이학박사 학위논문

Cryptographic Shuffles and Their Applications

(암호학적 셔플과 그 응용)

2012년 8월

서울대학교 대학원

수리과학부

김명선

Cryptographic Shuffles and Their Applications

(암호학적 셔플과 그 응용)

지도교수 천정희

이 논문을 이학박사 학위논문으로 제출함

2012년 5월

서울대학교 대학원

수리과학부

김명선

김명선의 이학박사 학위논문을 인준함

2012년 6월

위 원 장 _____ (인)

부 위 원 장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

Cryptographic Shuffles and Their Applications

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Myungsun Kim

Dissertation Director : Professor Jung Hee Cheon

Department of Mathematical Sciences
Seoul National University

August 2012

Abstract

Cryptographic Shuffles and Their Applications

Myungsun Kim

Department of Mathematical Sciences

The Graduate School

Seoul National University

For anonymization purposes, one can use a mix-net. A mix-net is a multi-party protocol to shuffle elements so that neither of the parties knows the permutation linking the input and output. One way to construct a mix-net is to let a set of mixers, so called mix-servers, take turns in permuting and re-encrypting or decrypting the inputs. If at least one of the mixers is honest, the input data and the output data can no longer be linked. In this role, shuffling constitutes an important building block in anonymization protocols and voting schemes. The problem is that the standard shuffle requires anyone who shuffles the input messages to keep his random permutation and randomizers secret. The assumption of a party keeping the secret information may be in some ways quite strong.

Secondly, for this anonymization guarantee to hold we do need to ensure that all mixers act according to the protocol. In general, zero-knowledge proofs (ZKPs) are used for this purpose. However, ZKPs requires the expensive cost in the light of computation and communication.

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 dissertation, we study a method to construct a public shuffle without relying on permutations 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 a constant ciphertext complexity and the other is to use simple ElGamal encryption with a linear ciphertext complexity in the number of users. Both constructions are free of zero-knowledge proofs and publicly verifiable.

Key words: Shuffle, Verifiable Secret Shuffle, Public Shuffle, Mix-net, ElGamal Encryption

Student Number: No. 20008-30081

Contents

Chapter 1

Introduction

We begin with a history of shuffles and related technology, focusing specifically on the introduction of the verifiable schemes. Details can be also found in the work of Nguyen et al. [NSK04] and their journal version [NSK06].

1.1 A Brief History of Shuffles

A number of efficient constructions for verifiable shuffles have been proposed [Abe98, Abe99, FS01, Nef01, FMM+02, Nef03, Fur05, Wik05, GL07, Wik09, BG12]. In Crypto 2001, Furukawa and Sako [FS01] gave a characterization of permutation matrices in terms of two equations that could be efficiently proved, hence proposing an efficient verifiable shuffle with a 3-round proof system. However, the zero-knowledge property of the proof system remains an open problem. Furukawa et al. [FMM+02] noted a flaw in their original proof, proposed a new definition of security for shuffles and proved security of their system with respect to that definition. Neff later gave another efficient construction [Nef01], which was based on a generalization of Chaum-Pedersen proof of knowledge of equality of discrete logarithms

and the fact that a polynomial of degree n has at most n roots [CP92]. An improved version of this proof system is given in [Nef03]. However, like the Furukawa-Sako scheme, the zero-knowledge property of the Neff proof system has not been correctly proved and still remains an open problem. All these schemes use the El Gamal cryptosystem and their security relies on the discrete logarithm assumption. Based on Neff's method, Groth [Gro03] proposed a very efficient proof system that uses homomorphic commitments. The input ciphertexts in this scheme can be encrypted by any homomorphic cryptosystem. More recently, Bayer and Groth [BG12] proposed a verifiable secret shuffle with only sublinear size in the number of senders.

However, all of these shuffle schemes has the common crucial drawbacks as follows: (1) Each shuffle scheme should rely on the secrecy of permutation and randomness; (2) For public verifiability, zero-knowledge proof techniques extensively should be employed, whose cost is usually expensive with respect to computation and communication. As a more or less direct consequence, Adida and Wikström [AW07] proposed a way to shuffle the ciphertexts in public. Later Parampalli et al. [PRT12] improved the computational efficiency.

1.2 Why Shuffling in Public Hard?

The main security objective of a shuffle is to provide unlinkability of its input elements to output elements, and so effectively keeping the permutation secret. A second important property of shuffles is verifiability: that is providing a proof that the output is correctly constructed. Verifiability of shuffles is used to provide robustness for mix-nets: that is ensuring that a mix-net works correctly even if a number of its mix-centrers are malicious. If

a shuffle's proof can be verified by any party, it allows the mix-net to provide public verifiability: that means the mix-net can prove its correct operation to any party. These are important properties of mix-nets and so verifiability of shuffles has received much attention. Shuffles must be efficient and the cost is measured in terms of computation and communication (number of rounds and communicated bits). Proving security properties of shuffles traditionally relied on proving the zero-knowledgeness of the underlying proof system.

In contrast, in [AW07] 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 must be 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 [BY86] proposed a Yes/No voting scheme using homomorphic tallying. 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., by using 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.

1.3 Cryptographic Shuffle Schemes

The idea of a shuffle was introduced by Chaum [Cha81] but he did not give any method to guarantee the correctness. Many suggestions had been made how to build mix-nets or prove the correctness of a shuffle since then, but many of these approaches have been partially or fully broken, and the remaining schemes sometimes suffer from other drawbacks. The scheme of Desmedt and Kurosawa [DK00] assumed that only a small number of mix-servers are corrupt. The approach of Jakobson, Juels, and Rivest [JJR02] needed a relatively big number of mix-server to minimize the risk of tampering with messages or compromising privacy of the senders. Peng et al. [PBDV04] restrained the class of possible permutations and also required that a part of the senders are honest. None of these drawbacks are suffered by the shuffle scheme of Wikström [Wik02] and approaches based on zero-knowledge arguments.

Early contributions using zero-knowledge arguments were made by Sako and Killian [SK95] and Abe [Abe98, Abe99, AH01]. Furukawa and Sako [FS01] and Neff [Nef01, Nef03] proposed the first shuffles for ElGamal encryption with a complexity that depends linearly on the number of ciphertexts. Furukawa and Sako's approach is based on permutation matrices and has been refined further [Fur05, GL07]. Furukawa, Miyachi, Mori, Obana and Sako [FMM+02] presented an implementation of a shuffle argument based on permutation matrices and tested it on mix-nets handling $100,000$ ElGamal ciphertexts. Recently, Furukawa and Sako [FMS10] have reported on another implementation based on elliptic curve groups.

Wikström [Wik09] also used the idea of permutation matrices and suggested a shuffle argument which splits in an offline and online phase. Furthermore, Terelius and Wikström [TW10] constructed conceptually simple

shuffle arguments that allowed the restriction of the shuffles to certain classes of permutations. Both protocols are implemented in the Verificatum mix-net library [Wik10].

Neff's approach [Nef01] is based on the invariance of polynomials under permutation of the roots. This idea was picked up by Groth who suggested a perfect honest verifier zero-knowledge protocol [Gro10]. Stamer [Sta05] reported on an implementation of this scheme. Later Groth and Ishai [GI08] proposed the first shuffle argument where the communication complexity is sublinear in the number of ciphertexts

The goal of shuffling in public is the public-key obfuscation of the shuffle phase of a mix-net comprising either a decryption shuffle or re-encryption shuffle functionality (program) [AW07]. Informally, a public-key obfuscator $\mathcal{O}$ takes a program $\mathcal{F}$ and outputs a new program $\mathcal{O}(\mathcal{F})$ which outputs encryptions of $\mathcal{F}$'s outputs. That is $\exists \diamond, \forall x \, \mathsf{O}(\mathcal{F}) \diamond x = \mathcal{O}(\mathcal{F}(x))$ for some encryption function $\mathcal{O}$ and we say the operator $\diamond$ evaluates the obfuscated program on input x. A formal model is proposed in Definition 3 [AW07] which builds upon an earlier definition by Ostrovsky and Skeith [OS05]. Adida and Wikström present obfuscators for decryption and re-encryption shuffles in the BGN [BGN05] and Paillier [Pai99] cryptosystems respectively. They also proved that their obfuscators are semantically secure (Definition 4 [AW07]). Given a set of parties who sample and obfuscate a shuffle before any input is received, one can construct a mix-net provided that joint decryption is verifiable.

