. *Our RMN protocol given in Construction* **??** *UC-realizes the ideal RMN functionality $\mathcal{F}_{\mathsf{MIX}}$ in the $\mathbb{H}$-hybrid model with respect to an adversary that can statically corrupt at most a dishonest minority of mix servers and a dishonest majority of senders, assuming the DDH assumption in $\mathbb{G}_q$.*

Proof. We assume that all parties implicitly agree on a session ID as either a prefix of all messages or a default parameter and that they agree on the protocol parameter $\mathsf{pp} = (\mathbb{G}_q, g(z), p, q)$.

We first define a simulator $\mathcal{S}$ as follows. We fix the PPT environment $\mathcal{Z}$ and, as usual, assume a dummy adversary $\mathcal{A}$. Then, the simulator $\mathcal{S}$ runs a copy of $\mathcal{A}$, simulates the other parties for $\mathcal{A}$'s manipulation, and forwards all messages from $\mathcal{Z}$ to its simulated $\mathcal{A}$ and back. We remark that $\mathcal{S}$ has access to the randomness used for every ElGamal ciphertext of S_i and M_j since it is handed to a counterpart ZPK functionality, and the simulator $\mathcal{S}$ also knows the private key shares as well as the private key for each receiver R_k. This is because for the honest receivers, $\mathcal{S}$ generates the private key share, but the corrupt ones will send the private key shares to $\mathcal{F}_{\mathsf{DKg}}$ to prove that they know the private key shares.

We need to show that $\mathcal{Z}$ cannot distinguish between interacting with $\mathcal{A}$ in the $\mathbb{H}$-hybrid execution and interacting with $\mathcal{S}$ in an ideal execution, particularly when the simulator can access only $\mathcal{F}_{\mathsf{MIX}}$, with nonnegligible probability. Our strategy is to show that the DDH assumption can be broken using a simulator that does not imply the security of the protocol.

The simulator $\mathcal{S}$. Let $I_{\mathcal{P}}$ be the index set of parties of entity type $\mathcal{P} = \{\mathsf{S}, \mathsf{M}, \mathsf{R}\}$ that are corrupted by $\mathcal{A}$. Thus, the simulator $\mathcal{S}$ corrupts the dummy parties $\mathtt{S}_{i \in I_{\mathsf{S}}}$, $\mathtt{M}_{j \in I_{\mathsf{M}}}$, and $\mathtt{R}_{k \in I_{\mathsf{R}}}$.

The simulator $\mathcal{S}$ simulates all parties except for the corrupted parties $\mathsf{S}_{i \in I_{\mathsf{S}}}, \mathsf{M}_{j \in I_{\mathsf{M}}}$, and $\mathtt{R}_{k \in I_{\mathsf{R}}}$ under the real adversary $\mathcal{A}$. Thus, $\mathcal{S}$ simulates the honest parties S_i (i.e., $i \notin I_{\mathsf{S}}$) and the ideal functionalities $\mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$, and $\mathcal{F}_{\mathsf{DKg}}$ honestly. Likewise, it simulates all honest M_j for $j \notin I_{\mathsf{M}}$ and the functionalities $\mathcal{F}_{\mathsf{DKg}}, \mathcal{F}_{\mathsf{SC}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$, and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$ honestly. Then, it honestly simulates all $\mathsf{R}_{k \notin I_{\mathsf{R}}}$ as well as the functionalities $\mathcal{F}_{\mathsf{DKg}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{CD}}}$, and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$.

