A Public Shuffle without Private Permutations

Myungsun Kim, Jinsu Kim, and Jung Hee Cheon

Dep. of Mathematical Sciences, Seoul National University
1 Gwanak-ro, Gwanak-gu, Seoul 151-747, Korea
{msunkim,kjs2002,jhcheon}@snu.ac.kr

Abstract. In TCC 2007, Adida and Wikström proposed a novel approach to shuffle, called a public shuffle, in which a shuffler can perform shuffle publicly without needing information kept secret. Their scheme uses an encrypted permutation matrix to shuffle ciphertexts publicly. This approach significantly reduces the cost of constructing a mix-net to verifiable joint decryption. Though their method is successful in making shuffle to be a public operation, their scheme still requires that some trusted parties should choose a permutation to be encrypted and construct zero-knowledge proofs on the well-formedness of this permutation.

In this paper, we propose a method to construct a public shuffle without relying on permutations and randomizers generated privately: Given an n-tuple of ciphertext $(c_1, \ldots, c_n)$, our shuffle algorithm computes $f_i(c_1, \ldots, c_n)$ for $i = 1, \ldots, \ell$ where each $f_i(x_1, \ldots, x_n)$ is a symmetric polynomial in $x_1, \ldots, x_n$. Depending on the symmetric polynomials we use, we propose two concrete constructions. One is to use ring homomorphic encryption with constant ciphertext complexity and the other is to use simple ElGamal encryption with linear ciphertext complexity in the number of senders. Both constructions are free of zero-knowledge proofs and publicly verifiable.

Keywords: secret shuffle, public shuffle, private permutation, mix-net, ElGamal encryption

1 Introduction

Given n distinct elements, $(m_1, \ldots, m_n)$, from each sender, a shuffle is an n-party functionality that allows all users to learn $\bigcup_{i=1}^{n}\{m_i\}$, but does not reveal any information on link between m_i and its sender without negligible probability. Shuffles can be used in various applications including e-voting and private set union ensured to hide the link between messages and their senders. Chaum [12] firstly provided a way to mix messages by combining a private permutation and a fresh randomness, which is called a mix-net. Here the private permutation means the random permutation of each mix-server. A mix-net consists of multiple mix-servers which have their private permutation and randomizers. If a mix-net consists of a single mix-server, then the mix-server knows who sent what message. Thus, there must be at least one honest mix-server in a mix-net.

The assumption that there exists an honest mix-server (a.k.a., a trusted third party) in real life, however, may be quite strong. Thus many researchers have focused on strengthening verifiability in Chaum's construction (e.g., [18,26,27,17,35,21]). Their goal is to efficiently enforce each mix-server to behave as being in public under the assumption. When a shuffle allows public verifiability, in general, by using zero-knowledge proofs, but requires a secret permutation and randomizers, Neff [26] (and later Groth [20]) call it a verifiable *secret shuffle*.

In TCC 2007, Adida and Wikström [3] proposed a way by which mix-servers carry out shuffling in public. Their work is based on the notion of public-key obfuscation studied by Ostrovsky and Skeith [29] for different purposes. Later, in PKC 2012, Parampalli et al. [31] provided efficiency improvements using permutation networks instead of permutation matrix. Very informally, their basic idea is that mix-servers precompute their private permutation and then publish it in public. Though secret information is concealed by a homomorphic cryptosystem, it should be generated by a trusted party. Here and in what follows, we call their work a public shuffle with a private permutation. In this paper, we will try to construct a verifiable *public* shuffle *without* a private permutation.

1.1 Our Contributions and Underlying Ideas

In this work, we consider the problem of constructing a verifiable public shuffle. Our contributions are twofold: (1) definitions; (2) constructions.

Motivations. As mentioned above, in [3] shuffles are precomputed with a random permutation and randomizers and published in public together with zero-knowledge proofs. Although shuffling can be run in public, secret information used in precomputing is assumed to be kept secret. In order to find the possibility of removing the secret information that precomputing shuffles needs to use, we consider homomorphic tallying since only public computation is required for the anonymization process. Indeed, Benaloh and Yung [8] proposed a Yes/No voting scheme using homomorphic tallying.[1] However, homomorphic tallying cannot recover the individual input plaintexts. This can be problematic in some cases including write-in votes. One feasible solution is to encode input messages into primes before encrypting them. However, this way has two limitations: (1) the ciphertext space should be large; (2) recovering the original messages (e.g., factorization over $\mathbb{Z}$) may require exponential computation complexity. In this paper, we give verifiable public shuffles that require only public computation, and support the original message recovery in polynomial time.

Our definitions. In [28], the authors define a shuffle over a re-randomizable public-key cryptosystem as a polynomial-time algorithm that takes a set of n input ciphertexts and a random permutation, and outputs a set of n output ciphertexts. Other definitions do not make a big difference from this. As we will show later, this definition seems too restrictive to exploit all possibilities for achieving a construction that roughly corresponds to our goal. Our definitional approach consists of two steps. First, we relax the restriction that the number of output ciphertexts should be equal to that of input ciphertexts. We call it *generalized shuffle*. Our interpretation of verifiable secret shuffles is that they play a role of hiding the order of input ciphertexts using a secret permutation and a fresh randomness. In contrast, our verifiable public shuffles remove the order of input ciphertexts itself. In Section 2, we formally define this concept. Then, we formally describe what means by a secure shuffle with respect to verifiability and unlinkability (in [28] the authors called it shuffle privacy) in Section 2.2.

Our constructions. Our construction of verifiable public shuffles also consists of two steps. First, in Section 3, we show how construct a verifiable public shuffle from a ring homomorphic cryptosystem. We would like to stress that if we assume a ring homomorphic cryptosystem, this construction is a more or less straightforward result, and therefore may seem obvious in hindsight, but it is actually non-trivial as long as a group homomorphic cryptosystem is concerned. In Section 4, we then show how to construct public shuffle schemes from a group homomorphic cryptosystem.

Our idea is to use a homomorphic encryption Enc on a Unique Factorization Domain (UFD) R and symmetric polynomials $f_1, \ldots, f_\ell \in R[x_1, \ldots, x_n]$ satisfying

$$f_i(\mathsf{Enc}_{pk}(m_1), \ldots, \mathsf{Enc}_{pk}(m_n)) = \mathsf{Enc}_{pk}(f_i(m_1, \ldots, m_n))$$

for $m_1, \ldots, m_n \in R$. Given an n-tuple of ciphertexts $(c_1, \ldots, c_n)$ with $c_i = \mathsf{Enc}_{pk}(m_i)$, our shuffle algorithm outputs $f_i(c_1, \ldots, c_n) = \mathsf{Enc}_{pk}(f_i(m_1, \ldots, m_n))$ for $i = 1, \ldots, \ell$. This output is not a shuffle of $(c_1, \ldots, c_n)$, but plays the same role with it, i.e. their decryption can be transformed into the set of original messages $\{m_1, \ldots, m_n\}$ using factorization on $R[x]$, which is a UFD. It is easy to see that this shuffle provides *unlinkability* between inputs and outputs because a permutation of inputs does not result in changes of the output of shuffle.

[1] Due to the Paillier cryptosystem [30], even though the message space dramatically increased, aggregate voting schemes [13,14,6] do not consider the original message recovering.

Using a ring homomorphic cryptosystem, we can construct a public shuffle with $O(1)$ ciphertext complexity in the number of senders. However, ring homomorphic cryptosystems are highly expensive and not practical yet. Thus, we construct public shuffles using a group homomorphic encryption–CCA1-secure ElGamal encryption, at the cost of $O(n)$ ciphertext complexity. Note that a basic public shuffle without relying on a trusted third party yields $O(n^2)$ ciphertext complexity where n is the number of senders.

Our construction using a ring homomorphic encryption has $O(n)(\mathsf{E}+\mathsf{D})+O(n^2 \log p)\mathsf{M}_{\mathbb{F}_p}$ computational complexity, where E, D, and $\mathsf{M}_{\mathbb{F}_p}$ denote the cost of encryption, decryption and multiplication in $\mathbb{F}_p$, respectively. The construction using CCA1-secure ElGamal encryption over $\mathbb{F}_{p^3}$ has $O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ computational complexity. In contrast, the Adida and Wikström scheme requires $O(n^2)$ exponentiations to precompute and evaluate.

1.2 Related Work

Shuffles were introduced by Chaum [12] as a new primitive that can be used to build a mix-net, and the problem of verifiable shuffles was introduced by Sako and Kilian in [33]. Since then the work on verifiable shuffles in the next years has been extensive and varied [1,2,18,26,27,17,35,21,36,7]. Abe [1] considered the problem of compact proofs of shuffles. Later Furukawa and Sako [18] use a permutation matrix to shuffle the ciphertexts More recently, generalizing the Neff's scheme [26], Groth and Lu [21] give a verifiable shuffle that is non-interactive, uses paring-based verifiability, and obtains linear proof size in the number of senders. In Eurocrypt 2012, Bayer and Groth [7] achieves sub-linear proof size in the number of senders. The common properties that all of these schemes hold are that a shuffler must keep his permutation and randomness secret. Therefore, if we build a mix-net using one of these schemes, then we should assume that there exists at least one honest shuffler (a.k.a a mix-server). To weaken such a strong assumption, Adida and Wikström [3] proposed a way to shuffle the ciphertexts in public. Later Parampalli et al. [31] improved the computational efficiency. However, a private permutation and secret randomizers are still required in the setup phase but not in the shuffle phase. The advantage, as outlined above, that our construction has over all of these is that our shuffles do not require any secret information such as permutations and randomizers; we do not require expensive zero-knowledge proofs to support public verifiability. See Appendix E for details on concurrent related work.

2 Generalized Shuffle

Notation. For $n \in \mathbb{N}$, $[1, n]$ denotes the set $\{1, \ldots, n\}$. If A is a probabilistic polynomial-time (PPT) machine, we use $a \leftarrow A$ to denote A which produces output according to its internal randomness. In particular, if U is a set, then $r \xleftarrow{\$} U$ is used to denote sampling from the uniform distribution on U. For an integer a, $\|a\|$ denotes the bit length of a.

We shall write

$$\Pr[x_1 \xleftarrow{\$} X_1, x_2 \xleftarrow{\$} X_2(x_1), \ldots, x_n \xleftarrow{\$} X_n(x_1, \ldots, x_{n-1}) : \varphi(x_1, \ldots, x_n)]$$

to denote the probability that when x_1 is drawn from a certain distribution X_1, and x_2 is drawn from a certain distribution $X_2(x_1)$, possibly depending on the particular choice of x_1, and so on, all the way to x_n, the predicate $\varphi(x_1, \ldots, x_n)$ is true.

A function $g : \mathbb{N} \to \mathbb{R}$ is negligible if for every positive polynomial $\mu(\cdot)$ there exists an integer N such that $g(n) < 1/\mu(n)$ for all $n > N$.

Let $\mathcal{R}(\cdot, \cdot)$ be a polynomial-time computable relation in the size of its first input. Associated with $\mathcal{R}$, we consider a language $\mathcal{L}_{\mathcal{R}} = \{x : \exists\, w \text{ such that } \mathcal{R}(x, w) = 1\}$. A proof system $(\mathcal{P}, \mathcal{V})$ for a relation

$\mathcal{R}$ allowing a prover $\mathcal{P}$ to prove that a value x is in the associated language $\mathcal{L}_{\mathcal{R}}$ consists of two PPT algorithms: The algorithm $\mathcal{P}$ that outputs a proof Γ that $\Gamma \in \mathcal{L}_{\mathcal{R}}$ and the algorithm $\mathcal{V}$ that verifies the proof.

2.1 Definitions

In this section, we give a formal definition of generalized shuffle in a public-key setting. We begin with the definition of public-key encryption, following its variants supporting group operations or ring operations.