1.4 Contributions of This Work

This dissertation contributes to the practice and theory of cryptographic shuffles, which are twofolds: (1) Definitions; (2) Concrete constructions. Also, each contribution attempts to make cryptographic shuffles more useful and realistic.

1.4.1 Our Definitional Approach

In [NSK04], 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 later 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. Our verifiable public shuffles however remove the order of input ciphertexts itself. We formally define this concept. Then, we formally describe what means by a secure shuffle with respect to verifiability and unlinkability (in [NSK04] the authors called it shuffle privacy).

1.4.2 Our Constructions

Our construction of verifiable public shuffles consists of two steps. First, 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. Then, we 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.

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– 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 $O(\ell)(\mathsf{E} + \mathsf{D}) + O(n^2 \log p)\mathsf{M}_{\mathbb{F}_p}$ 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 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.

1.5 Organization

Chapter 2 reviews a number of cryptographic concepts important to shuffle protocols, including public-key cryptography, homomorphic cryptosystems, and public-key obfuscation. Chapter 3 reviews the legacy shuffle literature with their formal definition, their security model, and several concrete instantiations. Chapter 4 defines the concept of public shuffles, and their exact security model. Then we provide two concrete constructions using homomorphic encryptions.

Chapter 2

Preliminaries

Protocols for shuffles rely on numerous cryptographic building blocks. In this chapter, we review the concepts and notation of these building blocks. We begin with a review of public-key cryptography, its security definitions, and the principal algorithms that we use in practical protocols. We review homomorphic cryptosystems, the interesting properties they yield, and the security consequences of these properties. Then, we consider threshold cryptosystems, where the process of key generation and decryption can be distributed among trustees, a task of great importance to voting systems. We also review zero-knowledge proofs, another critical component of universally verifiable voting, and we briefly review program obfuscation, which is of particular importance to understanding this dissertation.

2.1 Basics

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 $\mathsf{R}(\cdot, \cdot)$ be a polynomial-time computable relation in the size of its first input. Associated with R, we consider a language $\mathcal{L}_{\mathsf{R}} = \{x : \exists\, w \text{ such that } \mathsf{R}(x, w) = 1\}$. A proof system $(\mathcal{P}, \mathcal{V})$ for a relation R allowing a prover $\mathcal{P}$ to prove that a value x is in the associated language $\mathcal{L}_{\mathsf{R}}$. The algorithm $\mathcal{P}$ that outputs a proof Γ that $\Gamma \in \mathcal{L}_{\mathsf{R}}$ and the algorithm $\mathcal{V}$ that verifies the proof.

2.2 Public Key Encryption

Public-key encryption was first suggested by Diffie and Helman [DH76] in 1976, and first implemented by Rivest, Shamir, and Adleman [RSA78] in 1978. At its core, it is a simple, though somewhat counter-intuitive, idea: anyone can encrypt a message destined for Alice, but only Alice can decrypt it. More precisely, Alice can generate a key pair composed of a public key pk and a secret key sk . She then distributes pk publicly, but keeps sk to herself. Using pk, Bob can encrypt a plaintext m into a ciphertext c. The ciphertext c is then effectively "destined" for Alice, since only Alice possesses sk , with which she can decrypt c back into m.

More formally, we can define a public-key cryptosystem as follows.

Definition 2.2.1. 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: $m \leftarrow \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$.

Given such a cryptosystem, one can consider different security definitions.

2.2.1 IND-CPA Security

Intuitively, a cryptosystem is said to be semantically secure if, given a ciphertext c, an adversary cannot determine any property of the underlying plaintext m. In other words, an adversary cannot extract any semantic information of plaintext m from an encryption of m. Semantic security was first defined in 1982 by Goldwasser and Micali [GM82], who also showed that semantic security is equivalent to ciphertext indistinguishability with

chosen plaintexts [GM84]. This latter definition, known as GM Security or IND-CPA , is a more natural one, so we state it here.

In this definition, given a public key pk, the adversary chooses two plaintexts m_0 and m_1 and is then presented with c, a ciphertext of one of these plaintexts, chosen at random. If the adversary cannot guess which of the two plaintexts was chosen for encryption with noticeably better than 50% chance (i.e. picking one at random), then the scheme is said to be secure against chosen plaintext attack.

Definition 2.2.2 ([GM84])**.** 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.

We know of a number of efficient schemes that are IND-CPA-secure.

El Gamal. El Gamal [El 84] is the prime example of an IND-CPA-secure cryptosystem. Consider g the generator of a q-order subgroup of $\mathbb{Z}_p^\times$, where p is prime and q is a large prime factor of $p-1$. Key generation involves selecting a random $x \in \mathbb{Z}_q^\times$, at which point $sk = x$ and $pk = y = g^x \pmod p$. Encryption is then given as

$$c = (\alpha, \beta) = (g^r, m \cdot y^r), r \xleftarrow{\$} \mathbb{Z}_q^\times.$$

Decryption is performed as

$$m = \frac{\beta}{\alpha^x}.$$

Paillier. Paillier [Pai99] is another good example of an IND-CPA-secure cryptosystem. Consider $n = pq$ as in the RSA setting. Consider $\lambda = \text{lcm}(p-1, q-1)$. Consider the function $L(x) = (x-1)/n$. Consider a generator g of $\mathbb{Z}_{n^2}^\times$ specially formed such that $g = 1 \pmod{n}$. The public key is then simply n, while the private key is λ. Encryption of $m \in \mathbb{Z}_n$ is performed as $c = g^m r^n \pmod{n}^2$ for a random $r \xleftarrow{\$} \mathbb{Z}_n^\times$. Decryption is performed as

$$m \equiv \frac{L(c^\lambda \mod n^2)}{L(g^\lambda \mod n^2)} \mod n$$

We provide here a brief explanation of the Paillier cryptosystem, given that it is particularly interesting and useful for our work in this dissertation. Recall that:

- $\varphi(n) = (p-1)(q-1)$ is Euler's totient function
- $\lambda = \text{lcm}(p-1, q-1)$ is the output of Carmichael's function on n
- The order of $\mathbb{Z}_{n^2}^\times$ is $n\varphi(n)$
- For any $a \in \mathbb{Z}_{n^2}^\times$:
  - $a^\lambda \equiv 1 \mod n$
  - $a^{\lambda n} \equiv 1 \mod n^2$

Thus, consider the decryption function defined above, in particular the denominator. Recall that $g = 1 \mod n$, which we can also write $g = n\alpha + 1$ for some integer α.

$$\begin{aligned} L(g^\lambda \mod n^2) &= \frac{((1+n\alpha)^\lambda \mod n^2) - 1}{n} \\ &= \frac{(n\alpha\lambda) \mod n^2}{n} \\ &\equiv \alpha\lambda \mod n^2 \end{aligned}$$

Note that the exponentiation above reduces to the multiplication because all other monomials in the expansion are multiples of n^2. One can then easily see that, because r^n will cancel out by exponentiation to λ:

$$L(c\lambda \mod n^2) \equiv m\alpha\lambda \mod n^2$$

and thus that the decryption works as specified.

2.2.2 IND-CCA Security

Indistinguishability with respect to adaptively-chosen plaintexts is not enough for all applications. Intuitively, one should also consider the possibility that the adversary can obtain the decryption of a few chosen ciphertexts before receiving the challenge ciphertext. This notion of security is called IND-CCA-security, informally known as "security against lunchtime attacks." The model is that the adversary might have access to a decryption box while the owner is "out to lunch" (possibly metaphorically.) Later, the adversary will try to use the information gained over lunch to decrypt other ciphertexts.

Definition 2.2.3. labeldef-indcca1 A public-key cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ with a security parameter λ is is said to be IND-CCA-secure given a decryption oracle $\mathcal{O}_{\mathsf{D}}$: if after the standard CCA game being played with any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the advantage $\mathbf{Adv}^{\mathsf{cca}}_{\mathcal{E},\mathcal{A}}(\lambda)$, formally defined as

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

is negligible in λ for all sufficiently large λ.