1. *Simulation of communications between corrupted parties and $\mathcal{Z}$.* We use $\tilde{\mathcal{Z}}$ to indicate a copy of a PPT machine $\mathcal{Z}$ for the hybrid model. The simulator $\mathcal{S}$ simulates all corrupted parties S_i for $i \in I_{\mathsf{S}}$, M_j for $j \in I_{\mathsf{M}}$, and R_k for $k \in I_{\mathsf{R}}$ such that it appears as though $\mathcal{Z}$ directly communicates with corrupted parties $\mathsf{S}_i, \mathsf{M}_j$, or R_k as follows:
   - If $\tilde{\mathcal{Z}}$ receives a message m from $\mathcal{A}$, then $\mathcal{S}$ writes m to $\mathcal{Z}$. However, if $\mathcal{S}$ receives m from $\mathcal{Z}$, then $\tilde{\mathcal{Z}}$ writes it to $\mathcal{A}$.
   - To simulate a corrupted party $\mathcal{P} \in \{\mathsf{S}_i, \mathsf{M}_j, \mathsf{R}_k\}$ directly communicating with $\mathcal{Z}$, where $i \in I_{\mathsf{S}}, j \in I_{\mathsf{M}}, k \in I_{\mathsf{R}}$, if $\tilde{\mathcal{Z}}$ receives a message m from $\mathcal{P}$, then $\mathtt{S}_i$ writes m to $\mathcal{Z}$. Conversely, if $\mathtt{S}_{i \in I_{\mathsf{S}}}$ receives m from $\mathcal{Z}$, then $\tilde{\mathcal{Z}}$ writes it to S_i.
2. *Extraction and computation for the senders.* When a corrupt sender $\mathsf{S}_{i \in I_{\mathsf{S}}}$ produces a ciphertext and informs $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$ that the input is valid, then $\mathcal{S}$ must instruct $\mathtt{S}_i$ to hand this as input to $\mathcal{F}_{\mathsf{MIX}}$.
   When a dummy sender $\mathtt{S}_i$ for $i \notin I_{\mathsf{S}}$ receives a message m_{ij} from $\mathcal{Z}$, $\mathcal{S}$ must ensure that S_i receives some message m'_{ij} from $\tilde{\mathcal{Z}}$. However, since $\mathcal{S}$ cannot see m_{ij}, it needs to hand some other message $m'_{ij} \neq m_{ij}$ to S_i. More specifically, this step is performed as follows:
   (a) The case of $i \in I_{\mathsf{S}}$. Before $\mathcal{S}$ receives $\langle(\mathcal{S} : \mathsf{M}_j, \mathtt{Forward}), \{\mathsf{M}_j : \mathtt{Forward}, \boldsymbol{l}_j^*\}_{j \in [n_{\mathrm{m}}]}\rangle$ from $\mathcal{Z}$, it proceeds as follows:
      - If $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$ receives $(\mathsf{S}_i : \mathtt{Prove}, (g(z), \beta(z), e_{ij}), r_{ij})$ such that $\langle(g(z), \beta(z), e_{ij}), r_{ij}\rangle \in \mathcal{R}_{\mathrm{PT}}$, then $\mathcal{S}$ checks whether the message $(\mathsf{S}_i : \mathtt{Send}, e_{ij}, \mathsf{M}_j)$ has been written to $\mathcal{F}_{\mathsf{SC}}$.
      - If $\mathcal{F}_{\mathsf{SC}}$ receives $((\mathsf{S}_i : \mathtt{Send}, e_{ij}, \mathsf{M}_j)$, it is determined whether $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$ recorded r_{ij} such that $\langle(g(z), \beta(z), e_{ij}), r_{ij}\rangle \in \mathcal{R}_{\mathrm{PT}}$.

      Using r_{ij}, $\mathtt{S}_i$ later recovers $m_{ij} = v_{ij}/\beta^{r_{ij}}$ and sends it to $\mathcal{F}_{\mathsf{MIX}}$. Then, $\mathcal{S}$ waits until it receives $(\mathcal{S} : \mathsf{S}_i, \mathtt{Set}, (i, c'_i))$ from $\mathcal{Z}$. It records (i, c_i, c'_i) and proceeds to the simulation of $\mathcal{F}_{\mathsf{SC}}$ and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{PT}}}$.
   (b) The case of $i \notin I_{\mathsf{S}}$. If $\mathcal{S}$ receives $(\mathcal{S} : \mathsf{S}_i, \mathtt{Set}, c'_i)$, then $\tilde{\mathcal{Z}}$ sets $m'_{ij} = 0$, chooses a random padding δ'_i, and sends (m'_{ij}, δ'_i) to S_i. Then, S_i chooses $r'_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $e_{ij} = \mathsf{E}_{pk}(m'_{ij} \parallel \delta'_i, r'_{ij})$. If $\mathcal{F}_{\mathsf{SC}}$ hands $(\mathcal{A} : \mathtt{Set}, \mathsf{S}_i, e_{ij}, (i, c'_i))$ to $\mathcal{Z}$, then the simulation is interrupted. Then, $\mathcal{S}$ records (i, c_i, c'_i) and continues the simulation.
   (c) If $\mathcal{F}_{\mathsf{SC}}$ receives $(\mathcal{A} : \mathsf{S}_i, \mathtt{Get}, (i, c_i))$, then the simulation of $\mathcal{F}_{\mathsf{SC}}$ is paused. If there exists a tuple (i, c_i, c'_i) for some c'_i, then $\mathcal{S}$ hands $(\mathcal{F}_{\mathsf{MIX}} : \mathtt{Get}, c'_i)$ to $\mathcal{Z}$. When $\mathcal{F}_{\mathsf{SC}}$ receives $(\mathcal{S} : \mathtt{S}_i, e_{ij})$, the simulation of $\mathcal{F}_{\mathsf{SC}}$ continues.