Definition 1 *A* public-key cryptosystem $\mathcal{E}$ *is a 3-tuple of PPT algorithms* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *such that*

1. *The key generation algorithm* KG *takes as input the security parameter* λ *and outputs a pair of keys* (pk, sk)*. For given* pk*, the message space* $\mathbf{M}_{pk}$ *and the randomness space* $\mathbf{R}_{pk}$ *are uniquely determined.*
2. *The encryption algorithm* Enc *takes as input a public key* pk *and a message* $m \in \mathbf{M}_{pk}$*, and outputs a ciphertext* $c \in \mathbf{C}_{pk}$ *where* $\mathbf{C}_{pk}$ *is a finite set of ciphertexts. We write this as* $c \leftarrow \mathsf{Enc}_{pk}(m)$*. We sometimes write* $\mathsf{Enc}_{pk}(m)$ *as* $\mathsf{Enc}_{pk}(m, r)$ *when the randomness* $r \in \mathbf{R}_{pk}$ *used by* Enc *needs to be emphasized. .*
3. *The decryption algorithm* Dec *takes as input a private key* sk *and a ciphertext* c*, and outputs a message* m *or a special symbol* $\perp$ *which means failure.*

We say that a public-key cryptosystem $\mathcal{E}$ is *correct* if, for any key-pair $(pk, sk) \leftarrow \mathsf{KG}(\lambda)$ and any $m \in \mathbf{M}_{pk}$, it is the case that: $m \leftarrow \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$.

Definition 2 ([19]) *A public-key cryptosystem* $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *with a security parameter* λ *is called to be* semantically secure (IND-CPA secure) *if after the standard CPA game being played with any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, the advantage* $\mathbf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(\lambda)$*, formally defined as*

$$\left| \Pr_{b,r} \left[\begin{array}{l} (pk, sk) \leftarrow \mathsf{KG}(\lambda), (\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_1(pk), \\ c = \mathsf{Enc}_{pk}(m_b; r) : b \leftarrow \mathcal{A}_2(\mathsf{state}, m_0, m_1, c) \end{array} \right] - \frac{1}{2} \right|,$$

is negligible in λ *for all sufficiently large* λ*.*

In the experiment above, when we allow $\mathcal{A}_1$ to query the decryption oracle, if the advantage $\mathbf{Adv}^{\mathsf{cca}}_{\mathcal{E},\mathcal{A}}(\lambda)$ is negligible, we say $\mathcal{E}$ is IND-CCA1 secure, in short, CCA1 secure.

For a public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ with an additional randomized algorithm ReRand that, on input a ciphertext outputs a new ciphertext with the same message, a given adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, let $\mathbf{Adv}^{\mathsf{rerand}}_{\mathcal{E},\mathcal{A}}(\lambda)$ be the advantage of the following game:

$$\left| \Pr_{b} \left[\begin{array}{l} (pk, sk) \leftarrow \mathsf{KG}(\lambda), (\mathsf{state}, c) \leftarrow \mathcal{A}_1(pk), \\ \hat{c} = \begin{cases} \mathsf{Enc}_{pk}(\mathsf{Dec}_{sk}(c)) & \text{if } b = 0 \\ \mathsf{ReRand}_{pk}(c) & \text{if } b = 1 \end{cases} : b' \leftarrow \mathcal{A}_2(\mathsf{state}, c, \hat{c}) \end{array} \right] - \frac{1}{2} \right|.$$

We say that the public-key encryption scheme is *re-randomizable* if for all PPT algorithms $\mathcal{A}$, the advantage in the game above is negligible in λ.

Most public-key cryptosystems are defined over algebraic groups or rings, such as $\mathbb{Z}_N^{\times}$ or $\mathbb{Z}_N$. Public-key cryptosystems defined over a group naturally support a single operation, usually denoted by multiplication or addition, and cryptosystems defined over a ring naturally support two operations, usually denoted by addition and multiplication. Thus, if the encryption algorithm for a public-key cryptosystem, where both the message space and the ciphertext space are groups (or rings), is homomorphic, then such public-key cryptosystems are referred to as homomorphic cryptosystems. Now we define them more formally.

Definition 3 *A* group homomorphic *cryptosystem is a public-key cryptosystem* $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *where the set of possible messages* $\mathbf{M}_{pk}$ *and the set of possible ciphertexts* $\mathbf{C}_{pk}$ *are both groups such that for any public key* pk *and any two ciphertexts* $c_1 \in \mathsf{Enc}_{pk}(m_1), c_2 \in \mathsf{Enc}_{pk}(m_2)$*, the following condition holds:*

$$\mathsf{Dec}_{sk}(c_1 \cdot c_2) = m_1 \cdot m_2$$

where $\cdot$ *represents the respective group operations in* $\mathbf{C}_{pk}$ *and* $\mathbf{M}_{pk}$*. When additive notation is used,* $\mathsf{Dec}_{sk}(c_1 + c_2) = m_1 + m_2$*.*

We can easily define a homomorphic encryption scheme with a re-randomization algorithm using a similar way above.

Definition 4 *A* ring homomorphic *cryptosystem is a public-key cryptosystem where the set of possible messages* $\mathbf{M}_{pk}$ *and the set of possible ciphertexts* $\mathbf{C}_{pk}$ *are both rings such that for any public key* pk *and any two ciphertexts* $c_1 \in \mathsf{Enc}_{pk}(m_1), c_2 \in \mathsf{Enc}_{pk}(m_2)$*, the following conditions hold:*

1. $\mathsf{Dec}_{sk}(c_1 + c_2) = m_1 + m_2$
2. $\mathsf{Dec}_{sk}(c_1 \cdot c_2) = m_1 \cdot m_2$

where $+$ *and* $\cdot$ *represent the respective ring operations in* $\mathbf{C}_{pk}$ *and* $\mathbf{M}_{pk}$*.*

Now we describe the syntax of a generalized shuffle. First, we rephrase the formal definition of a verifiable shuffle given by Nguyen et al. [28, Def. 4]. In [28] they extensively use a re-randomizable public-key encryption scheme. We then extend it to the definition of a generalized shuffle. We additionally introduce some notation used to define public verifiability.

Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ be an encryption scheme with a re-randomization algorithm satisfying semantic security. Let $\mathbf{c}, \hat{\mathbf{c}}$ be two lists of ciphertexts, but all elements of each list belong to the ciphertext space $\mathbf{C}_{pk}$ defined in $\mathcal{E}$. We use Σ_n to denote the set of all permutations on $[1, n]$. For a set $X = \{a_1, \ldots, a_n\}$, we denote by $|X|$ the number of elements in the set, i.e., $|X| = n$. Let $\Phi(\cdot, \cdot)$ be an efficient *shuffle relation* that holds if the witness $w = \left(\pi, s_1, \ldots, s_{|\mathbf{c}|}\right)$ demonstrates that $|\mathbf{c}| = |\hat{\mathbf{c}}|$ and

$$\exists \left(\pi, s_1, \ldots, s_{|\mathbf{c}|}\right), \forall\, i \in [1, |\mathbf{c}|] : \hat{c}_i = \mathsf{ReRand}_{pk}(c_{\pi(i)}, s_{\pi(i)}) \tag{2.1}$$

where $\pi \in \Sigma_{|\mathbf{c}|}, c_i \in \mathbf{c}$, and $\hat{c}_{\pi(i)} \in \hat{\mathbf{c}}$. Associated with Φ, we define a language $\mathcal{L}_\Phi = \{x = (\delta, \mathbf{c}, \hat{\mathbf{c}}) : \exists\, w$ such that $\Phi(x, w) = 1\}$ where δ is a public parameter including pk.

Definition 5 *A* verifiable shuffle *scheme* $\Phi_{\mathcal{E}}$ *over a re-randomizable public-key cryptosystem* $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ *is a triple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ *which works as follows:*

- $\delta \leftarrow \mathsf{Setup}(\lambda, n)$: *The setup algorithm takes as input a security parameter* λ *and* $n \in \mathbb{N}$*, and outputs a public parameter* $\delta := (pk, \Sigma_n)$ *where* $pk \leftarrow \mathsf{KG}(1^\lambda)$*.*
- $(\hat{\mathbf{c}}, \Gamma) \leftarrow \mathsf{Shuffle}(\delta, w, \mathbf{c})$: *First the shuffle algorithm generates a random permutation* $\pi \in \Sigma_n$ *and a list of randomness* $(s_1, \ldots, s_n) \in (\mathbf{R}_{pk})^n$*, and sets the secret parameter* $w = (\pi, s_1, \ldots, s_n)$*. Using the public parameter* δ *and secret parameter* w*, the shuffle algorithm encodes a list of ciphertexts* $\mathbf{c} = (c_1, \ldots, c_n)$ *as a shuffled set of ciphertexts* $\hat{\mathbf{c}} = \{\hat{c}_1, \ldots, \hat{c}_n\}$ *such that* $\mathsf{Dec}_{sk}\left(c_{\pi(i)}\right) = \mathsf{Dec}_{sk}\left(\hat{c}_i\right)$ *for all* $i \in [1, n]$ *where* $c_i = \mathsf{Enc}_{pk}(m_i, r_i)$ *and* $\hat{c}_i = \mathsf{ReRand}_{pk}\left(c_{\pi(i)}, s_{\pi(i)}\right)$*. Finally it forms a proof* Γ *for the shuffle performed by the shuffler in possession of* $\pi \xleftarrow{\$} \Sigma_n$ *and a list of randomness* $\{s_1, \ldots, s_n\}$*.*
- $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{Verify}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$: *The verification algorithm takes as input the public parameter* δ*, two lists of ciphertexts* $\mathbf{c}, \hat{\mathbf{c}}$ *and a proof* Γ*, and checks the validity of the proof by running* $(\mathcal{P}, \mathcal{V})(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$*; if this fails output* reject *and otherwise output* accept*.*

When the shuffle algorithm requires the secret parameter in order to output a permuted and re-randomized version of input ciphertexts, we call it *secret* shuffle. If the verification algorithm does not requires any secret parameter, we call it *(publicly) verifiable* shuffle. Thus, if w is secret but Verify does not take it as input, we call this type of shuffle schemes publicly verifiable secret shuffle. We remark that decryption shuffles also belong to secret shuffle because they use a random secret permutation in shuffling.

As a symmetry it is not difficult to make the definition of publicly verifiable public shuffle or *public* shuffle for short. Namely, public shuffle is a publicly verifiable shuffle scheme such that its shuffle algorithm also does not require any secret parameter. However, it is not easy to design and construct a public shuffle scheme following Definition 5. Although Adida and Wikström [3] and Parampalli et al. [31] achieve public shuffle by utilizing the public-key obfuscation technique, secret parameters in their schemes are required in the setup algorithm instead of the shuffle algorithm. To remove dependencies on secret parameters in a shuffle scheme, we first consider how to construct a secret shuffle without a secret permutation as a intermediate step toward public shuffle. However, we observed that it is difficult to achieve a secret shuffle without requiring a secret permutation under the legacy definition. Hence, we will relax the shuffle definition above in order to realize the notion of public shuffle. In particular, it is worth noting it has been a long standing hard problem to design a secure shuffle protocol without relying on TTP.