2.3 Homomorphic Public-key Encryption

Homomorphic public-key cryptosystems exhibit a particularly interesting algebraic property: when two ciphertexts are combined in a specific, publicly-computable fashion, the resulting ciphertext encodes the combination of the underlying plaintexts under a specific group operation, usually multiplication or addition.

Definition 2.3.1. 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 2.3.2. 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 $\mathbf{M}_{pk}$.

Re-Randomization

An immediate consequence of a cryptosystem's homomorphic property is its ability to perform re-randomization: given a ciphertext c, anyone can create a different ciphertext c that encodes the same plaintext as c. Recall that $\mathcal{E}$ is homomorphic for addition if $(\mathbf{M}_{pk}, +)$ forms a group, which means there exists an identity plaintext m_0 such that, $\forall\, m \in \mathbf{M}_{pk}, m + m_0 = m$. Thus, given a homomorphic cryptosystem $\mathcal{E}$, we can define the re-randomization algorithm as follows:

$$\mathsf{ReRand}_{pk}(c; r) = c \cdot \mathsf{Enc}_{pk}(m_0; r)$$

If $\mathsf{Dec}_{sk}(c) = m$, then $\mathsf{Dec}_{sk}(\mathsf{ReRand}_{pk}(c)) = m$, too.

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.

Security of Homomorphic Cryptosystems

The malleability of ciphertexts in homomorphic cryptosystems limits the security of such schemes. In particular, the ability to re-randomization immediately indicates that the system is not IND-CCA2-secure, and can be at best IND-RCCA-secure. Even more significant, the ability to create a ciphertext

of a related but different plaintext breaks even IND-RCCA security. Specifically, an adversary can take the challenge ciphertext c, create $c = c \cdot \mathsf{Enc}_{pk}(\bar{m})$ for some $\bar{m}$ known to the adversary, query $\mathcal{O}_{\mathsf{D}}$ with c to obtain m, and compute $m = m + \bar{m} - 1$. It has not been known whether homomorphic schemes can be IND-CCA-secure, but in 1991, Damgård proposed what we will call the Damgård's Elgamal (DEG) cryptosystem [Dam91]. DEG is a relatively straightforward modification of Elgamal that employs an additional exponentiation to reject "invalid" ciphertexts. DEG was proven to be CCA1-secure under a nonfalsifiable knowledge-of-the-exponent assumption [Nao03].

Homomorphic Schemes in Practice

A number of practical schemes are homomorphic.

RSA. In raw RSA, encryption is performed as $c = m^e \mod n$. Thus, clearly, $c_0 \cdot c_1 = (m_0 \cdot m_1)^e \mod n$. Raw RSA is thus homomorphic on operation $\cdot$. That said, raw RSA is not even IND-CPA-secure, which means it is not very useful in many applications. RSA-OAEP, on the other hand, is quite useful, but loses the homomorphic property due to the non-malleable OAEP padding.

El Gamal. In El Gamal, encryption is performed as $c = (g^r, m \cdot y^r)$. Thus, if we define $\times$ as the element-wise product of ciphertext pairs, then El Gamal is homomorphic for $\times$:

$$(g^{r_1}, m_1 \cdot y^{r_1}) \times (g^{r_2}, m_2 \cdot y^{r_2}) = \left(g^{r_1+r_2}, (m_1 m_2) \cdot y^{r_1+r_2}\right).$$

El Gamal is particularly interesting: it exhibits a homomorphism and is IND-CPA-secure.

Paillier. In Paillier, encryption is performed as $c = g^m r^n \mod n^2$. Clearly, this scheme is homomorphic for $+$ over the plaintext space $\mathbb{Z}_n$:

$$\begin{aligned} \mathsf{Enc}_{pk}(m_1, r_1) \cdot \mathsf{Enc}_{pk}(m_2, r_2) &= (g^{m_1} r_1^n) \cdot (g^{m_2} r_2^n) \\ &= g^{m_1+m_2}(r_1 r_2)^n \\ &= \mathsf{Enc}_{pk}(m_1 + m_2, r_1 r_2). \end{aligned}$$

Note that Paillier decryption is efficient, which means the plaintext domain can be superpolynomial while retaining the additive homomorphism.

2.4 Zero-Knowledge Proofs

A major component of verifiable voting protocols is the zero-knowledge proof. In a zeroknowledge proof, a prover $\mathcal{P}$ interacts with a verifier $\mathcal{V}$ to demonstrate the validity of an assertion, e.g., "ciphertext c under public key pk decrypts to '`I am Sim`'." If the prover is honest—i.e. the assertion is true—then the verifier should accept this proof. If the prover is dishonest—i.e., the assertion is false—then the verifier should reject this proof with noticeable probability. Finally, the verifier should learn nothing more than the truth of the assertion. In particular, the verifier should be unable to turn around and perform this same (or similar) proof to a third party. The notion of "zero-knowledge" is tricky to define: how can one capture the concept that no knowledge has been transferred? The accepted approach is to look at the verifier and see if its participation in the proof protocol bequeathed it any new capability. The protocol is zero-knowledge if, no matter what the verifier outputs after the protocol, it could have produced the very same output without interacting with the prover. Thus, though the verifier may be personally convinced from its interaction that the prover's assertion is

indeed true, the verifier is unable to relay any new information convincingly, in particular he cannot perform the proof on his own.

The prover's assertion is formally defined as "x is in language $\mathcal{L}$," where x is a string, and $\mathcal{L}$ is a language, usually an NP language. Thus, the prover $\mathcal{P}$ is given x and a witness w for x such that $\mathsf{R}(x, w) = 1$, where R is the binary relation for language $\mathcal{L}$. The verifier $\mathcal{V}$ only gets x as input, of course. The zero-knowledge property of the protocol ensures that the witness w, and in fact any non-trivial function of the witness, remains hidden from $\mathcal{V}$.

Definition 2.4.1 (Perfect Zero-Knowledge Proof)**.** An interactive protocol $(\mathcal{P}, \mathcal{V})$ for language $\mathcal{L}$ is defined as a perfect zero-knowledge proof if there exists a negligible function $\nu(\cdot)$ such that the protocol has the following properties:

- **Completeness**: $\forall\, x \in \mathcal{L}, \Pr[(\mathcal{P}, \mathcal{V})(x, w) = 1] > 1 - \nu(\lambda)$.
- **Soundness**: $\forall\, \mathcal{P}^*, \forall\, x \notin \mathcal{L}, \Pr[(\mathcal{P}^*, \mathcal{V}) = 1] < \frac{1}{2}$.
- **Zero-Knowledge**: $\exists$ PPT $S, \forall\, \mathcal{V}^*, \forall\, x \in \mathcal{L}, S(x) \approx (\mathcal{P}, \mathcal{V}^*)(x, w)$

2.4.1 Zero-Knowledge Variants

A few variants of this definition exist:

- **Computational Zero-Knowledge (CZK)**: The verifier $\mathcal{V}$, and thus the dishonest version $\mathcal{V}^*$, are probabilistic polynomial-time. In other words, a surprisingly powerful verifier might be able to extract some knowledge from an execution of a CZK protocol.
- **Zero-Knowledge Argument**: The prover $\mathcal{P}$ is assumed to be computationally constrained, i.e., it is a PPT algorithm. This setting must

be considered with care, as the PPT limitation is dependent on the security parameter λ, but $\mathcal{P}$ may spend significant time preparing for the protocol execution.

– **Honest-Verifier Zero-Knowledge (HVZK)**: The verifier $\mathcal{V}$ is expected to perform according to the protocol. In particular, as the verifier is usually expected to submit a random challenge to the prover, an honest verifier will always flip coins when picking a challenge and will never base his challenge on the prover's messages. The result of an HVZK assumption is that the simulation proof can focus on simulating a transcript of the interaction, rather than simulating anything $\mathcal{V}$ could output. An HVZK protocol can be turned into a non-interactive zero-knowledge (NIZK) proof using the Fiat- Shamir heuristic [FS87], where the verifier's random messages are generated using a hash function applied to the prior protocol messages. This hash function must be modeled as random oracle, which has recently caused some concern in the theoretical cryptography community [GT03].