3. *Extraction and computation for the mix servers.* When a corrupt mix server $\mathsf{M}_{j\in I_\mathsf{M}}$ hands $(\mathsf{M}_j : \mathtt{Mix})$ to $\mathcal{F}_\mathsf{BC}$, $\mathcal{S}$ must ensure that $\mathtt{M}_j$ hands $(\mathtt{M}_j : \mathtt{Mix})$ to $\mathcal{F}_{mix}$. Likewise, if the honest dummy mix server $\mathtt{M}_j$ receives a message $(\mathsf{M}_j : \mathtt{Mix})$ from $\mathcal{Z}$, $\mathcal{S}$ must ensure that M_j receives a message $(\mathsf{M}_j : \mathtt{Mix})$ from $\tilde{\mathcal{Z}}$. Regarding the command $(\mathtt{Forward})$, $\mathcal{S}$ must ensure that if an honest $\mathsf{M}_{j\notin I_\mathsf{M}}$ hands $(\mathsf{M}_j : \mathtt{Forward}, \boldsymbol{l}_j^*)$ to $\mathcal{F}_\mathsf{BC}$, the honest dummy $\mathtt{M}_j$ does the same. More specifically, the process is as follows:
   - The case of $j \in I_\mathsf{M}$. Before $\mathcal{S}$ receives $\langle(\mathcal{S} : \mathsf{R}_k, \mathtt{Recover}, \boldsymbol{l}^\bullet), \{(\mathsf{R}_k : \mathtt{Recover}, \boldsymbol{l}^\bullet\}_{k\in[n_r]})\rangle$ from $\mathcal{Z}$, it proceeds as follows:
     - If $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$ receives $(\mathsf{M}_j : \mathtt{Prove}, \langle(g(z), \beta(z), e_j, e'_j), \gamma_j\rangle)$ such that $(g(z), \beta(z), e_j, e'_j) \in \mathcal{R}_{\mathrm{EDL}}$ from $\mathcal{Z}$, it searches for $(\mathsf{M}_j, (e'_j, e_j), c)$ in the table of $\mathcal{F}_\mathsf{BC}$.
     - If $\mathcal{F}_\mathsf{BC}$ receives a message $(\mathsf{M}_j : \mathtt{Mix})$, then $\mathcal{S}$ continues the simulation before the functionality hands $(\mathcal{A} : \mathtt{Mix}, \mathsf{M}_j, c)$ to $\mathcal{Z}$. Then, the simulation of $\mathcal{F}_\mathsf{BC}$ is interrupted, and $\mathtt{M}_j$ hands $(\mathtt{M}_j : \mathtt{Mix})$ to $\mathcal{F}_\mathsf{MIX}$. If the simulator $\mathcal{S}$ receives $(\mathtt{M}_j : \mathtt{Set}, c')$ from $\mathcal{Z}$, then it records a pair of counters (c, c') and restarts the simulation of $\mathcal{F}_\mathsf{BC}$.
   - The case of $j \notin I_\mathsf{M}$. If the simulator $\mathcal{S}$ receives $(\mathcal{S} : \mathtt{Set}, \mathtt{M}_j, c')$ from $\mathcal{Z}$, then $\tilde{\mathcal{Z}}$ hands $(\mathsf{M}_j, \mathtt{Mix})$ to $\mathcal{Z}$ and continues to simulate $\mathcal{F}_\mathsf{BC}$ before $\mathcal{F}_\mathsf{BC}$ hands $(\mathcal{A} : \mathtt{Set}, \mathtt{M}_j, \mathtt{Mix}, c)$ to $\mathcal{Z}$. Then, $\mathcal{S}$ records the pair of counters (c, c') and continues the simulation of $\mathcal{F}_\mathsf{BC}$.
   - If $\mathcal{F}_\mathsf{BC}$ receives a message $(\mathcal{A} : \mathtt{Get}, \mathsf{M}_j, c)$, the simulation of $\mathcal{F}_\mathsf{BC}$ is interrupted. Then, if the record (c, c') is found, $\mathcal{S}$ hands $(\mathcal{F}_\mathsf{MIX}) : \mathtt{Get}, c')$ to $\mathcal{Z}$. There is a pause until $\mathcal{S}$ receives $(\mathcal{S} : \mathtt{Mix}, \mathtt{M}_j)$ or $\langle(\mathcal{S} : \mathtt{Forward}, \mathtt{M}_j, \boldsymbol{l}_j^*), \{(\mathtt{M}_j : \mathtt{Forward}, \boldsymbol{l}'_j)\}_{j\in[n_\mathrm{m}]}\rangle$ from $\mathcal{Z}$. Then, $\mathcal{S}$ proceeds with the simulation of $\mathcal{F}_\mathsf{BC}$.
4. *Extraction and computation for the receivers.* When a corrupt receiver R_k for $k \in I_\mathsf{R}$ writes a message $(\mathsf{R}_k : \mathtt{Recover})$ on $\mathcal{F}_\mathsf{BC}$, $\mathcal{S}$ must ensure that $\mathtt{R}_k$ hands $(\mathtt{R}_k : \mathtt{Recover})$ to $\mathcal{F}_\mathsf{MIX}$. If an honest dummy receiver $\mathtt{R}_k$ receives a message $(\mathtt{R}_k : \mathtt{Recover})$ from $\mathcal{Z}$, then $\mathcal{S}$ must ensure that R_k receives $(\mathsf{R}_k : \mathtt{Recover})$ from $\tilde{\mathcal{Z}}$. To do this, $\mathcal{S}$ first sends a message and instructs $\mathcal{F}_\mathsf{MIX}$ to accept the given message. The honest receiver R_k outputs $(\mathsf{R}_k : \mathtt{Recover}, \boldsymbol{m})$, and $\mathcal{S}$ should ensure that $\mathtt{R}_k$ does the same. This process is as follows:
   - The case of $k \in I_\mathsf{R}$. If $\mathcal{F}_\mathsf{BC}$ receives a message $(\mathsf{R}_k : \mathtt{Recover})$, then $\mathcal{S}$ continues the simulation until just before $\mathcal{F}_\mathsf{BC}$ hands $(\mathcal{A}, \mathsf{R}_k, \mathtt{Recover}, c)$ to $\mathcal{Z}$. Then, it interrupts the simulation of $\mathcal{F}_\mathsf{BC}$ and hands $(\mathtt{R}_k : \mathtt{Recover})$ to $\mathcal{F}_\mathsf{MIX}$. When $\mathcal{S}$ receives $(\mathtt{R}_k : \mathtt{Set}, c')$ from $\mathcal{Z}$, it records a pair of counters (c, c') and returns to the simulation of $\mathcal{F}_\mathsf{BC}$.
   - The case of $k \notin I_\mathsf{R}$. If $\mathcal{S}$ receives $(\mathcal{S} : \mathtt{Set}, \mathtt{R}_k, c)$ from $\mathcal{Z}$, then $\tilde{\mathcal{Z}}$ hands $(\mathsf{R}_k : \mathtt{Recover})$ to R_k and proceeds with the simulation before $\mathcal{F}_\mathsf{BC}$ hands $(\mathcal{A} : \mathtt{Set}, \mathtt{R}_k, c)$ to $\mathcal{Z}$. Then, it records (c, c') and continues the simulation.
   - If $\mathcal{F}_\mathsf{BC}$ receives a message $(\mathcal{A} : \mathtt{Get}, \mathtt{R}_k, c)$, then the simulation of $\mathcal{F}_\mathsf{BC}$ is interrupted. If the pair (c, c') is found in the table under index c', then $\mathcal{S}$ hands $(\mathcal{F}_\mathsf{MIX} : \mathtt{Get}, c')$ to $\mathcal{Z}$. It waits until it receives $(\mathcal{S} : \mathsf{R}_k, \mathtt{Receive})$ or $((\mathcal{S} : \mathsf{R}_k, \mathtt{Recover}, \boldsymbol{l}^\bullet), \{(\mathsf{R}_k : \mathtt{Recover}, \boldsymbol{l}^\bullet)\}_{k\in[n_r]})$ from $\mathcal{Z}$. Then, the simulation of $\mathcal{F}_\mathsf{BC}$ continues.