Definition 6 (Generalized Shuffle) *Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReRand})$ be a re-randomizable public-key cryptosystem with semantic security. A* generalized shuffle *scheme $\tilde{\Phi}_{\mathcal{E}}$ over $\mathcal{E}$ is a triple of PPT algorithms as defined in Definition 5 except for*

- $(\delta, w) \leftarrow \mathsf{Setup}(\lambda, n, \ell)$: *The setup algorithm takes as input a security parameter λ and parameters $n, \ell \in \mathbb{N}$, and outputs a public parameter $\delta := (pk)$, a parameter $w := \left(\{\sigma_j\}_{j=1}^{\ell}, \{\mathsf{T}_i\}_{i=1}^{n}\right)$ where $pk \leftarrow \mathsf{KG}(1^{\lambda})$, $\sigma_j : (\mathbf{C}_{pk})^n \rightarrow \mathbf{C}_{pk}$, and $\mathsf{T}_i : (\mathbf{M}_{pk})^{\ell} \rightarrow \mathbf{M}_{pk}$.*
- $(\hat{\mathbf{c}}, \Gamma) \leftarrow \mathsf{Shuffle}(\delta, w, \mathbf{c})$: *The shuffle algorithm takes as input a pair of parameters (δ, w) and a list of ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$ where $c_i \in \mathsf{Enc}_{pk}(m_i)$, and outputs a set of ciphertexts $\hat{\mathbf{c}} = \{\hat{c}_1, \ldots, \hat{c}_{\ell}\}$ where $\hat{c}_j = \mathsf{ReRand}_{pk}\left(\sigma_j(c_1, \ldots, c_n), \hat{r}_j\right)$ along with a proof Γ, satisfying*

$$\mathsf{Dec}_{sk}(c_i) = \mathsf{T}_{i'}\left(\mathsf{Dec}_{sk}(\hat{c}_1), \ldots, \mathsf{Dec}_{sk}(\hat{c}_{\ell})\right)$$

for some $i, i' \in [1, n], j \in [1, \ell]$.

A generalized shuffle scheme is *correct* if for all messages $m_i \in \mathbf{M}_{pk}$ and any $n, \ell \in \mathbb{N}$, there exists each transformation $\mathsf{T}_i : (\mathbf{M}_{pk})^{\ell} \rightarrow \mathbf{M}_{pk}$ such that

$$\begin{aligned} &\{\mathsf{T}_1\left(\mathsf{Dec}_{sk}(\hat{c}_1), \ldots, \mathsf{Dec}_{sk}(\hat{c}_{\ell})\right), \ldots, \mathsf{T}_n\left(\mathsf{Dec}_{sk}(\hat{c}_1), \ldots, \mathsf{Dec}_{sk}(\hat{c}_{\ell})\right)\} \\ &\quad = \{m_1, \ldots, m_n\} \\ &\quad = \{\mathsf{Dec}_{sk}(c_1), \ldots, \mathsf{Dec}_{sk}(c_n)\}. \end{aligned} \tag{2.2}$$

In the above definition, if we choose functions σ_j's and transformations T_i's such that $\{\sigma_1, \ldots, \sigma_n\}$ and $\{\mathsf{T}_1, \ldots, \mathsf{T}_n\}$ are the set of all projection maps corresponding to random permutation π, then we obtain a standard shuffle defined in Definition 5. Note that in this case, $\ell = n$ and w is a secret parameter.

Definition 7 (Generalized Public Shuffle) *A generalized shuffle scheme $\tilde{\Phi}_{\mathcal{E}}$ is* public *if the parameter w is public.*

2.2 Security Model

In this section we give the security definition for generalized shuffle. Before describing the formal definition we identify which classes of entities participate in a given shuffle scheme, which will be given in Appendix A. Our security definition for shuffle begins with the definition given by [28], but we need to slightly modify theirs since some parameters could be public or secret in generalized shuffle.

Security Definition. As mentioned in [28], one of the primary requirements for being secure is verifiability and the other is unlinkability. Roughly speaking, verifiability means that a malicious shuffler cannot produce an incorrect output without detection by verifiers. What means that a shuffle scheme is unlinkable is that it is hard to find a permutation from input ciphertexts and output ciphertexts.

In this paper the adversary is PPT bounded and can be either semi-honest or malicious. A semi-honest party is assumed to follow the protocol exactly as what is prescribed by the protocol, except that it analyzes the records of intermediate computations. On the other hand, a malicious party can arbitrarily deviate from the protocol. However, we will not consider preventing those malicious behaviors such as independently and arbitrarily selecting inputs from the message space, and quitting the protocol at any step.

Verifiability. For a generalized shuffle scheme, we first modify the shuffle relation described in Eq. (2.1). A generalized shuffle relation $\tilde{\Phi}(x, w)$ is satisfied if the witness $w = (s_1, \ldots, s_\ell)$ demonstrates that

$$\exists\ (s_1, \ldots, s_\ell)\,, \forall\, j \in [1, \ell] : \hat{c}_j = \mathsf{ReRand}_{pk}\left(\sigma_j(c_1, \ldots, c_n), s_j\right). \tag{2.3}$$

The completeness condition of a proof system requires that for all $x = (\delta, \mathbf{c}, \hat{\mathbf{c}}) \in \mathcal{L}_{\tilde{\Phi}}$, the verification algorithm $\mathcal{V}$ of the proof system always accept. The soundness condition requires that if $x \notin \mathcal{L}_{\tilde{\Phi}}$, then $\mathcal{V}$ rejects with overwhelming probability. Verifiability is formally rephrased in Appendix A.

Recall that our eventual goal is to construct a public shuffle scheme. According to our definition, the public shuffle scheme makes its shuffle algorithm run without any secret information. What this means is that we need to use a different technique from zero-knowledge proofs for checking whether a shuffler works correctly. Indeed it can be easily done by re-computing what the shuffler computed only using public values. Let denote ϵ the empty string. We define a public shuffle relation $\tilde{\Phi}_{\mathsf{Pub}}(x, w)$ with the witness $w = \epsilon$ that holds if

$$\exists(\sigma_1, \ldots, \sigma_\ell), \forall j \in [1, \ell] : \hat{c}_j = \sigma_j(c_1, \ldots, c_n) \tag{2.4}$$

where $x = (\delta, \mathbf{c}, \hat{\mathbf{c}})$ and $\delta, \mathbf{c}$ with $\hat{\mathbf{c}}$ defined as in Definition 6. Since $\sigma_{j\in[1,\ell]}$ is a public n-argument function, any verifier is able to check whether a public shuffler is cheating or not. It is straightforward to define completeness and soundness of a proof system for a public shuffle relation with associated language $\mathcal{L}_{\tilde{\Phi}_{\mathsf{Pub}}}$.

Unlinkability. In order to show that a verifiable *secret* shuffle is unlikable, Nguyen et al. [28] proposed two security models: Chosen Permutation Attack (CPA_Σ) and Chosen Transcript Attack (CTA_Σ). The CPA_Σ security condition requires that even though the adversary $\mathcal{A}$ chooses two permutations of his choice, it should not distinguish which permutation was used to produce an output list of ciphertexts, with non-negligible advantage. On the other hand, the CTA_Σ security notion states that although the adversary can query an inversion oracle on $(\mathbf{c}, \hat{\mathbf{c}})$, which will give $\mathcal{A}$ a permutation π such that $\hat{c}_{\pi(i)} = \mathsf{ReRand}(c_i, \cdot)$ for all $i \in [1, |\mathbf{c}|]$, it should not have non-negligible advantage in guessing which of the two permutations in its challenge was used. The unlinkability security experiment by Nguyen et al. [28] is shown in Appendix A.

In Definition 6, w can be public or secret. That is, a generalized shuffle does not always take the information about permutation as a secret parameter. So we cannot directly apply the Nguyen et al.'s

model to prove the unlinkability security of generalized shuffles. Recall that even if a generalized shuffle scheme requires only a list of randomness in its definition as a secret parameter, it is a secret shuffle. We need a new one, but this is not very much different from the Nguyen et al.'s model. For completeness, we provide the security model for unlinkability of generalized secret shuffles in Appendix A.

Now we consider the case that a generalized shuffle scheme does not require even a list of randomness, i.e., during shuffling a shuffler does not use any secret information. We see that we cannot rely on the Nguyen et al.'s model at all. Instead we define a specific security experiment for generalized *public* shuffles.

Definition 8 (Unlinkability for Generalized Public Shuffle) *Let $\tilde{\Phi}_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ be a generalized public shuffle scheme and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda)$
$(\delta, w) \leftarrow \mathsf{Setup}(\lambda, n, \ell)$;
$(\mathsf{state}, \pi_0, \pi_1, \mathbf{m}) \leftarrow \mathcal{A}_1^{\mathcal{O}_{\mathsf{D}}}(\delta, n, \ell)$ *where* $\pi_i \in \Sigma_n, i \in \{0,1\}$ *and* $\mathbf{m} = (m_1, \ldots, m_n)$;
$(\hat{\mathbf{c}}, \Gamma) \leftarrow \mathsf{Shuffle}(\delta, w, \mathbf{c})$ *where* $c_i = \mathsf{Enc}_{pk}\left(m_{\pi_b(i)}, r_i\right)$ *with* $b \xleftarrow{\$} \{0,1\}$;
$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{D}}}(\hat{\mathbf{c}}, \mathbf{c}, w, \mathsf{state})$;
where $\mathcal{O}_{\mathsf{D}}$ is the decryption oracle.

In the experiment above, $\mathcal{A}_2$ is not permitted make the query $\mathcal{O}_{\mathsf{D}}(c_i)$ for all $c_{i\in[1,n]} \in \mathbf{c}$. We define the advantage of an adversary $\mathcal{A}$, running in probabilistic polynomial time and making a polynomial number of queries, as:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

A generalized public shuffle scheme is unlikable *if the advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{PubShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda)$ is negligible in the security parameter λ.*

2.3 Cryptographic Assumption

Let $\mathbb{G}_q$ be a cyclic group of order q, not necessarily prime, with a generator g. Given an algorithm $\mathcal{D}$, that takes as input quadruples of group elements and outputs a bit, the DDH-advantage of $\mathcal{D}$ with a generator g is defined as

$$\mathbf{Adv}_{\mathcal{D},g}^{\mathsf{ddh}}(\lambda) := \left| \Pr\left[\alpha, \beta \xleftarrow{\$} \mathbb{Z}_q : \mathcal{D}(g, g^{\alpha}, g^{\beta}, g^{\alpha\beta}) = 1\right] - \Pr\left[\alpha, \beta, \gamma \xleftarrow{\$} \mathbb{Z}_q : \mathcal{D}(g, g^{\alpha}, g^{\beta}, g^{\gamma}) = 1\right] \right|.$$

If $\mathbf{Adv}_{\mathcal{D},g}^{\mathsf{ddh}}$ is negligible for any polynomial time adversary $\mathcal{D}$ and any generator g, we say that the DDH assumption holds for $\mathbb{G}_q$.

We consider a group $\mathbb{G}_q$ where DDH problem is hard. It induces a subgroup of order q in the group of modular residues $\mathbb{Z}_p^{\times}$ such that $q|(p-1)$, $\|p\| = 2048$, $\|q\| = 256$ and a group of points on an elliptic curve with order q for $\|q\| = 256$. For more examples of groups, refer to [9].

3 Instructive Constructions

In this section we provide a generalized public shuffle using a *ring homomorphic* cryptosystem. That is, the shuffle schemes work correctly without a secret parameter such as a private permutation. Let us denote (ρ, η)-$\mathcal{E}$ a ring homomorphic cryptosystem supports ρ additions and η multiplications on encrypted data.

Construction 1. The basic intuition of our first generalized shuffle scheme is as follows: Let $n_1 = \binom{n}{\lfloor n/2 \rfloor}$ and $n_2 = n - 1$. Consider a semantically secure cryptosystem (n_1, n_2)-$\mathcal{E}$ on a Unique Factorization Domain (UFD) R, which allows re-randomization. Each message m_i is encrypted into $c_i \in \mathsf{Enc}_{pk}^{(n_1,n_2)}(m_i)$ by each sender S_i for $1 \le i \le n$. After receiving all the c_i's from each sender, a shuffler computes $\hat{c}_k = \sigma_k(c_1, \ldots, c_n) \in \mathsf{Enc}_{pk}^{(n_1,n_2)}(\sigma_k(m_1, \ldots, m_n))$ where σ_k is the k-elementary symmetric polynomial with

$$\sigma_k(x_1, \ldots, x_n) = \sum_{1 \le i_1 < \ldots < i_k \le n} x_{i_1} \cdots x_{i_k},$$

for each $k \in [1, \ell]$. Since the underlying encryption is a ring homomorphism, the shuffler can carry out such computations over ciphertexts.