Zero-knowledge proofs play a big role in verifiable shuffle protocols, where each sender must prove that it performed its designated action correctly while preserving shuffler's privacy. As the integrity of the shuffler takes precedence over his privacy, it can be immediately said that computational zero-knowledge proofs will be preferable to zero-knowledge arguments.

2.4.2 Proof of Knowledge

Certain zero-knowledge proofs provide an additional property that is particularly useful in proving overall protocol security: they prove knowledge of the witness, not just existence. In particular, this means that, given rewindable,

black-box access to the prover program $\mathcal{P}$, one can extract a witness w to the assertion that $x \in \mathcal{L}$. More formally, we define a zero-knowledge proof of knowledge as follows.

Definition 2.4.2 (Zero-Knowledge Proof of Knowledge)**.** An interactive protocol $(\mathcal{P}, \mathcal{V})$ for a language $\mathcal{L}$ is defined as a zero-knowledge proof of knowledge if the protocol is zero-knowledge and it has the following, additional property:

- **Extraction**: $\exists$ PPT $\mathcal{E}, \forall (x, w) \in \mathsf{R}, \mathcal{E}^{\mathcal{P}(x,w)}() = w$. By $\mathcal{E}^{\mathcal{P}(x,w)}$, we mean that we take the prover program $\mathcal{P}$, provide it with inputs (x, w), and give the extractor Enc black-box access to this initialized prover program, allowing the extractor to rewind, reply, and provide continuing inputs to $\mathcal{P}$.

A proof-of-knowledge protocol can be particularly useful in the context of reduction proofs, since the extraction property allows a simulator to get the witness and use it in the reduction process. A zero-knowledge proof without extractability is much more difficult to integrate into a complete protocol security proof.

2.5 Public-Key Obfuscation

Ostrovsky and Skeith [OS05] proposed a slightly different and weaker model of obfuscation, where the outputs of the obfuscated program are encrypted versions of the outputs of the original, unobfuscated program. In other words, their technique allows for outsourcing most of a computation, but not all of it: a final decryption is still required after the obfuscated program has been executed. They name this model public-key obfuscation. Interestingly, because

the outputs of a public-key-obfuscated program are encrypted, Ostrovsky and Skeith's definition is able to capture the additional notion of security missing from the Barak et al. and Tauman-Kalai and Goldwasser definitions: output indistinguishability. Informally, a public-key obfuscator is secure when an adversary cannot distinguish between the public-key obfuscations of two programs it selected. We now provide a more formal definition.

Definition 2.5.1. A program class is a family $\{\mathbb{P}_\lambda\}_{\lambda\in\mathbb{N}}$ of sets of programs such that there exists a polynomial $s(\cdot)$ such that $|P| \leq s(\lambda)$ for every $p \in \mathbb{P}_\lambda$. The program class is said to be PPT if, for every $\lambda \in \mathbb{N}$, for every $p \in \mathbb{P}_\lambda$, P runs in probabilistic polynomial time in λ.

Definition 2.5.2 (Public-Key Obfuscation)**.** The algorithm $\mathcal{O}$ is a public-key obfuscator for the program class $\{\mathbb{P}_\lambda\}$ and the cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ if:

– Correctness: there exist a negligible function $\nu(\cdot)$ such that, for every $\lambda \in \mathbb{N}$, for every $P \in \mathbb{P}_\lambda$, for all inputs x:

$$\Pr[\mathsf{Dec}_{sk}(\mathcal{O}(P)(x, r)) = P(x)] > 1 - \nu(\lambda)$$

taken over the choice of r, an extra input which parameterizes the execution of $\mathcal{O}(P)$.

– Conciseness: there is a polynomial $l(\cdot)$ such that, for every $\lambda \in \mathbb{N}$ and for every $P \in \mathbb{P}_\lambda$,

$$|\mathcal{O}(P)| \leq l(|P|).$$

Now, we must describe what it means for this public-key obfuscator to be secure. Ostrovsky and Skeith give an indistinguishability-based definition. Thus, consider first the indistinguishability experiment. Informally, we

first generate a keypair. Based on the public key, the adversary selects two programs from the program class. We obfuscate one of the two, selected at random, and we ask the adversary to guess which one was obfuscated. We now formalize this intuition, which is much like the semantic security for encryption schemes which we explored earlier in this chapter. We denote $\mathbb{P} = \{\mathbb{P}_\lambda\}$.

Experiment $\mathbf{Exp}^{\mathsf{ind}_b}_{\mathbb{P},\mathcal{O},\mathcal{E},\mathcal{A}}(\lambda)$

$$(pk, sk) \xleftarrow{\$} \mathsf{KG}(1^\lambda);$$

$$(P_0, P_1, \mathsf{state}) \leftarrow \mathcal{A}_1(pk);$$

$$b' \leftarrow \mathcal{A}_2(\mathcal{O}(1^\lambda, pk, sk, P_b), \mathsf{state})$$

If $P_0, P_1 \in \mathbb{P}_\lambda$ return b', otherwise a random bit.

We can now define the security property we seek from a public-key obfuscator.

Definition 2.5.3 (Secure Public-Key Obfuscation)**.** A public-key obfuscator $\mathcal{O}$ for a program class with respect to a cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is secure, or polynomially indistinguishable, if there exists a negligible function $\nu(\cdot)$ such that:

$$\left|\mathbf{Exp}^{\mathsf{ind}_0}_{\mathbb{P},\mathcal{O},\mathcal{E},\mathcal{A}}(\lambda) - \mathbf{Exp}^{\mathsf{ind}_1}_{\mathbb{P},\mathcal{O},\mathcal{E},\mathcal{A}}(\lambda)\right| < \nu(\lambda).$$

Chapter 3

Verifiable Secret Shuffles: A Review

3.1 Introduction

Consider a set of senders, each with a private message, who wish to generate a shuffled list of these messages, while keeping the permutation secret. Protocols that implement this functionality were first introduced by Chaum [Cha81] in 1981, who called them mix-nets. There are many different types of mix-nets, and many different definitions and constructions.

Non-cryptographic mixnets tend to mix inputs more or less synchronously for low-latency applications such as anonymized web browsing [DMS04]. These mixnets generally focus on achieving some level of privacy, without usually worrying about robustness: if a few mix servers drop or otherwise corrupt messages, the impact on the application is generally not horrible: a sender can simply retry using a different set of mix servers.

By contrast, robust mixnets handle applications like voting, which have significantly different requirements. On the one hand, they provide far more

flexibility: mixing can take hours or, in some cases, even days, because shuffling is performed in large, well-defined batches, with no need for real-time responses. On the other hand, the correctness requirements are much more stringent: inputs should not be lost or altered, in some cases even when all mix servers are corrupt. The privacy of the shuffle permutation is also important, and should be provably–not just heuristically–protected.

In this chapter, we review the past 25 years of literature on verifiable shuffles. We note that this area of research has been quite productive, with numerous directions explored, and fascinating techniques developed to improve efficiency. The security definitions have evolved. In short, shuffles have been a fertile area of research.

3.2 Notation and Definitions

Firstly, we rephrase the formal definition of a verifiable shuffle given by Nguyen et al. [NSK04, Def. 4], In [NSK04] they extensively use a re-randomizable public-key encryption scheme. We do not construct a secret verifiable shuffle, but we also rely on the re-randomization property of an encryption scheme with semantic security. We additionally introduce some notation used to define public verifiability. We then extend it to the definition of a generalized shuffle.

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

$$\exists \left(\pi, s_1, \ldots, s_{|\mathbf{c}|}\right), \forall i \in [1, |\mathbf{c}|] : \hat{c}_{\pi(i)} = \mathsf{ReRand}_{pk}(c_i, s_j) \text{ for some } j \in [1, |\mathbf{c}|] \tag{3.2.1}$$

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

Definition 3.2.1 (Verifiable Shuffle)**.** 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 it 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 $\mathsf{Dec}_{sk}(c_i) = \mathsf{Dec}_{sk}\left(\hat{c}_{\pi(i)}\right)$ for some $i \in [1, n]$ where $c_i = \mathsf{Enc}_{pk}(m_i, r_i)$ and $\hat{c}_{\pi(i)} = \mathsf{ReRand}_{pk}\left(c_{\pi(i)}, s_{\pi(i)}\right)$. 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 a secret parameter w is not empty but Verify does not take it as input, we call this type of shuffle schemes a publicly verifiable secret shuffle scheme, shortly a *secret* shuffle. We remark that decryption shuffles also belong to secret shuffle because they use a random secret permutation in shuffling.