5. *Replacing each fake message $m'_{ij} = 0$ with a real message m_{ij}* for $i \notin I_{\mathsf{S}}$. Since the simulator $\mathcal{S}$ does not know the real messages m_{ij} of the honest dummy senders $\mathtt{S}_i$ for $i \notin I_{\mathsf{S}}$, we need to address this flaw to obtain a correct simulation. Fortunately, as constructed above, because $\mathcal{S}$ has received $(\mathcal{S} : \mathsf{R}_k, \mathtt{Recover}, \boldsymbol{l}^\bullet)$, where $\boldsymbol{l}^\bullet = \{m_i \parallel \delta_i\}_{i\in[n_{\mathsf{s}}]}$, it can build a set $\widehat{\boldsymbol{m}} = \boldsymbol{l}^\bullet \backslash \{m_i \parallel \delta_i\}_{i\in I_{\mathsf{S}}}$. Of course, it cannot see which message in $\widehat{\boldsymbol{m}}$ is owned by which sender, but we can replace $m'_{ij} = 0$ with a message in $\widehat{\boldsymbol{m}}$. For simplicity of discussion, we fix an honest mix server M_τ. This is done as follows:
   - For $i \notin I_{\mathsf{S}}$, compute $m_i = \oplus_{j\in[n_{\mathsf{m}}]} m_{ij}$ and build a list $(m_{ij} \parallel \delta_i)_{j\in[n_{\mathsf{m}}]}$. Then, for each j, choose $r_{ij} \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $e_{ij} \leftarrow \mathtt{ElG}.\mathsf{E}_{pk}(m_{ij} \parallel \delta_i, r_{ij})$.
   - Modify M_τ and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$ so that they pretend to run with the real messages.
     - Modify Step 3c: $\mathcal{S}$ computes $e'_\tau = \times_{\mathtt{h}i=1} n_{\mathsf{s}} e_{i\tau}$ with the newly generated ciphertexts for the honest senders and re-encrypts it as e_τ with $\gamma_\tau \xleftarrow{\$} \mathbb{Z}_q^*$. Then, it hands $(\mathsf{M}_\tau : \mathtt{Prove}, \cdots)$ to $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$ and $(\mathsf{M}_\tau : \mathtt{Broadcast}, \{\mathsf{R}_k\}_k)$ to $\mathcal{F}_{\mathsf{BC}}$.
     - If $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{\mathrm{EDL}}}$ receives $(\mathsf{R}_k : \mathtt{Verify}, (g(z), \beta(z), e_\tau, e'_\tau,))$, then we check whether it has received $(\mathsf{M}_\tau : \mathtt{Prove}, \cdots)$ and, if so, set $b_{k_2} = 1$; we set $b_{k_2} = 0$ otherwise.