Let us check that this scheme is correct. Decrypting an ℓ-tuple ciphertext $\{\hat{c}_1, \ldots, \hat{c}_\ell\}$ received from the shuffle protocol, any party who holds the private key sk learns all the coefficients of $F(t) = \prod_{i=1}^{n}(t - m_i) \in R[t]$. Since $R[t]$ is also a UFD, $F(t)$ is uniquely factorized into irreducibles $(t - m_i)$. For example, such a computation clearly runs in polynomial time in $\log p$ on $R = \mathbb{F}_p$. Since a factorization algorithm outputs the same result on inputs $F(t)$ and $F_\pi(t) = \prod_{i=1}^{n}\left(t - m_{\pi(i)}\right)$ for any permutation π of n elements, by the Definition 6 $\hat{c}_1, \ldots, \hat{c}_\ell$ can be regarded as a generalized shuffle of $c_1, \ldots, c_n$.

As a careful reader will observe, another generalized shuffle can be constructed from $(1, n)$-$\mathcal{E}$. The full description of this construction is given in Appendix B.

4 Main Constructions

The constructions presented in the previous section require the use of a ring homomorphic encryption scheme, which currently may not be practical, but apparently would be an overkill for applications such as shuffle. In this section we show how to construct generalized public shuffle schemes using an encryption scheme with only a group homomorphism, specifically ElGamal encryption [16] (due to technical reason, in fact we will use Damgård ElGamal [15]). The first generalized shuffle scheme extensively uses ElGamal encryption over extension fields. The other shuffle scheme is based on ElGamal encryption on prime fields, so it is more intuitive than the former but has a restriction on the size of input messages.

4.1 Building Blocks

We present some building blocks used to construct generalized public shuffle schemes.

ElGamal Encryption over $\mathbb{F}_{p^3}$. An ElGamal encryption scheme over $\mathbb{F}_{p^3}$ consists of the following three polynomial time algorithms $(\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{KG}(1^\lambda)$: The key generation algorithm chooses a large prime p such that $(p^3 - 1) = (p-1)(p^2+p+1) = 2q_1q_2$ for large primes q_1, q_2. Then select an irreducible polynomial $\wp(t) \in \mathbb{F}_p[t]$ of degree 3 and a generator $g(t)$ from $\mathbb{G}_{q_1q_2}$ which is a multiplicative subgroup of $\mathbb{F}_{p^3}^\times$ of order q_1q_2. It computes $y(t) = g(t)^x \bmod \wp(t)$ where a secret key x is randomly chosen from $[0, p^3 - 2]$, and publishes a public key $pk = \langle p, \mathbb{G}_{q_1q_2}, g(t), y(t), \wp(t)\rangle$.
- $\mathsf{Enc}_{pk}(m(t))$: Encryption with the public key pk and message $m(t) \in \mathbb{G}_{q_1q_2}$ proceeds as follows. First, a random value $r \in [0, p^3 - 2]$ is chosen. The ciphertext is then published as:

$$C(t) = (v(t), u(t)) := \left(g(t)^r \bmod \wp(t), m(t) \cdot y(t)^r \bmod \wp(t)\right).$$

- $\mathsf{Dec}_{sk}(C(t))$: Suppose that a ciphertext $C(t) = (v(t), u(t))$ is encrypted with a public key pk and we have a secret key x. Then the ciphertext can be decrypted as:

$$m(t) \equiv u(t) \cdot v(t)^{-x} \bmod \wp(t).$$

Parameter Generation. First, we check whether there exists a large prime p such that $p^3 - 1 = (p-1)(p^2+p+1)$, and $p = 2q_1+1$ and a prime $q_2 = p^2+p+1$. Assuming the Bateman-Horn conjecture [4,5], the number of primes of the form $(p^d-1)/(p-1) = \psi_d(p)$ not exceeding t, denoted by $H(t)$, is given by

$$H(t) \sim c \int_2^{t^{1/2}} (\log u)^{-2} du$$

for a constant $c \approx 2$ where $\psi_d(p)$ is the d-th cyclotomic polynomial. Therefore, we see that the probability that $\psi_d(p)$ is prime for an integer $p \ll t$ is significant.

In addition, we need to choose a sufficiently large prime p to resist against the index-calculus attack. In order to obtain the ElGamal encryption scheme with semantic security, we take two subgroups $\mathbb{G}_{q_1}$ and $\mathbb{G}_{q_2}$ as follows:

$$\mathbb{G}_{q_1} = \{a(t)^{2q_2} : a(t) \in (\mathbb{F}_p[t]/\wp(t))^\times\} \text{ and } \mathbb{G}_{q_2} = \{a(t)^{2q_1} : a(t) \in (\mathbb{F}_p[t]/\wp(t))^\times\}.$$

In particular, we set a generator $g = g_1 g_2$ of $\mathbb{G}_{q_1 q_2}$ such that $\langle g_1 \rangle = \mathbb{G}_{q_1}$ and $\langle g_2 \rangle = \mathbb{G}_{q_2}$.

Security Analysis. Now we verify whether the DDH assumption holds in $\mathbb{G}_{q_1 q_2}$.

Lemma 1 *Let $\mathbb{G}_{q_1}$ and $\mathbb{G}_{q_2}$ be groups of prime order q_1, q_2, respectively, where $\gcd(q_1, q_2) = 1$. Suppose that the DDH assumption holds in $\mathbb{G}_{q_1}$ and $\mathbb{G}_{q_2}$. Then the DDH assumption holds in the group $\mathbb{G}_{q_1 q_2}$.*

Proof. Suppose that there exists an algorithm $\mathcal{D}$ and a generator $g_0 \in \mathbb{G}_{q_1 q_2}$ such that $\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{D},g_0}$ is not negligible. We want to show that there exists an algorithm $\mathcal{D}'$ and generator $g_1 \in \mathbb{G}_{q_1}$ such that $\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{D}',g_1}$ is not negligible. Choose $g_1 := g_0^{q_2}$ and suppose that we are given a quadruple $(g_1, g_1^a, g_1^b, g_1^c)$. Then we submit the quadruple $(g_0, g_1^a, g_1^b, (g_1^c)^{q_2})$ to $\mathcal{D}$. If the output of $\mathcal{D}$ is 1, then $ab \equiv c \bmod q_1$. A similar argument holds for $\mathbb{G}_{q_2}$. □

Message Encoding. Since a message $m \in \{0,1\}^*$ or $m \in \mathbb{F}_p$ in general, we need to give a way to encode the message into a message space of our ElGamal encryption. Without loss of generality, suppose that a message $m \in \mathbb{F}_p$. We write the message m by $m(t) := t - m$. We then encrypt $m(t)$ using the ElGmal encryption scheme over $\mathbb{F}_{p^3}$. As a result, to provide a natural encoding that embeds an input $m(t) \in \mathbb{F}_p[t]$ into $\mathbb{G}_{q_1 q_2}$, we should slightly modify the encryption algorithm $\mathsf{Enc}_{pk}(\cdot)$ as follows:

$$u(t) = m(t)^2 \cdot y(t)^r \bmod \wp(t),$$

while keeping $v(t)$ unchanged. We can easily check that the modified ElGamal encryption scheme with this message encoding is semantically secure under the DDH assumption in $\mathbb{G}_{q_1 q_2}$ by Lemma 1.

Keeping the Shuffler Honest without Zero-knowledge Proofs. One important property of our construction allows to prevent a shuffler from behavior maliciously without depending on zero-knowledge proofs (ZKPs). This gets rid of the expensive cost of computation and communication required for ZKPs mandatorily. For this purpose, a verifier only have to re-compute the shuffler's output using public values.

4.2 A Generalized Public Shuffle Scheme Based on Polynomial Factorization

Since the decryption oracle is given to adversary in the definition 8, we need CCA1-secure ElGamal encryption. One candidate is Damgård ElGamal which satisfies CCA1 security [15]. We consider a variant of Damgård ElGamal encryption over $\mathbb{F}_{p^3}$ given in Appendix F, which is also CCA1-secure in a cyclic subgroup of $\mathbb{F}_{p^3}^\times$.

Extended Damgård ElGamal Encryption. Turning our basic idea into constructing a generalized public shuffle scheme requires that we modify the basic Damgård ElGamal (DEG) encryption over $\mathbb{F}_{p^3}$. We just describe modifications for extended DEG encryption over $\mathbb{F}_{p^3}$, denoted by $\mathcal{E}^{\mathsf{d}} = (\mathsf{KG}^{\mathsf{d}}, \mathsf{Enc}^{\mathsf{d}}, \mathsf{Dec}^{\mathsf{d}})$. According to modified parameters, its encryption and decryption algorithms should be modified as follows:

- *Modifying Key Generation.* We run $\mathsf{KG}^{\mathsf{d}}(1^\lambda)$ as in the basic scheme. Further, choose ℓ irreducible polynomials $\wp_1(t), \ldots, \wp_\ell(t) \in \mathbb{F}_p[t]$ of degree 3. Find a field isomorphism $\phi_j : \mathbb{F}_p[t]/\wp(t) \to \mathbb{F}_p[t]/\wp_j(t)$ for $j \in [1, \ell]$. Finally compute $y_j = \phi_j(y)$ and $h_j = \phi_j(h)$ for $j \in [1, \ell]$, and publish $pk = (g, y, h, \{y_i\}_{i=1}^{\ell}, \{h_i\}_{i=1}^{\ell}, \mathbb{G}_{q_1 q_2}, \wp(t), \{\wp_i(t)\}_{i=1}^{\ell}, \{\phi_i\}_{i=1}^{\ell})$ and keep a secret key $sk = (x_1, x_2)$.
- *Modifying Encryption and Decryption Algorithms.* We define ℓ-tuple DEG encryption by extending DEG encryption over $\mathbb{F}_{p^3}$. Given a message $m(t) \in \mathbb{F}_p[t]$, its encryption algorithm $\ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(\cdot)$ is defined as follows:

  $$\ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m(t)) := (g^r, y_1^r, m(t)^2 \cdot h_1^r, \ldots, y_\ell^r, m(t)^2 \cdot h_\ell^r) \in \mathbb{F}_p[t]/\wp(t) \times (\mathbb{F}_p[t]/\wp_1(t))^2 \times \cdots \times (\mathbb{F}_p[t]/\wp_\ell(t))^2.$$

  For decryption, first compute $\phi_j(g^r)$ and $m(t)^2 \equiv (\phi_j(g^r))^{-x_2} \cdot m(t)^2 \cdot h_j^r \bmod \wp_j$. Then we get $m(t)^2 \pmod{\wp_1 \cdots \wp_\ell}$ using the Chinese remaindering algorithm (in short, CRT). After computing square root of the value, we get $m(t), -m(t) \pmod{\wp_1 \cdots \wp_\ell}$. Since $m(t)$ is linear, we can determine the original message $m(t)$ uniquely.

The Construction. We describe the generalized public shuffle using the ℓ-tuple DEG encryption scheme over extension fields.