3.3 Security

There are two security requirements. Privacy requires an honest shuffle to protect its secret permutation whereas verifiability requires that any attempt by a malicious shuffle to produce an incorrect output must be detectable.

3.3.1 Verifiability for Secret Shuffles

We rephrase the verifiability condition for secret shuffles in our language. The reader is encouraged to refer to [NSK04] for in-depth discussions on the verifiability condition of shuffles.

Definition 3.3.1 ([NSK04])**.** Let a set of algorithms $(\mathcal{P}, \mathcal{V})$ be a proof system for an efficient shuffle relation Φ with associated language $\mathcal{L}_\Phi$. A shuffle scheme $\Phi_\mathcal{E} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ 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 $x = (\delta, \mathbf{c}, \hat{\mathbf{c}}) \in \mathcal{L}_\Phi$, $(\mathcal{P}, \mathcal{V})(x, \Gamma) = 1$ 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 $x \notin \mathcal{L}_\Phi$ but $(\mathcal{A}, \mathcal{V})(x, \Gamma) = 1$, is negligible in the security parameter λ.

3.3.2 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 [NSK04]. Nguyen et al. [NSK04] 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 3.3.2 (Unlinkability in [NSK04])**.** 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 $w_b \overset{\$}{\leftarrow} \{\pi_0, \pi_1\}$;

$v \leftarrow \left(\hat{\mathbf{c}}, \mathsf{View}^{\mathcal{P},\mathcal{V}}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma), \mathbf{c}, \{m_i\}_{i=1}^n, \{r_i\}_{i=1}^n\right)$ where $c_i = \mathsf{Enc}_{pk}(m_i, r_i)$;

$b' \leftarrow \mathcal{A}(\delta, v)$;

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

Definition 3.3.3 (Unlinkability for a Secret Shuffle)**.** Let $\Phi_{\mathcal{E}} = (\mathsf{Setup}, \mathsf{Shuffle}, \mathsf{Verify})$ be a generalized secret shuffle scheme.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{Shuffle}}(\Phi_{\mathcal{E}}, \lambda)$

$\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$;

$(\mathbf{r}_0, \mathbf{r}_1, \mathbf{c}) \leftarrow \mathcal{A}(\delta, n, \ell)$ 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 $w_b \xleftarrow{\$} \{\mathbf{r}_0, \mathbf{r}_1\}$;

$\nu \leftarrow \big(\hat{\mathbf{c}}, \mathsf{View}^{\mathcal{P},\mathcal{V}}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma), \mathbf{c}, \{m_i\}_{i=1}^n, \{r_i\}_{i=1}^n\big)$ 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 secret shuffle scheme is *unlikable* if the advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{Shuffle}}(\Phi_{\mathcal{E}}, \lambda)$ is negligible in the security parameter λ.

3.4 Selected Prior Work

As mentioned above, there are numerous prior work we have to pay attention to. However, in this section we just review two selected work. The reason

is why the Furukawa-Sako scheme [FS01] is the first verifiable, efficient, and secure shuffle scheme, and the Groth scheme [Gro03] is the most efficient.

3.4.1 Furukawa-Sako Protocol

Represent the permutation π_j by the permutation matrix M^j , with $M^j_{ab} = 1$ if and only if $\pi_j(a) = b$, and $M^j_{ab} = 0$, otherwise. A nice way of using this matrix representation to achieve efficient zero-knowledge proofs is described in [FS01, Fur05]. It is based on the next fact [FS01]: Let δ_{ij} be 1 if $i = j$ and 0 otherwise. Let δ_{ijk} be 1 if $i = j = k$ and 0 otherwise. Let q be a large prime. An $n \times n$ matrix M is a permutation matrix if and only if

$$\sum_h M_{hi} M_{hj} = \delta_{ij} \tag{3.4.2}$$

and

$$\sum_h M_{hi} M_{hj} M_{hk} = \delta_{ijk}. \tag{3.4.3}$$

Thus, instead of re-randomization, one could prove that

$$c_{ji} = \mathsf{ReRand}_{pk}\left(\prod_{i=1}^{n} c_{j-1,i}^{M_{ji}}, r'_{ji}\right)$$

and that Eq. (3.4.2) and Eq. (3.4.3) are true.

Equation (3.4.2) can be verified by defining $s_i = \sum_{i=1}^n M_{ji} e_j$, for e_j chosen randomly by verifier, and then checking that $\sum_{i=1} s_i^2 = \sum_{i=1}^n e_j^2$. Due to Eq. (3.4.2),

$$s_i^2 = \sum_{j=1}^{n} M_{ij} M_{ik} e_j e_k = \sum e^2_{\chi(i)}$$

where $\chi(j-1)$ is some permutation, and

$$\sum_{i=1}^{n} s_i^2 = \sum e^2_{\chi(i)} = \sum_{i=1}^{n} e_i^2.$$

Analogously, Eq. (3.4.3) is verified by checking that $(\sum M_{ij}e_j)^3 = \sum_{i=1}^{n} e_i^3$. Some more care has to be taken to achieve complete security [FS01, Fur05].

In this approach, the prover must make approximately $8n$ exponentiations, and the verifier must make approximately $10n$ exponentiations. When $\|p\|= 1024$ and $\|q\|= 160$, it takes about $5280n$ bits to communicate the proof of knowledge.

3.4.2 Groth Protocol

An alternative, somewhat more efficient, verifiable shuffle was proposed by Groth [Gro03]. It assumes the use of an IND-CPA secure homomorphic cryptosystem (e.g., ElGamal, Paillier, or Damgård and Jurik [DJ01]), and of a compatible homomorphic commitment scheme. In this verifiable shuffle, the prover first commits to the shuffle. The verifier picks a vector of random integers, and the prover proves that the scalar product of this vector and the vector of encrypted votes is preserved after the shuffling.

Commitment Schemes. A commitment scheme is a function $\mathsf{com} : X \times R \to Y$ from the plaintext space X and random coin space R to the commitment space Y. A commitment scheme com is (a) statistically hiding if the commitment $y = \mathsf{com}(m, r)$ leaks a statistically insignificant amount of information about the plaintext m and the coin r; and (b) computationally binding if given commitment $y = \mathsf{com}(m, r)$ to some element r from the plaintext space, it is hard to find $m' \in \mathbf{M}_{pk}, m' \neq m$, and an r', s.t. $y = \mathsf{com}(m', r')$. For the best known commitment schemes (e.g., Pedersen's [Ped91]), the plaintext space is equal to $\mathbb{Z}_N$ for some N. Therefore, $\mathsf{com}(m, r) = \mathsf{com}(m + N, r)$ and therefore, such commitment schemes are not binding over the integers.

Groth's Verifiable Shuffle. In more details, Groth's verifiable shuffle is as follows:

- Prover: For $j = \{1, \ldots, n\}$, commit to commit to $C_{1,i} \leftarrow \mathsf{com}_{pk}(\pi(j), r_{2,j})$. Send $C_{1,i}$, together with a proof of correct shuffle, to verifier.
- Verifier: For $j = \{1, \ldots, n\}$, generate a random t_j and send t_j to prover.
- Prover: For $j = \{1, \ldots, n\}$, $C_{2,i} \leftarrow \mathsf{com}_{pk}(t_{\pi(j)}, r_{t_j})$. Send $\{C_{1,i}\}_i$, together with a proof of correct shuffle and that this shuffle was the same as on Step 1, to verifier.
- Prover proves in zero-knowledge that $\mathsf{Dec}_{sk}\left(\prod c_{j,i}^{t_{\pi(i)}}\right) = \mathsf{Dec}_{sk}\left(\prod c_{j-1,i}^{t_i}\right)$.