     We note that we used the same randomness r_{ij}, γ_j in the modification for all corrupted parties, and the ciphertexts of all corrupted parties are distributed exactly as in the real protocol.

Defining a sequence of hybrid execution. For proof by contradiction, we assume that the ideal adversary $\mathcal{S}$ does not successfully simulate the real adversary $\mathcal{A}$ and then show that we can break the DDH assumption.

We assume that $\mathcal{S}$ does not ensure the security of our protocol Π_{MIX} in Construction **??**. Then, there exists a hybrid adversary $\mathcal{A}$, an environment $\mathcal{Z}$, and a constant λ such that for $\nu \in \mathbb{N}$,

$$|\Pr[\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{MIX}},\mathcal{S},\mathcal{Z}} = 1] - \Pr[\mathsf{REAL}_{\Pi_{\mathsf{MIX}},\mathcal{A},\mathcal{Z}} = 1]| \geqslant 1/\nu^\lambda.$$

- *A sequence of hybrids.* For convenience of exposition, let $J_{\mathsf{S}} = [n_{\mathsf{s}}] \backslash I_{\mathsf{S}}$. Then, consider a sequence of hybrids $H_0, \ldots, H_{|J_{\mathsf{S}}|}$, where H_0 is the ideal environment consisting of the ideal adversary $\mathcal{S}$ and the ideal functionality $\mathcal{F}_{\mathsf{MIX}}$ with dummy parties; i.e., $H_0 = \mathcal{Z}(\mathcal{F}_{\mathsf{MIX}}, \mathcal{S}, \{\mathtt{S}_i\}_i, \{\mathtt{M}_j\}_j, \{\mathtt{R}_k\}_k)$, with some abuse of notation. As usual, we define H_ℓ by the following modification of H_0:
  1. If $\mathcal{S}$ receives $(\mathtt{S}_i : \mathtt{Send})$ from $\mathcal{F}_{\mathsf{MIX}}$ for $i \in J_{\mathsf{S}}$, then it checks whether $i \in [\ell]$.
  2. If $i \in [\ell]$, it finds the message m_{ij} of $\mathtt{S}_i$ that was sent. Then, $\tilde{\mathcal{Z}}$ sends m_{ij} to S_i. In this case, we treat S_i as a corrupt sender and thus set $r_{ij} = r'_{ij}, m'_{ij} = m_{ij}$.
  3. Otherwise, $\tilde{\mathcal{Z}}$ chooses a random message m'_{ij} and sends it to S_i, as in the original simulation.