$\mathsf{Setup}(\lambda, n, \ell)$**.** This algorithm takes as input a security parameter λ and size parameter n, ℓ. It outputs a description of $\sigma : (\mathbb{G}_{q_1 q_2})^n \to \mathbb{G}_{q_1 q_2}$ given by $(c_1, \ldots, c_n) \mapsto c_1 \cdots c_n$ and $\{\mathsf{T}_i\}_{i=1}^n$ as runnung the CRT, a square root finding algorithm, and a factorization in turn along with the public key pk, i.e., it outputs two public parameters $\delta = (pk)$ and $w = (\sigma, \{\mathsf{T}_i\}_{i=1}^n)$

$\mathsf{Shuffle}(\delta, w, \mathbf{c})$**.** Shuffling with the public parameter δ and a list of ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$ where c_i is an ℓ-tuple DEG ciphertext, given from each sender S_i, proceeds as follows. Here $c_i \in \ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m_i(t))$ and

$$\ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m_i(t)) = \left(g^{r_i}, y_1^{r_i}, m_i(t)^2 \cdot h_1^{r_i}, y_2^{r_i}, m_i(t)^2 \cdot h_2^{r_i}, \ldots, y_\ell^{r_i}, m_i(t)^2 \cdot h_\ell^{r_i}\right)$$

where $r_i \xleftarrow{\$} [0, p^3 - 2]$ for $1 \le i \le n$.

1. The shuffler computes $\prod_{i=1}^n \ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m_i(t))$ where the product of $\ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m_i(t))$ means coordinate-wise product. Namely,

$$\begin{aligned}\prod_{i=1}^{n} \ell\text{-}\mathsf{Enc}^{\mathsf{d}}_{pk}(m_i(t)) = \Big(&\sigma\left(g^{r_1}, \ldots, g^{r_n}\right), \sigma\left(y_1^{r_1}, \ldots, y_1^{r_n}\right), \sigma\left(m_1(t)^2 \cdot h_1^{r_1}, \ldots, m_n(t)^2 \cdot h_1^{r_n}\right), \ldots, \\ &\sigma\left(y_\ell^{r_1}, \ldots, y_\ell^{r_n}\right), \ \sigma\left(m_1(t)^2 \cdot h_\ell^{r_1}, \ldots, m_n(t)^2 \cdot h_\ell^{r_n}\right)\Big) \\ = &\left(g^{\sum_{i=1}^n r_i}, y_1^{\sum_{i=1}^n r_i}, (\prod_{i=1}^{n} m_i(t))^2 \cdot h_1^{\sum_{i=1}^n r_i}, \ldots, y_\ell^{\sum_{i=1}^n r_i}, (\prod_{i=1}^{n} m_i(t))^2 \cdot h_\ell^{\sum_{i=1}^n r_i}\right)\end{aligned}$$

   And for all $j \in [1, \ell]$ set

$$\hat{c}_j = \left(\phi_j\left(g^{\sum_{i=1}^n r_i}\right), y_j^{\sum_{i=1}^n r_i}, \left(\prod_{i=1}^{n} m_i(t)\right)^2 \cdot h_j^{\sum_{i=1}^n r_i}\right)$$

2. The shuffler outputs a list of ciphertexts $\hat{\mathbf{c}} = (\hat{c}_1, \ldots, \hat{c}_\ell)$ along with a proof $\Gamma = \epsilon$.

Verify$(\delta, \sigma, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$**.** Upon receiving this tuple, the verifier will first run the verification algorithm by non-interactively running $\mathcal{V}(\delta, \sigma, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$ – whether all $\hat{c}_j \in \hat{\mathbf{c}}$ were correctly computed by using $\mathbf{c}$ from senders and δ; if this fails abort and return reject. Otherwise, output accept.

Theorem 1 *If the shuffler performs correctly the scheme, our public shuffle scheme is correct.*

Proof. We know that each transformation T_i $(1 \leq i \leq n)$ as running the CRT, a square root finding algorithm, and a factorization algorithm in turn. The correctness of shuffle can be easily checked. We know that if one knows the secret key x, he decrypts

$$\left(\phi_j\left(g^{\sum_{i=1}^n r_i}\right), y_j^{\sum_{i=1}^n r_i}, \left(\prod_{i=1}^n m_i(t)\right)^2 \cdot h_j^{\sum_{i=1}^n r_i}\right)$$

to $(\prod_{i=1}^n m_i(t))^2 \bmod \wp_j(t)$, $1 \leq j \leq \ell$. He then computes $(\prod_{i=1}^n m_i(t))^2 \bmod \wp_1(t) \cdots \wp_\ell(t)$ from each $(\prod_{i=1}^n m_i(t))^2 \bmod \wp_j(t)$ by using a Chinese remainder algorithm. He obtains $\prod_{i=1}^n m_i(t)$ by solving square root of $(\prod_{i=1}^n m_i(t))^2$ over $\mathbb{F}_p[t]$, since $m(t)$ is monic. Finally a factorization algorithm outputs $\{m_1, \ldots, m_n\}$. □

According to our definitions, the next theorem proves that the generalized public shuffle satisfies unlinkability if the DDH assumption holds. The proof is presented in Appendix D.

Theorem 2 *Assuming the DDH assumption holds, our public shuffle scheme is unlinkable.*

Theorem 3 *Assuming the DDH assumption holds, our public shuffle scheme is verifiable.*

Proof. It follows from the fact that completeness and soundness conditions can be easily checked by the verifier's re-computation. □

Computational Complexity. Each sender encrypts his plaintext ℓ times with $O(\ell \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ complexity. The shuffler computes the product of encrypted data. It takes $O(n\ell)$ $\mathsf{M}_{\mathbb{F}_p}$. The shuffler computes isomorphism $\phi_j(g(t)^{\sum_{i=1}^n r_i})$, $1 \leq j \leq \ell$, with $O(1)$ $\mathsf{M}_{\mathbb{F}_p}$. The decryption requires $O(\ell \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ and $\prod_{i=1}^n (m_i(t))^2 \bmod \prod_{i=1}^\ell \wp_i(t)$ is obtained by using a fast CRT in $O(\ell \log \ell)$ $\mathsf{M}_{\mathbb{F}_p}$. Solving square root of $(\prod_{i=1}^n m_i(t))^2 \bmod \prod_{i=1}^\ell \wp_i(t)$ requires $O(\ell \log p)$ $\mathsf{M}_{\mathbb{F}_p}$, and factoring $\prod_{i=1}^n m_i(t)$ over $\mathbb{F}_p[t]$ incurs $O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$. Therefore, the total complexity amounts to $O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$.

Ciphertext Size. The number of ciphertexts each sender transmits is $O(\ell)$ and the shuffler takes as input $O(n\ell)$ ciphertexts and outputs $O(\ell)$ ciphertexts.

Keeping the Sender Honest. To prevent the sender himself from attempting to cheat the shuffle, we require that each sender should be prepared to give a zero-knowledge proof of the plaintext of his ciphertext. For example, given a DEG ciphertext $c = (u, v, w) = (g^r, y^r, mh^r)$ under the public key y, h, a sender prover knowledge of m by instead proving knowledge of r.

It is unlikely to detect all malicious behavior of dishonest senders during encoding and encrypting their messages. Instead we can deal with the case where a malicious sender replaces at most α positions with random values of his choice instead of all the same m_i's. When decrypting the output of the shuffle, after applying the CRT, we will run the extended Euclidean algorithm and apply the rational reconstruction theorem [34, Sec. 4.6]. If the number of malicious positions is at most α, we can efficiently recover the original value m_i from its malicious encoding. The polynomial analog takes the same approach [34, Sec. 17.5].

4.3 A Generalized Public Shuffle Scheme Based on Integer Factorization

In this section we present another public shuffle which can work correctly and efficiently, especially when each user has short messages enough to support recovering original messages in polynomial time. However, the intuition is the same as the public shuffle scheme given in Section 4.2.

This construction uses ℓ-tuple DEG encryption extended by standard Damgård ElGamal encryption over prime fields. We just describe the differences compared to that used in Section 4.2: (1) Since a field isomorphism is not available, each user should send ℓ full DEG ciphertexts to a shuffler. Namely, the number of group elements transmitted by each sender is 3ℓ. Recall that the previous construction allows a sender to send $(\ell+1)$ group elements; (2) For unique factorization over the integers, we should provide a specific encoding algorithm. For example, the encoding algorithm converts an input message into a prime number in a message space. The full description of ElGamal and its extension over prime fields are shown in Appendix C. From this we can easily obtain DEG encryption over prime field. If no confusion arises, we abuse notation and use the same symbol for extended DEG encryption.

The Construction. The following is the description of the generalized public shuffle using the ℓ-tuple DEG encryption scheme over prime fields.

$\mathsf{Setup}(1^\lambda, n, \ell)$**.** This algorithm takes as input a security parameter λ and size parameter n, ℓ. It outputs a description of σ given by $(c_1, \ldots, c_n) \mapsto c_1 \cdots c_n$ and $\{\mathsf{T}_i\}_{i=1}^n$ as runnung the CRT, a square root finding algorithm, and a factorization in turn along with the public key pk, i.e., it outputs two public parameters $\delta = (pk)$ and $w = (\sigma, \{\mathsf{T}_i\}_{i=1}^n)$.

$\mathsf{Shuffle}(\delta, w, \mathbf{c})$**.** Shuffling with the public parameter δ and a list of ciphertexts $\mathbf{c} = (c_1, \ldots, c_n)$ where $c_i \in \ell\text{-}\mathsf{Enc}_{pk}^{\mathsf{d}}(m_i)$ from a sender S_i, proceeds as follows. Here $\ell\text{-}\mathsf{Enc}_{pk}^{\mathsf{d}}(m_i) = \{(u_{ij}, v_{ij}, w_{ij})\}_{j=1}^{\ell}$ with $u_{ij} = g_j^{r_{ij}}, v_{ij} = y_j^{r_{ij}}, w_{ij} = m_i^2 \cdot h_j^{r_{ij}}$.

1. The shuffler computes and outputs

$$(\hat{c}_1, \ldots, \hat{c}_\ell) = \left(\left(\prod_{i=1}^n u_{i1}, \prod_{i=1}^n v_{i1}, \prod_{i=1}^n w_{i1} \right), \ldots, \left(\prod_{i=1}^n u_{i\ell}, \prod_{i=1}^n v_{i\ell}, \prod_{i=1}^n w_{i\ell} \right) \right)$$

with a proof $\Gamma = \epsilon$.

$\mathsf{Verify}(\delta, w, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$**.** The verification algorithm checks if each $\hat{c}_j \in \hat{\mathbf{c}}$ was correctly computed by using $\mathbf{c}$ and δ; if this fails abort and return reject. Otherwise, output accept.

Theorem 4 *If the shuffler performs correctly, our public shuffle scheme is correct.*

Proof. Suppose that the shuffler follows the above algorithm properly. We know that each transformation T_i $(1 \le i \le n)$ as running the CRT, a square root finding algorithm and a factorization algorithm in turn. Then the correctness of shuffle can be easily checked. Specifically, a decryption algorithm takes as input ℓ-tuple DEG ciphertexts $(\prod_{i=1}^n u_{ij}, \prod_{i=1}^n v_{ij}, \prod_{i=1}^n w_{ij})$ for $1 \le j \le \ell$, and outputs

$$\begin{aligned} M_1 &= (m_1 m_2 \cdots m_n)^2 \pmod{p_1} \\ &\vdots \\ M_n &= (m_1 m_2 \cdots m_n)^2 \pmod{p_\ell} \end{aligned}$$

$M = (m_1 m_2 \cdots m_n)^2 \bmod p_1 \cdots p_\ell$ is obtained by using the CRT. It computes square roots of M modular $p_1 \cdots p_\ell$, say $z_1 = m_1 \cdots m_n$ and $z_2 = -m_1 m_2 \cdots m_n$, respectively. Since all m_i's are odd prime numbers, the least significant bit (LSB) of z_1 is 1. On the other hand, the LSB of z_2 is 0 since $p_1 \cdots p_\ell - m_1 \cdots m_n = -m_1 \cdots m_n \pmod{p_1 \cdots p_\ell}$. Hence, it can uniquely determine which one is a correct product of $\{m_1, \ldots, m_n\}$. Finally it runs a factorization algorithm for $m_1 \cdots m_n$ over $\mathbb{Z}$ using trial division since m_i's are small. □

Further, with respect to unlinkability and public verifiability it is straightforward from a similar argument proved in the previous section.