The three first proofs of knowledge can be executed jointly, by proving that for a random γ chosen by the verifier, $\{C_{1,i}C_{2,i}^{\gamma}\}$ commits to $\{i + \gamma t_i\}$. The proof that $\{c_i\}$ commits to $\{m_i\}$ can be done as follows: Prover sets $c_m = \mathsf{com}_{pk}(m; 0)$, for m generated by the verifier, and proves that the multiplication of the contents of $c_1 c_m^{-1}, \ldots, c_n c_m^{-1}$ is equal to $\prod_{i=1}^{n}(m_i - m)$ All (or at least a significant fraction) of the resulting voters zero-knowledge multiplication proofs can be done in parallel by using multi-commitments.

In this approach, the prover must perform approximately $6n$ exponentiations, and the verifier must perform approximately $6n$ exponentiations. When $\|p\| = 1024$ and $\|q\| = 160$, it takes about $1184n$ bits to communicate the proof of knowledge.

3.5 Public Shuffles with Private Permutation

3.5.1 Introduction

The goal of shuffling in public is the public-key obfuscation of the shuffle phase of a mix-net comprising either a decryption shuffle or re-encryption shuffle functionality (program) [AW07]. Informally, a public-key obfuscator $\mathcal{O}$ takes a program $\mathcal{F}$ and outputs a new program $\mathcal{O}(\mathcal{F})$ which outputs encryptions of $\mathcal{F}$'s outputs. That is $\exists \diamond, \forall x \, \mathcal{O}(\mathcal{F}) \diamond x = \mathcal{O}(\mathcal{F}(x))$ for some encryption function $\mathcal{O}$ and we say the operator $\diamond$ evaluates the obfuscated program on input x. A formal model is proposed in Definition 3 [AW07] which builds upon an earlier definition by Ostrovsky and Skeith [OS05]. Adida and Wikström present obfuscators for decryption and re-encryption shuffles in the BGN [BGN05] and Paillier [Pai99] cryptosystems respectively. They also prove that their obfuscators are semantically secure (Definition 4 [AW07]). Given a set of parties who sample and obfuscate a shuffle before any input is received, one can construct a mix-net provided that joint decryption is verifiable.

3.5.2 Adida and Wikström Protocol

Adida and Wikström [AW07] proposed two schemes shuffling in public allow a shuffle to be precomputed. These schemes imply that no mix servers need be present at election time for mixing to take place. One downside of their schemes is that the scheme significantly restricts the number and size of votes. Additionally the main disadvantage of shuffling in public is its inefficiency, with generation and evaluation of the precomputed shuffle requiring $O(n^2)$ exponentiations.

The BGN Cryptosystem

We denote the BGN cryptosystem by $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$. It operates in two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ both of order $n = q_1 q_2$, where q_1 and q_2 are distinct prime integers. We use multiplicative notation in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and denote by g a generator in $\mathbb{G}_1$. The groups $\mathbb{G}_1$ and $\mathbb{G}_2$ exhibit a polynomial-time computable bilinear map $e : \mathbb{G}_1 \times G1 \rightarrow \mathbb{G}_2$ such that $G = e(g, g)$ generates $\mathbb{G}_2$. Bilinearity implies that $\forall\, u, v \in \mathbb{G}_1$ and $\forall\, a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$. We refer the reader to [BGN05] for details on how such groups can be generated and on the cryptosystem's properties, which we briefly summarize here.

Key generation. On input 1^λ, KG generates $(q_1, q_2, \mathbb{G}_1, g, \mathbb{G}_2, e(\cdot, \cdot))$ as above such that $n = q_1 q_2$ is a λ-bit integer. It chooses $u \in \mathbb{G}_1$ randomly, defines $h = u^{q_2}$, and outputs a public key $pk = (n, \mathbb{G}_1, \mathbb{G}_2, e(\cdot, \cdot), g, h)$ and secret key $sk = q_1$.

Encryption. On input pk and m, Enc selects $r \in \mathbb{Z}_n$ randomly and outputs $c = g^m h^r$.

Decryption. On input $sk = q_1$ and $c \in \mathbb{G}_1$, Dec outputs $\log_{g^{q_1}}(c^{q_1})$.

Homomorphisms. The BGN cryptosystem is additively homomorphic. This scheme needs this property, but this scheme also exploits its one-time multiplicative homomorphism implemented by the bilinear map:

$$e(\mathsf{Enc}_{pk}(m_0, r_0), \mathsf{Enc}_{pk}(m_1, r_1)) = \mathsf{Enc}_{pk}(m_0 m_1, m_0 r_1 + m_1 r_0 + (\log_g u) q_2 r_0 r_1)$$

The result is a ciphertext in $\mathbb{G}_2$ which cannot be efficiently converted back to an equivalent ciphertext in $\mathbb{G}_1$. Thus, the multiplicative homomorphism can be evaluated only once, after which only homomorphic additions are possible.

For notational clarity, we write $c_1 \oplus c_2 := c_1 c_2$ for ciphertexts in $\mathbb{G}_1$ or $\mathbb{G}_2$ and $c_1 \otimes c_2 := e(c_1, c_2)$ for ciphertexts in $\mathbb{G}_1$.

BGN-based Scheme

The first obfuscator is based on the fact that matrix multiplication only requires an arithmetic circuit with multiplication depth 1. Thus, the BGN cryptosystem can be used for homomorphic matrix multiplication. Consider an $n_1 \times n_2$-matrix $C = (c_{ij}) = \mathsf{Enc}_{pk}(a_{ij})$ and an $n_2 \times n_3$-matrix $C' = (d_{jk}) = \mathsf{Enc}_{pk}(b_{jk})$, and let $A = (a_{ij})$ and $B = (b_{jk})$. Define homomorphic matrix multiplication by

$$C \star C' := \left(\bigoplus_{j=1}^{n_2} c_{ij} \otimes d_{jk} \right)$$

and have

$$\mathsf{Dec}_{sk}(C \star C') = \left(\sum_{j=1}^{n_2} a_{ij} b_{jk} \right) = AB.$$

Paillier-based Scheme

We use the additive homomorphism and the special homomorphic property exhibited above to define a form of homomorphic matrix multiplication of matrices of ciphertexts. Given an n-permutation matrix $\Lambda^\pi = (\lambda^\pi_{ij})$ and randomness $r, s \in (\mathbb{Z}_N^\times)^{n \times n}$, define $C^\pi = \left(c^\pi_{ij}\right) = \mathsf{Enc}_{N^3}(\lambda^\pi_{ij} \mathsf{Enc}_{N^2}(0, r_{ij}), s_{ij})$ where Enc_{N^3} denotes Paillier's cryptosystem using modulus N^3 and Enc_{N^3} denotes that using modulus N^2. We define a kind of matrix multiplication of $d = (d_1, \ldots, d_n) \in C^n_{N^2}$ and C^π:

$$d \star C^\pi = \left(\prod_{i=1}^{n} (c^\pi_{ij})^{d_i} \right)$$

and have

$$\mathsf{Dec}_{p,2}(\mathsf{Dec}_{p,3}(d \star C^\pi)) = (m_{\pi(1)}, \ldots, m_{\pi(n)}).$$

Chapter 4

Verifiable Public Shuffles

In this chapter, we describe 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.

4.1 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. A Chaumian mix-net consists of multiple mix-servers which have their private permutation and randomness. 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.

However, the assumption that there exists an honest mix-server (a.k.a., a trusted third party) in real life, may be quite strong. Thus many researchers have focused on strengthening verifiability in Chaum's construction (e.g., [FS01, Nef01, Nef03, Fur05, Wik05, GL07]). Their goal is to efficiently enforce each mix-server to behave as being in public even though each mix-server should keep his permutation and randomizers secret. When a shuffle allows public verifiability, in general, by using zero-knowledge proofs, but requires a secret permutation and randomizers, Neff [Nef01] (and later Groth [Gro10]) call it a verifiable *secret shuffle*.

In TCC 2007, Adida and Wikström [AW07] 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 [OS05] for different purposes. 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 such as the BGN cryptosystem [BGN05] and the Paillier cryptosystem [Pai99], it should be generated by a trusted party. Thus, 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.

4.2 Generalized Shuffle

4.2.1 Syntax of Generalized Shuffle

Now we describe the syntax of a generalized shuffle.