  Then, the output of $H_{|J_{\mathsf{S}}|}$ is distributed identically to the output of $\mathcal{Z}(\Pi_{\mathsf{MIX}}, \mathcal{A}, \{\mathsf{S}_i\}_i, \{\mathsf{M}_j\}_j, \{\mathsf{R}_k\}_k)$ except that M_τ cannot set the real messages; however, this is not noticed. If we set $\rho_\ell = \Pr[H_\ell = 1]$, then we have $|\rho_\ell - \rho_{\ell-1}| \geqslant \frac{1}{|J_{\mathsf{S}}|\kappa^\lambda} \geqslant \frac{1}{n_{\mathsf{s}}\kappa^\lambda}$ by a simple calculation.

– *A distinguishing algorithm D.* Finally, we describe a distinguisher D for the DDH experiment. Thus, D is given the following test, where an oracle chooses $r, r', \alpha \in \mathbb{Z}_q^*$ and a bit $b \xleftarrow{\$} \{0,1\}$:
  - If $b = 0$, define $(g, u, w, v) = (g(z), g(z)^r, g(z)^\alpha, g(z)^{r'})$.
  - If $b = 1$, define $(g, u, w, v) = (g(z), g(z)^r, g(z)^\alpha, g(z)^{r\alpha})$.

  The point of constructing the distinguisher D is that it can embed w into the public key because this process does not change the key distribution without knowledge of the private key α. The distinguisher D continues to simulate H_ℓ before S_ℓ receives the message $(\mathsf{S}_\ell : \mathtt{Send})$. Then, it computes $(u_{\ell\tau}, v_{\ell\tau}) = (u, m_{\ell\tau} \cdot v)$ and instructs S_ℓ to hand $(\mathsf{S}_\ell : \mathtt{Send}, (u_{\ell\tau}, v_{\ell\tau}), \mathsf{M}_\tau)$ and $\mathcal{F}_{\mathsf{ZPK}}^{\mathcal{R}_{rpt}}$ to output $b_{j_1} = 1$. Then, D continues the simulation of H_ℓ until it outputs a bit b', and this bit is its final output. By construction, if $b = 0$, then both elements $u_{\ell\tau}, v_{\ell\tau}$ are random elements in $\mathbb{G}_q$, which corresponds to Step 2 in the description of the hybrid sequence. Thus, the output of D is identically distributed to that of $H_{\ell-1}$, and the two elements are also identically distributed to the corresponding ciphertexts in the simulation. On the other hand, if $b = 1$, then $u_{\ell\tau}, v_{\ell\tau}$ is a valid ciphertext with randomizer r. This corresponds to Step 3 in the hybrid description; thus, the output of D is identically distributed to that of H_ℓ. Thus, we have

$$|\Pr[D(\mathbb{G}_q, q, g(z), g(z)^r, g(z)^\alpha, g(z)^{r'}) = 1] - \Pr[D(\mathbb{G}_q, q, g(z), g(z)^r, g(z)^\alpha, g(z)^{r\alpha}) = 1]| \geqslant \frac{1}{n_{\mathsf{s}}\kappa^\lambda}.$$

This completes the proof of the theorem. □

7 Concluding Remarks

In this work, we suggest a new technique for designing a constant RMN protocol while UC-realizing a hybrid model with access to some functionalities for secure communication and several efficient ZKPs. Our main tool is secret sharing and generalized ElGamal encryption under field extension. The extended ElGamal encryption scheme allows us to utilize a decomposition capability that can efficiently and uniquely factorize a decrypted message into irreducible elements. In particular, one of the outstanding features is that our protocol does not require a ZKP of correct shuffling; thus, we obtain a considerable efficiency gain in terms of computation cost.

NewApproachContRndRMN