Computational Complexity. Let us define $\check{p} = \max\{p_1, \ldots, p_\ell\}$, and $\hat{p} = \min\{p_1, \ldots, p_\ell\}$. Each sender encrypts his plaintext ℓ times with $O(\ell \log \check{p})$ $\mathsf{M}_{\mathbb{F}_p}$ complexity. The shuffler computes $(\prod_{i=1}^{n} u_{ij}, \prod_{i=1}^{n} v_{ij}, \prod_{i=1}^{n} w_{ij})$ for $1 \leq j \leq \ell$ with $O(\ell^2)$ $\mathsf{M}_{\mathbb{F}_p}$ complexity. Decryption requires $O(\ell \log \check{p})$ $\mathsf{M}_{\mathbb{F}_p}$ complexity, and computing the CRT requires $O(\mathsf{M}(\log \check{p}) \log\log \check{p})\mathsf{M}_{\mathbb{F}_p}$ to get $(m_1 \cdots m_n)^2 \mod p_1 \cdots p_\ell$. Solving square roots of $(m_1 \cdots m_n)^2 \mod p_1 \cdots p_\ell$ incurs $O(n \log^3 \check{p}))$ $\mathsf{M}_{\mathbb{F}_p}$. Since the message space is small, factorizing $m_1 \cdots m_n$ using trial division takes $O(n\bar{m} \log \check{p})$ when messages are taken to be a prime less than $\bar{m}$

Ciphertext Size. The number of ciphertexts each user sends is $O(\ell)$ and the shuffler takes as input $O(n\ell)$ ciphertexts and outputs $O(\ell)$ ciphertexts.

Remark 1 *If the message space is small, the shuffle algorithm may output $(\hat{c}_1, \ldots, \hat{c}_\ell)$ for $\ell < n$. This reduces the computation and transmission cost. Suppose each message is encoded into a prime of κ bits. Decrypting $(\hat{c}_1, \ldots, \hat{c}_\ell)$ gives $(m_1 \cdots m_n)^2 \bmod p_1 \cdots p_\ell$. One can recover an integer $m_1 \cdots m_n$ when $2n\kappa < \ell \|p\|$, i.e. $\ell > (2n\kappa)/\|p\|$.*

For example, consider $\kappa = 10$, $n = 10^4$ and $\|p\| = 2048$. Then it is enough to take $\ell = 98$, which is much less than $n = 10^4$.

5 Further Discussions

We studied how to construct a public shuffle, which does not require any private setup for generating a random permutation. For this purpose, we proposed two constructions. Our constructions use CCA1 ElGamal encryption schemes,but one is based on integer factorization which requires exponential complexity in general, the other is based on polynomial factorization. Further, we exploit a field isomorphism to reduce the size of ciphertexts.

However, still there are two remaining open problems. The first one is that our schemes let each sender transmit $O(n)$ ciphertexts to a shuffler. Therefore, the total transmission complexity is $O(n^2)$. Thus, how to construct a public shuffle scheme with $O(n)$ transmission complexity in total is an interesting problem. The second one is to apply our technique to Adida and Wikström's work. Namely, how to generate an obfuscated permutation matrix by using our scheme is also an interesting question.

References

1. M. Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In K. Nyberg, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1403, pages 437–447, 1998.
2. M. Abe. Mix-networks on permutation networks. In K.-Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology-AsiaCrypt*, LNCS 1716, pages 258–273, 1999.
3. B. Adida and D. Wikström. How to shuffle in public. In S. Vadhan, editor, *TCC*, LNCS 4392, pages 555–574, 2007.
4. P. Bateman and R. Horn. A heuristic asymptotic formula concerning the distribution of prime numbers. *Mathematics of Computation*, 16:363–367, 1962.
5. P. Bateman and R. Stemmler. Waring's problem for algebraic number fields and primes of the form $(p^r - 1)/(p^d - 1)$. *Illinois J. Math.*, 6(1):142–156, 1962.
6. O. Baudron, P. Fouque, D. Pointcheval, J. Stern, and G. Poupard. Practical multi-candidate election system. In *PODC*, pages 274– 283, 2001.
7. S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In D. Pointcheval and T. Johansson, editors, *Advances in Cryptology-EuroCrypt*, LNCS 7237, pages 263–280, 2012.
8. J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *PODC*, pages 52–62, 1986.
9. D. Boneh. The decision Diffie-Hellman problem. In J. Buhler, editor, *ANTS*, LNCS 1423, pages 48–63, 1998.

10. D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *TCC*, LNCS 3378, pages 325–341, 2005.
11. D. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.
12. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.
13. R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In U. Maurer, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1070, pages 72–83, 1996.
14. R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In W. Fumy, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1233, pages 481–490, 1997.
15. I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In J. Feigenbaum, editor, *Advances in Cryptology-Crypto*, LNCS 576, pages 445–456, 1991.
16. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakely and D. Chaum, editors, *Advances in Cryptology-Crypto*, LNCS 196, pages 10–18, 1984.
17. J. Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE transactions*, 88-A(1):172–188, 2005.
18. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *Advances in Cryptology-Crypto*, LNCS 2139, pages 368–387, 2001.
19. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
20. J. Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.
21. J. Groth and S. Lu. Verifiable shuffle of large size ciphertexts. In T. Okamoto and X. Wang, editors, *Public Key Cryptography*, LNCS 4450, pages 377–392, 2007.
22. C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. In P. Q. Nguyen and D. Pointcheval, editors, *Public Key Cryptography*, LNCS 6056, pages 312–331, 2010.
23. M. Huxley. On the difference between consecutive primes. *Inventiones Math.*, 15:164–170, 1972.
24. L. Kissner and D. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology-Crypto*, LNCS 3621, pages 241–257, 2005.
25. H. Maier. Primes in short intervals. *Michigan Mathematical Journal*, 32(2):221–225, 1985.
26. C. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001.
27. C. Neff. Verifiable mixing (shuffling) of ElGamal pairs, 2003.
28. L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS*, LNCS 3089, pages 61–75, 2004.
29. R. Ostrovsky and W. Skeith III. Private searching on streaming data. In V. Shoup, editor, *Advances in Cryptology-Crypto*, LNCS 3621, pages 223–240, 2005.
30. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1592, pages 223–238, 1999.
31. U. Parampalli, K. Ramchen, and V. Teague. Efficiently shuffling in public. To appear in PKC'12, 2012.
32. C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In T. Helleseth, editor, *Advances in Cryptology-EuroCrypt*, LNCS 765, pages 248–259, 1993.
33. K. Sako and J. Kilian. Receipt-free mix-type voting scheme – a practical solution to the implementation of a voting booth. In L. Guillou and J.-J. Quisquater, editors, *Advances in Cryptology-EuroCrypt*, LNCS 921, pages 393–403, 1995.
34. V. Shoup. *A conputational introduction to number theory and algebra*. Cambridge University Press, 2nd edition, 2009.
35. D. Wikström. A sender verifiable mix-net and a new proof of a shuffle. In B. Roy, editor, *Advances in Cryptology-AsiaCrypt*, LNCS 3788, pages 273–292, 2005.
36. D. Wikström. A commitment-consistent proof of a shuffle. In C. Boyd and J. M. G. Nieto, editors, *ACISP*, LNCS 5594, pages 407–421, 2009.

A Basic Definitions

A.1 Participating Parties.

In our description, we use a few classes of entities which participate in shuffle.

- *Senders*. There are an arbitrary number of senders participating in a shuffle scheme (we will denote the number of senders by n). Each sender has a secret input.
- *Shuffler*. A shuffler receives the n ciphertexts of all the senders and outputs the ℓ ciphertexts as a result of shuffle.

– *Verifier*. A verifier is a party that verifies that the shuffler correctly follows the shuffle scheme. Although there can be many verifiers (and senders can be verifiers as well) the verifiers are deterministic and use only public information, so we model them as a single party.
– *Adversary*. The adversary attempts to subvert a shuffle scheme. We detail the adversarial model in the later.

A.2 Verifiability for Secret Shuffles

We rephrase the verifiability condition for secret shuffles in our language. The reader is encouraged to refer to [28] for in-depth discussions on the verifiability condition of shuffles.

Definition 9 ([28]) *Let a set of algorithms $(\mathcal{P}, \mathcal{V})$ be a proof system for an efficient generalized shuffle relation $\tilde{\Phi}$ with associated language $\mathcal{L}_{\tilde{\Phi}}$. A generalized shuffle scheme $\tilde{\Phi}_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ is verifiable if its proof system $(\mathcal{P}, \mathcal{V})$ has an efficient algorithm $\mathcal{V}$ and satisfies completeness and soundness below.*

1. *Completeness. For all $x = (\delta, \mathbf{c}, \hat{\mathbf{c}}) \in \mathcal{L}_{\tilde{\Phi}}$, $(\mathcal{P}, \mathcal{V})(x, \Gamma) = 1$ for all proofs $\Gamma \leftarrow \mathcal{P}(x, w)$ where $\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$.*
2. *Soundness. For all PPT $\mathcal{A}$ and for $\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$, the probability that $\mathcal{A}(\lambda, n, \ell, \delta)$ outputs (x, Γ) such that $x \notin \mathcal{L}_{\tilde{\Phi}}$ but $(\mathcal{A}, \mathcal{V})(x, \Gamma) = 1$, is negligible in the security parameter λ.*

A.3 Unlinkability Experiments

One definition for security of a secret shuffle $\Phi_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ is indistinguishability against chosen permutation attack (CPA_{Σ}), which is analogous to indistinguishability against chosen plaintext attack in public-key cryptosystems [28]. Nguyen et al. [28] proposed a different definition called semantic privacy against CPA_{Σ}, but they showed that the two notions are eventually equivalent.

For a proof system, we use $\mathsf{View}^{\mathcal{P},\mathcal{V}}(x)$ to denote all that $\mathcal{V}$ can see from the execution of the proof system on input x.

Definition 10 (Unlinkability in [28]) *Let $\Phi_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ be a secret shuffle scheme.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Shuffle}}(\Phi_{\mathcal{E}}, \lambda)$
$\delta \leftarrow \mathsf{Setup}(\lambda, n)$;
$(\pi_0, \pi_1, \mathbf{c}) \leftarrow \mathcal{A}(\delta, n)$ *where* $\pi_i \in \Sigma_n$ *for* $i = 1, 2$;
$(\hat{\mathbf{c}}, \Gamma) \leftarrow \mathsf{Shuffle}(\delta, w_b, \mathbf{c})$ *where* $w_b \xleftarrow{\$} \{\pi_0, \pi_1\}$;
$\nu \leftarrow \left(\hat{\mathbf{c}}, \mathsf{View}^{\mathcal{P},\mathcal{V}}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma), \mathbf{c}, \{m_i\}_{i=1}^n, \{r_i\}_{i=1}^n\right)$ *where* $c_i = \mathsf{Enc}_{pk}(m_i, r_i)$;
$b' \leftarrow \mathcal{A}(\delta, \nu)$;

In the experiment above, we define the advantage of an adversary $\mathcal{A}$, running in probabilistic polynomial time and making a polynomial number of queries, as:

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Shuffle}}(\Phi_{\mathcal{E}}, \lambda) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

A verifiable secret shuffle scheme is unlikable *if*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Shuffle}}(\Phi_{\mathcal{E}}, \lambda) \leq \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\cdot)$ *is a negligible function of its input.*

For a generalized secret shuffle, we describe a variant of the unlinkability notion against the *chosen random attack*.