As a symmetry of verifiable secret shuffles, 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 3.2.1. Although Adida et al. [AW07] and Parampalli et al. [PRT12] 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 4.2.1 (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 3.2.1 except for

– $\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$: The setup algorithm takes as input a security parameter λ and parameters $n, \ell \in \mathbb{N}$, and outputs a public parameter

$\delta := \left(pk, \{\sigma_j\}_{j=1}^{\ell}, \{\mathsf{T}_i\}_{i=1}^{n}\right)$ 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

$$\begin{aligned} &\{\mathsf{T}_1\left(\mathsf{Dec}_{sk}(\hat{c}_1), \ldots, \mathsf{Dec}_{sk}(\hat{c}_\ell)\right), \ldots, \mathsf{T}_n\left(\mathsf{Dec}_{sk}(\hat{c}_1), \ldots, \mathsf{Dec}_{sk}(\hat{c}_\ell)\right)\} \\ &\qquad = \{m_1, \ldots, m_n\} \\ &\qquad = \{\mathsf{Dec}_{sk}(c_1), \ldots, \mathsf{Dec}_{sk}(c_n)\}. \end{aligned} \tag{4.2.1}$$

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 with selecting a random permutation π as an additional secret parameter, then we obtain a standard shuffle defined in Definition 3.2.1. Note that in this case, $i, j \in [1, n]$.

4.2.2 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. Our security definition for shuffle also follows the definition given by [NSK04], but we need to slightly modify their security defintion since the secret parameter does not contain a private permutation any more.

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 n 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.

Security Definition

As mentioned in [NSK04], 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.

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. (3.2.1). A generalized shuffle relation $\tilde{\Phi}(x, w)$ is satisfied if the witness $w = (s_1, \ldots, s_\ell)$ demonstrates that

$$\exists\, (s_1, \ldots, s_\ell)\,, \forall\, j \in [1, \ell] : \hat{c}_j = \mathsf{ReRand}_{pk}\left(\sigma_j(c_1, \ldots, c_n), s_j\right). \tag{4.2.2}$$

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 **??**.

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

$$\exists(\sigma_1, \ldots, \sigma_\ell), \forall j \in [1, \ell] : \hat{c}_j = \sigma_j(c_1, \ldots, c_n) \tag{4.2.3}$$

where $x = (\delta, \mathbf{c}, \hat{\mathbf{c}})$ and $\delta, \mathbf{c}$ with $\hat{\mathbf{c}}$ defined as in Definition 4.2.1. 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. [NSK04] 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. [NSK04] is shown in Chapter 3.

Obviously a generalized shuffle however *does not take a 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.

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

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda)$

$\delta \leftarrow \mathsf{Setup}(\lambda, n, \ell)$;

$(\mathsf{state}, \pi_0, \pi_1, \mathbf{m}) \leftarrow \mathcal{A}_1^{\mathcal{O}_\mathsf{D}}(\delta, n, \ell)$ 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 $w = \epsilon, c_i = \mathsf{Enc}_{pk}\left(m_{\pi_b(i)}, r_i\right)$ with $b \xleftarrow{\$} \{0,1\}$;

$b' \leftarrow \mathcal{A}_2^{\mathcal{O}_\mathsf{D}}(\hat{\mathbf{c}}, \mathbf{c}, \mathsf{state})$;

where $\mathcal{O}_\mathsf{D}$ is the decryption oracle and the empty string is denoted by ϵ.

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.

4.2.3 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

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

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 [Bon98].

4.3 Constructions from Ring Homomorphic Encryption

In this section we provide two instantiations of generalized public shuffle using a *ring homomorphic* cryptosystem. That is, both 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. For example, the BGN cryptosystem [BGN05] is an example of $(\rho, 1)$ ring homomorphic cryptosystems.

4.3.1 Construction from $\left(\binom{n}{\lfloor n/2 \rfloor}, n-1\right)$-$\mathcal{E}$

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), 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 $1 \leq i \leq n$. 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

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

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

Lemma 4.3.1. *Assuming that there exists a ring homomorphic cryptosystem (n_1, n_2)-$\mathcal{E}$ that meets the conditions required in the construction above, our generalized shuffle scheme based on (n_1, n_2)-$\mathcal{E}$ is correct.*

Proof. 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 4.2.1 $\hat{c}_1, \ldots, \hat{c}_\ell$ can be regarded as a generalized shuffle of $c_1, \ldots, c_n$. □

4.3.2 Construction from $(1, n)$-$\mathcal{E}$

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 $\hat{c}_j \in \mathsf{Enc}_{pk}^{(1,n)}(B(\alpha_j))$, $1 \leq j \leq \ell$ 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 4.3.2. *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 $(1 \leq i \leq n)$ as running a polynomial

reconstruct algorithm and a factorization algorithm in turn, then he can easily see that the correctness condition – Eq. (4.2.1) holds.

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

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

Finally $\{m_1, \ldots, m_n\}$ can be recovered by using a factorization algorithm over the message space. □

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 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 $R = \mathbb{F}_p$.

For completeness we present the total complexity including a process recovering input plaintexts. Anyone who is authorized to decrypt should decrypt $\hat{c}_1, \ldots, \hat{c}_\ell$ and reconstruct the polynomial $B(x)$ of degree n with complexity ℓ $\mathsf{D} + O(n^2)$ $\mathsf{M}_{\mathbb{F}_p}$. Further, this incurs $O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ to factorize using Cantor-Zassenhaus algorithm [CZ81], if $R = \mathbb{F}_p$. Hence, the total complexity amounts to $O(\ell)(\mathsf{E} + \mathsf{D}) + O(n^2 \log p)$ $\mathsf{M}_{\mathbb{F}_p}$, 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 ℓ another ciphertexts.

4.4 Constructions from Group Homomorphic Encryption

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 [El 84]. We extend it to be secure against the malicious adversary and analyze its security. 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.

4.4.1 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 $y(t) = g(t)^x \mod \wp(t)$ where a secret key x

is randomly chosen from $[0, p^3 - 2]$, and publishes a public key $pk = \langle p, \mathbb{G}_{q_1 q_2}, g(t), y(t), \wp(t) \rangle$.

– $\mathsf{Enc}_{pk}(m(t))$: Encryption with the public key pk and message $m(t) \in \mathbb{G}_{q_1 q_2}$ proceeds as follows. First, a random value $r \in [0, p^3 - 2]$ is chosen. The ciphertext is then published as:

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

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

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

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 [BH62, BS62], 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 4.4.1. *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 $(g_1, g_1^a, g_1^b, g_1^c)$. We first choose a triple of random values $x, y, z \xleftarrow{\$} \mathbb{Z}_{q_1 q_2}$. Then compute

$$\left(g_1 g_2, g_1^a g_2^x, g_1^b g_2^y, g_1^c g_2^z\right),$$

and submit the quadruple to $\mathcal{D}$. According that $c = ab$ or c is a random value in $\mathbb{Z}_{q_1 q_2}$, the distinguisher will answer the query. Hence, if the output of $\mathcal{D}$ is 1, then $ab \equiv c \mod q_1$. 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:

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

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 4.4.1.

Keeping the Shuffler Honest without Zero-knowledge Proofs

One crucial 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 recompute the shuffler's output using public values.

4.4.2 A Generalized Public Shuffle Scheme Based on Polynomial Factorization

We begin with describing *extended* ElGamal encryption over $\mathbb{F}_{p^3}$. Then we present our public shuffle using the extended ElGamal encryption scheme which is also semantically secure assuming the DDH assumption in a cyclic subgroup of $\mathbb{F}_{p^3}^{\times}$ holds.

Extended ElGamal Encryption

Embedding our basic idea into constructing a generalized public shuffle scheme requires that we modify basic ElGamal encryption over $\mathbb{F}_{p^3}$ given as a building block above. We just describe modifications for extended ElGamal encryption over $\mathbb{F}_{p^3}$. According to modified parameters, its encryption and decryption algorithms should be modified as follows:

– *Modifying Key Generation.* We run $\mathsf{KG}(1^\lambda)$ 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 $y_j = \phi_j(y)$ for $j \in [1, \ell]$, and publish $pk = (g, y, \{y_i\}_{i=1}^{\ell}, \mathbb{G}_{q_1 q_2}, \wp(t), \{\wp_i(t)\}_{i=1}^{\ell}, \{\phi_i\}_{i=1}^{\ell})$ and keep a secret key $sk = x$.