Definition 11 (Unlinkability for a Generalized Secret Shuffle) *Let* $\tilde{\Phi}_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ *be a generalized secret shuffle scheme.*

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{GenShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda)$
$\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$*;*
$(\mathbf{r}_0, \mathbf{r}_1, \mathbf{c}) \leftarrow \mathcal{A}(\delta, n, \ell)$ *where* $\mathbf{r}_i = (r_{i1}, \ldots, r_{i\ell})$ *for* $i = 1, 2$*;*
$(\hat{\mathbf{c}}, \Gamma) \leftarrow \mathsf{Shuffle}(\delta, w_b, \mathbf{c})$ *where* $w_b \xleftarrow{\$} \{\mathbf{r}_0, \mathbf{r}_1\}$*;*
$\nu \leftarrow \left(\hat{\mathbf{c}}, \mathsf{View}^{\mathcal{P},\mathcal{V}}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma), \mathbf{c}, \{m_i\}_{i=1}^n, \{r_i\}_{i=1}^n\right)$ *where* $c_i = \mathsf{Enc}_{pk}(m_i, r_i)$*;*
$b' \leftarrow \mathcal{A}(\delta, \nu)$*;*

In the experiment above, we define the advantage of an adversary $\mathcal{A}$*, running in probabilistic polynomial time and making a polynomial number of queries, as:*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{GenShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = \left| \Pr[b = b'] - \frac{1}{2} \right|.$$

A generalize secret shuffle scheme is unlikable *if the advantage* $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{GenShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda)$ *is negligible in the security parameter* λ*.*

B Additional Constructions

We base another generalized shuffle scheme on $(1, n)$-$\mathcal{E}$ that is a ring homomorphic cryptosystem that supports 1 addition and n multiplications on ciphertexts, and re-randomization. In this construction, the intuition is that a shuffler first publishes all $\hat{c}_j \in \mathsf{Enc}_{pk}^{(1,n)}(B(\alpha_j))$, $1 \le j \le \ell = n$ for $B(t) = \prod_{i=1}^n (t + m_i)$ where α_j's are chosen uniformly at random from a random space. After decrypting properly, $B(t)$ is recovered through Lagrange interpolation and then factorized into each linear term as above.

Lemma 2 *Assuming that there exists a ring homomorphic cryptosystem* $(1, n)$-$\mathcal{E}$ *that meets the conditions required in the construction above, our generalized shuffle scheme based on* $(1, n)$-$\mathcal{E}$ *is correct.*

Proof. . Suppose that the shuffler follows the above algorithm properly. If one takes each transformation T_i $(1 \le i \le n)$ as running a polynomial reconstruct algorithm and a factorization algorithm in turn, then he can easily see that the correctness condition – Eq. (2.2) holds.

More specifically, anyone who can decrypt takes as input $(\hat{c}_1, \ldots, \hat{c}_n)$, and outputs $\prod_{i=1}^n (\alpha_j + m_i)$ for each $j \in [1, n]$. Then he reconstructs a polynomial $B(t) = \prod_{i=1}^n (t + m_i)$ using the Lagrange interpolation as follows:

$$B(t) = \sum_{j=1}^{n} B(\alpha_j) \prod_{1 \le i \le n, i \ne j} \frac{t - \alpha_i}{\alpha_j - \alpha_i}.$$

Finally $\{m_1, \ldots, m_n\}$ can be recovered by using a factorization algorithm over $R[t]$. □

Computational Complexity. Denote by E and D the cost of an encryption algorithm and a decryption algorithm for an underlying cryptosystem, respectively. M_D denotes the cost of multiplication in a domain D. Additionally, $\mathsf{M}(d)$ denotes the cost of multiplication of two d-bit integers, and $\mathsf{M}(d, p)$ the cost of multiplication of two polynomials of degree d over $\mathbb{F}_p$.

Each sender only has to encrypt his message once. The shuffler computes $\mathsf{Enc}_{pk}^{(1,n)}(\alpha_j)$, $1 \le j \le n$. The shuffler should compute $\prod_{i=1}^n \mathsf{Enc}_{pk}^{(1,n)}(\alpha_j + m_i)$ for each $j \in [1, n]$, whose complexity is $n\,\mathsf{E}$ and $n(n-1)\,\mathsf{M}_{\mathbb{F}_p}$, if $\mathbf{C}_{pk} = \mathbb{F}_p$. In summary, the total complexity amounts to $O(n)(\mathsf{E}) + O(n^2)\,\mathsf{M}_{\mathbb{F}_p}$, on $R = \mathbb{F}_p$.

Ciphertext Size. The number of ciphertexts each sender sends is 1. The shuffler takes as input n ciphertexts and outputs n another ciphertexts.

Remark 2 *For completeness we present the total complexity including a process recovering input plaintexts. Anyone who is authorized to decrypt should decrypt $\hat{c}_1, \ldots, \hat{c}_n$ and reconstruct the polynomial $B(x)$ of degree n with complexity n $\mathsf{D} + O(n^2)$ $\mathsf{M}_{\mathbb{F}_p}$. Further, this incurs $O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ to factorize using Cantor-Zassenhaus algorithm [11], if $R = \mathbb{F}_p$. Hence, the total complexity amounts to $O(n)(\mathsf{E} + \mathsf{D}) + O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$, on $R = \mathbb{F}_p$.*

C ElGamal and Its Extension over Prime Fields

The description of the ElGamal encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ over prime fields consists of the following algorithms. Let $\mathbb{F}_p$ be a prime field and $\mathbb{G}_q$ be a multiplicative cyclic subgroup of order q in $\mathbb{F}_p^\times$, where $p = 2q + 1$. Assume that the DDH assumption holds in $\mathbb{G}_q$.

- $\mathsf{KG}(1^\lambda)$. Choose a generator g of $\mathbb{G}_q$. Choose a random $x \in [0, q-2]$ and compute $y = g^x \pmod p$. A public key is $pk = (p, g, y, \mathbb{G}_q)$ and a secret key is $sk = x$.
- $\mathsf{Enc}_{pk}(m)$. Choose random $r \in [0, q-2]$ and compute g^r and $m \cdot y^r$. The ciphertext of $m \in \mathbb{G}_q$ is given by $(v, u) = (g^r, m \cdot y^r)$.
- $\mathsf{Dec}_{sk}(v, u)$. Compute $m = v^{-x}u \pmod p$.

If the input message $m \in \mathbb{G}_q$, then the encryption algorithm simply continues to the next step. However, if $m \notin \mathbb{G}_q$, it is required to convert m into an element of the group. Thus, we need to modify its encryption algorithm into computing $u = m^2 \cdot y^r \pmod p$. Also we define ℓ-tuple ElGamal encryption as its extension. That is, for $m \in \mathbf{M}_{pk}$

$$\ell\text{-}\mathsf{Enc}_{pk}(m) = (g_1^r, m^2 y_1^r, g_2^r, m^2 y_2^r, \ldots, g_\ell^r, m^2 y_\ell^r) \in \mathbb{F}_{p_1}^2 \times \mathbb{F}_{p_2}^2 \times \cdots \times \mathbb{F}_{p_\ell}^2,$$

where $p_1 < p_2 < \cdots < p_\ell$ are add primes and $y_j = g_j^x$ for all $j \in [1, \ell]$.

Actually, since we use factorization to get message, message space must be prime set which is smaller than p_1. Instead we use encoding to remove restriction of plaintexts. There is a plaintext incoding algorithm Ω to make prime number. We instantiate an message encoding algorithm Ω as follows: We first assign a prime number to a message by a small-sized random padding and check whether the padded message is a prime number. Namely, we append a padding s to the message $\bar{m}$, and then check whether $m = \bar{m} \parallel s$ is a prime number. When we define $\bar{m} \parallel s = \bar{m}^{\log s} + s$, the size of s is determined by the distribution of primes. Let $\pi(m)$ be the number of primes equal to or less than m. Huxley [23] proved that

$$\pi(m + \Delta(m)) - \pi(m) \sim \frac{\Delta(m)}{\log m}$$

is true for almost all x if $\Delta(m) = m^{1/6+\varepsilon}$ ($\varepsilon > 0$ fixed). (See [25] for a survey on this topic.) This result implies that there exists a prime number if $\|s\| = \lceil \frac{\kappa}{6} \rceil$ with overwhelming probability, where $\kappa = \|m\|$.

D Proof of Theorem(s)

As we have discussed above, the security of the generalized public shuffle is the fact that a permutation of inputs does not result in changes of the output of our shuffle. In this section, we prove the following theorem, stating that if its underlying encryption scheme is secure, our generalized public shuffle is unlinkable.

Theorem 2 *Assuming the DDH assumption holds, our generalized public shuffle scheme is unlinkable.*

Proof. We now construct a CCA1 adversary $\mathcal{A}_{\mathsf{cca}}$ that works as follows. A graphical representation of the attacker is given in Figure 1. First, $\mathcal{A}_{\mathsf{cca}}$ sets $\delta = pk$ and gets the system parameter w as defined in its definition. Then as a shuffle challenger, $\mathcal{B} = \mathcal{A}_{\mathsf{cca}}$ sends δ, w to the shuffle adversary $\mathcal{A}$. The adversary $\mathcal{A}$ choose a pair of permutations $\pi_0, \pi_1 \in \Sigma_n$ of his choice and a list of messages $\mathbf{m} = (m_1, \ldots, m_n) \in (\mathbf{M}_{pk})^n$, and sends all of these values to $\mathcal{B} = \mathcal{A}_{\mathsf{cca}}$. $\mathcal{A}_{\mathsf{cca}}$ gets a random bit $b \xleftarrow{\$} \{0,1\}$, from this choose a permutation π_b. Next, it computes $c_i = \mathsf{Enc}^{\mathsf{d}}_{pk}(m_{\pi_b(i)}, r_i)$ for $1 \leq i \leq n$ and $\mathbf{c} = \prod_{i=1}^{n} c_i$, and sends $(c_1, \ldots, c_n)$ and $\mathbf{c}$ to the adversary. The adversary verifies all computations; if this fails abort. Otherwise it can query the decryption oracle $\mathcal{O}_{\mathsf{D}}$ on $\mathbf{c}$. The only problem is that $\mathcal{A}_{\mathsf{cca}}$ does not have sk. Here, we use the fact that $\mathcal{E}$ is CCA2-secure and so in the CCA1 experiment, $\mathcal{A}_{\mathsf{cca}}$ can use the decryption oracle to decrypt everything. However, $\mathcal{A}_{\mathsf{cca}}$ cannot query $\mathcal{O}_{\mathsf{D}}$ on all c_i's and its challenge $\mathbf{c}^*$. This is the important point of this proof here. After finishing its training phase, the adversary sends to $\mathcal{A}_{\mathsf{cca}}$ its challenge consisting of a pair of challenge permutations $\pi_0^*, \pi_1^* \in \Sigma_n$ and a list of challenge messages $\mathbf{m}^* = (m_1^*, \ldots, m_n^*)$. On receiving the challenge, $\mathcal{A}_{\mathsf{cca}}$ does the following according to a random bit $b \xleftarrow{\$} \{0,1\}$ and a random index $j \xleftarrow{\$} [1, n]$:

1. Prepare a pair of challenge messages, $\bar{m}_0 = 1$ and $\bar{m}_1^* = m_{\pi_1^*(j)}$;
2. Send $\bar{m}_0^*, \bar{m}_1^*$ to the CCA1 challenger as its challenge;
3. Receive $c_\beta = \mathsf{Enc}^{\mathsf{d}}_{pk}(m_\beta^*, r^*)$ where β is a random bit chosen by the CCA1 challenger;
4. According to its random choice b,

$$c_j^* = \begin{cases} \mathsf{Enc}^{\mathsf{d}}_{pk}(m^*_{\pi_0(j)}, r_i^*) & \text{if } b = 0 \\ c_\beta & \text{if } b = 1 \end{cases}$$

5. For all $i = [1, n] \backslash \{j\}$, compute $c_i^* = \mathsf{Enc}^{\mathsf{d}}_{pk}\left(m^*_{\pi_b(i)}, r_i^*\right)$;
6. Compute $\mathbf{c}^* = \prod_{i=1}^{n} c_i^*$ and send it to the adversary.

Note that the adversary is not allowed to query $\mathcal{O}_{\mathsf{D}}$ on all c_i^*'s and the challenge ciphertext $\mathbf{c}^*$. Further, due to the restriction of CCA1 experiment the adversary cannot utilize the decryption oracle any more. When the adversary sends its guess b' to the shuffle challenger, $\mathcal{A}_{\mathsf{cca}}$ outputs its guess $\beta' = b'$ to the CCA1 challenger.

From here on, we can see that $\mathcal{A}_{\mathsf{cca}}$ perfectly simulates the generalized public shuffle experiment for the adversary $\mathcal{A}$. So far we have discussed the attack strategy by $\mathcal{A}_{\mathsf{cca}}$, and so we now proceed to prove that $\mathcal{A}_{\mathsf{cca}}$ outputs the correct β with probability $\frac{\varepsilon(\lambda)+1}{2}$ which is non-negligible if $\varepsilon(\lambda)$ is non-negligible.

Define Fail to be the event causing $\mathcal{A}_{\mathsf{cca}}$ to output a random bit in its attack. Further, we say that the generalized public shuffle experiment $\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1$ iff $b = b'$. We have

$$\begin{aligned} \Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1\right] = \Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1 | \neg\mathsf{Fail}\right] \cdot \Pr[\neg\mathsf{Fail}] + \\ \Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1 | \mathsf{Fail}\right] \cdot \Pr[\mathsf{Fail}]. \end{aligned}$$

Now, by the definition of Fail, we have that $\Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1 | \mathsf{Fail}\right] = \frac{1}{2}$. It can be seen that the probability $\mathcal{A}_{\mathsf{cca}}$ outputs an incorrect bit with Fail not happening is negligible, and

$$\Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1 | \neg\mathsf{Fail}\right] \geq 1 - \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\cdot)$. Then we compute $\Pr[\mathsf{Fail}]$ and $\Pr[\neg\mathsf{Fail}]$. By the assumption regarding $\mathcal{A}$, we assume that the advantage $\mathcal{A}$ breaks our shuffle is $\varepsilon(\lambda)$. Thus, $\Pr[\neg\mathsf{Fail}] = \varepsilon(\lambda)$.

In contrast, when $\mathcal{A}$ fails to output a correct bit, then $\mathcal{A}_{\mathsf{cca}}$ always outputs an incorrect bit. Thus, $\Pr[\mathsf{Fail}] = 1 - \varepsilon(\lambda)$. Combining the above, we have

$$\begin{aligned}\Pr\left[\mathbf{Exp}^{\mathsf{PubShf}}_{\mathcal{A}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1\right] &= (1 - \mathsf{negl}(\lambda)) \cdot \varepsilon(\lambda) + \frac{1}{2} \cdot (1 - \varepsilon(\lambda)) \\ &= \varepsilon(\lambda) - \mathsf{negl}'(\lambda) + \frac{1}{2} - \frac{\varepsilon(\lambda)}{2} \\ &= \frac{\varepsilon(\lambda) + 1}{2} - \mathsf{negl}'(\lambda).\end{aligned}$$

Thus, if $\varepsilon(\lambda)$ is non-negligible, then $\mathcal{A}_{\mathsf{cca}}$ succeeds in the generalized public shuffle experiment with non-negligible probability.

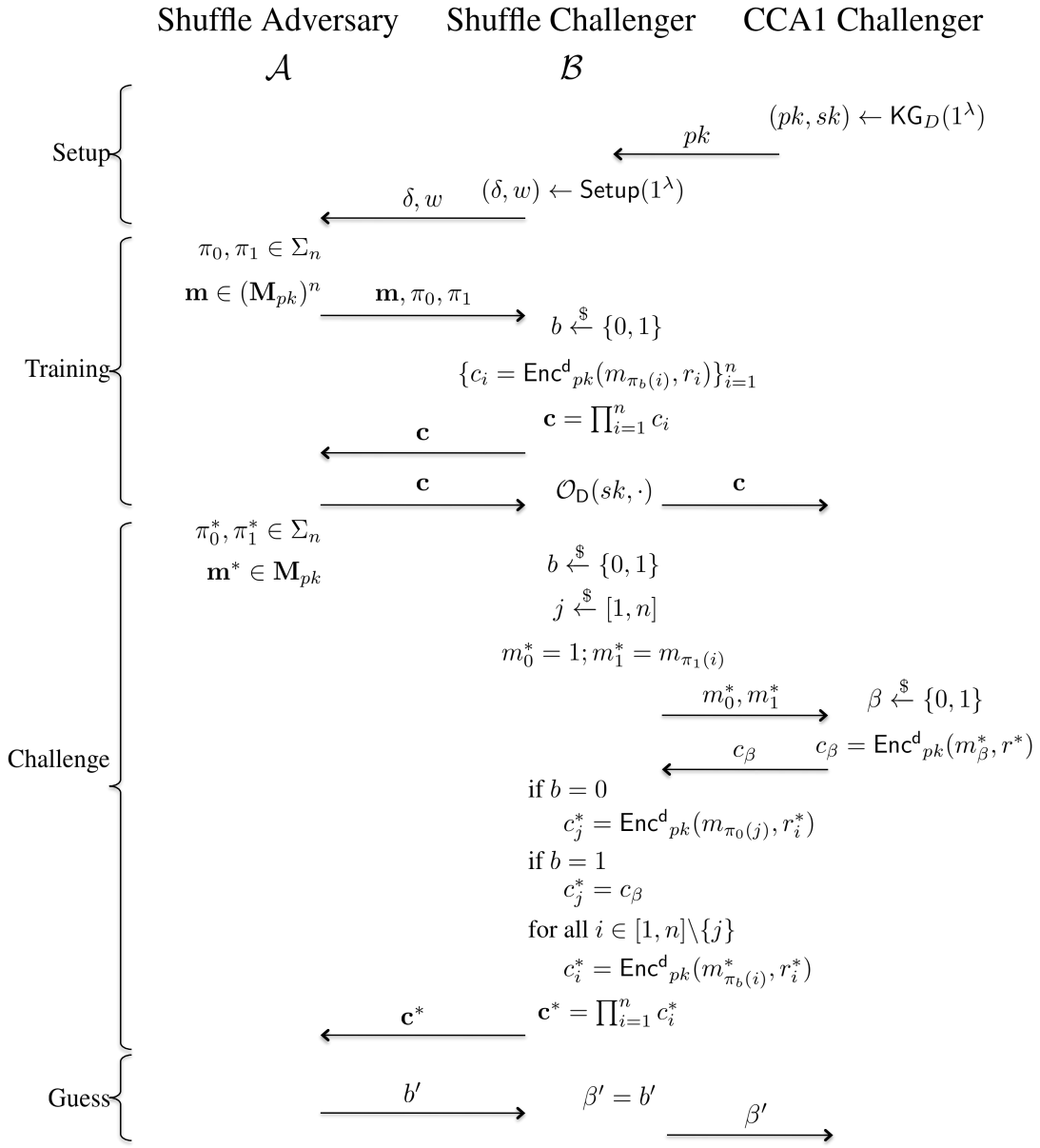

Fig. 1. Graphical View of Security

E Related Work

A mix-net consists of n senders with his own message m_i and a center called a mix-server which publishes his RSA public key. Each sender sends a ciphertext under the mix-server's public-key to the mix-server. The mix-server decrypts them and publicizes $\{m_1, \ldots, m_n\}$ in the lexicographical order. The problem of this single mix-server construction is that the mix-server knows who sent what message. Thus, Chaum proposed a mix-net in which k mix-servers are sequentially connected. Anonymity is protected if at least one mix-server is honest.

A problem of Chaum's construction based on RSA, is that the encryption complexity required by senders increases linearly in the number of mix-servers. Park et al. [32] resolved this problem using ElGamal encryption. Since then, many suggestions have been made how to build mix-nets or prove the correctness of a shuffle [33,1,18,26,27,35,21,36,20]. So this type of shuffles is called a verifiable secret shuffle. Among them, Furukawa and Sako's approach [18] and Neff's work [26,27] are considered as a breakthrough; a linear complexity in the number of input ciphertexts. Furukawa and Sako's approach utilizes permutation matrices and has been further developed in [17,21]. Neff's approach is based on the invariance of polynomials under permutation of the roots. This idea was generalized by Groth [20].

The idea of a public shuffle was introduced by Adida and Wikström [3]. Their approach exploits the notion of public-key obfuscation by Ostrovsky and Skeith [29]. More specifically, they showed how mix-servers can publish an encrypted permutation matrix by the BGN cryptosystem [10] and the Paillier cryptosystem [30]. However, there remain mix-servers having the private permutation. Parampalli et al. [31] combined the Paillier cryptosystem and permutation networks, so obtained sub-quadratic computational complexity ($O(n \log^{3.5} n)$ exponentiations where n is the number of senders) rather than quadratic complexity in [3].

As a different line of relevant work, we need to review private set union protocols [24,22]. Kissner and Song [24] pointed out that private set union protocols are a variant of shuffle protocol. They proposed a protocol for the threshold set union operation, where given a union of input sets, outputs the elements that appears in the union more than a threshold value. More recently, Hazay and Nissim [22] provided more efficient set-union protocols, however, for two-party setting.

F Damgård ElGamal Encryption

Damgård ElGamal Encryption over $\mathbb{F}_{p^3}$. A Damgård ElGamal encryption scheme over $\mathbb{F}_{p^3}$ consists of the following three polynomial time algorithms $(\mathsf{KG}_D, \mathsf{Enc}_D, \mathsf{Dec}_D)$:

- $\mathsf{KG}^{\mathsf{d}}(1^\lambda)$: The key generation algorithm chooses a large prime p such that $(p^3-1) = (p-1)(p^2+p+1) = 2q_1q_2$ for large primes q_1, q_2. Then select an irreducible polynomial $\wp(t) \in \mathbb{F}_p[t]$ of degree 3 and a generator $g(t)$ from $\mathbb{G}_{q_1q_2}$ which is a multiplicative subgroup of $\mathbb{F}^\times_{p^3}$ of order q_1q_2. It computes $y(t) = g(t)^{x_1} \bmod \wp(t)$, $h(t) = g(t)^{x_2} \bmod \wp(t)$ where a secret key x_1, x_2 are randomly chosen from $[0, p^3-2]$, and publishes a public key $pk = \langle p, \mathbb{G}_{q_1q_2}, g(t), y(t), h(t), \wp(t)\rangle$.
- $\mathsf{Enc}^{\mathsf{d}}_{pk}(m(t))$: Encryption with the public key pk and message $m(t) \in \mathbb{G}_{q_1q_2}$ proceeds as follows. First, a random value $r \in [0, p^3-2]$ is chosen. The ciphertext is then published as:

$$C(t) = (u(t), v(t), w(t)) := (g(t)^r \bmod \wp(t), y(t)^r \bmod \wp(t), m(t) \cdot h(t)^r \bmod \wp(t)).$$

- $\mathsf{Dec}^{\mathsf{d}}_{sk}(C(t))$: Suppose that a ciphertext $C(t) = (u(t), v(t), w(t))$ is encrypted with a public key pk and we have a secret key x_1, x_2. Return $\perp$ if $v(t) \neq u(t)^{x_1}$. Otherwise the ciphertext can be decrypted as:

$$m(t) \equiv w(t) \cdot u(t)^{-x_2} \bmod \wp(t).$$