– *Modifying Encryption and Decryption Algorithms.* We define ℓ-tuple ElGamal encryption by extending ElGamal encryption over $\mathbb{F}_{p^3}$. Given a message $m(t) \in \mathbb{F}_p[t]$, its encryption algorithm $\ell\text{-}\mathsf{Enc}_{pk}(\cdot)$ is defined as follows:

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

For decryption, first compute $\phi_j(g^r)$ and $m(t)^2 \equiv (\phi_j(g^r))^{-x} \cdot m(t)^2 \cdot y_j^r \mod \wp_j$. 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 shuffle using the ℓ-tuple ElGamal encryption scheme over extension fields.

$\mathsf{Setup}(1^\lambda, n, \ell)$**.** This algorithm is run by the shuffler and takes a security parameter λ and the input size n. It outputs a description of $\sigma : (\mathbb{G}_{q_1 q_2})^n \to \mathbb{G}_{q_1 q_2}$ given by $(c_1, \ldots, c_n) \mapsto c_1 \cdots c_n$ along with the public key pk, i.e., $\delta = (pk, \sigma)$.

$\mathsf{Shuffle}(\delta, \mathbf{c})$**.** 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 ElGamal ciphertext, given from

each sender S_i, proceeds as follows. Here $c_i \in \ell\text{-}\mathsf{Enc}_{pk}(m_i(t))$ and

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

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

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

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

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

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

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

$\mathsf{Verify}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$**.** Upon receiving this tuple, the verifier will first run the verification algorithm by non-interactively running $\mathcal{V}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$ – 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 4.4.1. *If the shuffler performs correctly the scheme, our public shuffle scheme is correct.*

Proof. We take 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

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

to $(\prod_{i=1}^n m_i(t))^2 \mod \wp_j(t)$, $1 \leq j \leq \ell$. He then computes $(\prod_{i=1}^n m_i(t))^2 \mod \wp_1(t) \cdots \wp_\ell(t)$ from each $(\prod_{i=1}^n m_i(t))^2 \mod \wp_j(t)$ 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.

Theorem 4.4.2. *Assuming the DDH assumption holds, our 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 4.1. 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 $c_\beta = \mathsf{Enc}_{pk}^{\mathsf{d}}(m_\beta^*, r^*)$ where β is a random bit chosen by the CCA1 challenger;
4. According to its random choice b,
$$c_j^* = \begin{cases} \mathsf{Enc}_{pk}^{\mathsf{d}}(m_{\pi_0(j)}^*, r_i^*) & \text{if } b = 0 \\ c_\beta & \text{if } b = 1 \end{cases}$$
5. For all $i = [1,n] \backslash \{j\}$, compute $c_i^* = \mathsf{Enc}_{pk}^{\mathsf{d}}\left(m_{\pi_b(i)}^*, r_i^*\right)$;
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 $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{PubShf}}(\tilde{\Phi}_{\mathcal{E}}, \lambda) = 1$ iff $b = b'$. We have

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

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

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

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

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

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

□

Theorem 4.4.3. *Assuming the DDH assumption holds, our public shuffle scheme is verifiable.*

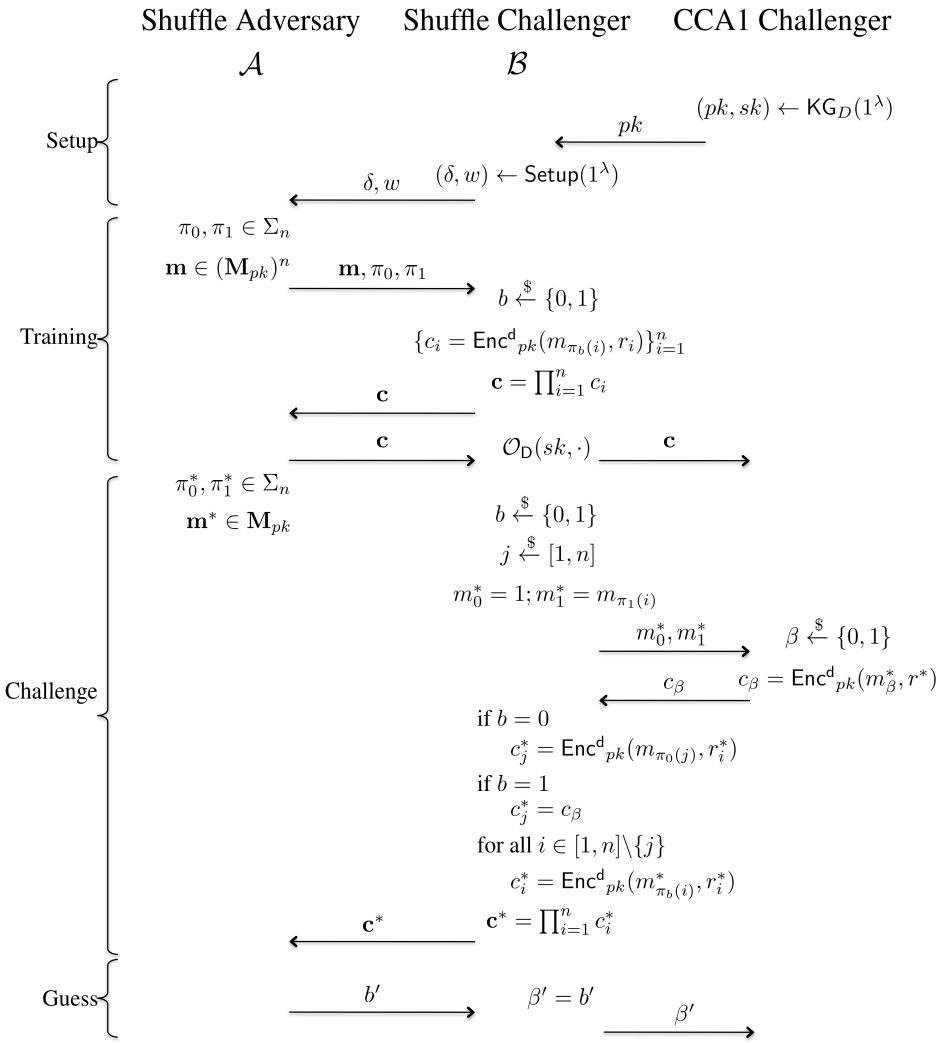

Figure 4.1: Graphical View of Security

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 $\phi_j(g(t)^{\sum_{i=1}^n r_i}) = g(\phi_j(t))^{\sum_{i=1}^n r_i}$, $1 \leq j \leq \ell$, with $O(\ell)$ $\mathsf{M}_{\mathbb{F}_p}$ using Horner's rule.

The decryption requires $O(\ell \log p)$ $\mathsf{M}_{\mathbb{F}_p}$ and $\prod_{i=1}^{n}(m_i(t))^2 \mod \prod_{i=1}^{\ell} \wp_i(t)$ is obtained by using a fast CRT in $O(\ell \log \ell)$ $\mathsf{M}_{\mathbb{F}_p}$. Solving square root of $\left(\prod_{i=1}^{n} m_i(t)\right)^2 \mod \prod_{i=1}^{\ell} \wp_i(t)$ 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 an ElGamal ciphertext $c = (u, v) = (g^r, my^r)$ under the public key y, 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 [Sho09, 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 [Sho09, Sec. 17.5].

4.4.3 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 4.4.2.

This construction uses ℓ-tuple ElGamal encryption extended by standard ElGamal encryption over prime fields. We just describe the differences compared to that used in Section 4.4.2: (1) Since a field isomorphism is not available, each user should send ℓ full ElGamal ciphertexts to a shuffler. Namely, the number of group elements transmitted by each sender is 2ℓ. 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. If no confusion arises, we abuse notation and use the same symbol for extended ElGamal encryption. The full description of ElGamal and its extension over prime fields are as follows.

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 [Hux72] 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 [Mai85] 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 $\kappa = \|m\|$.

The Construction

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

$\mathsf{Setup}(1^\lambda, n, \ell)$**.** This algorithm takes as input a security parameter λ and size parameters n, ℓ, outputs a public parameter $\delta = (pk)$.

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

1. The shuffler computes and outputs

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

with a proof $\Gamma = \epsilon$.

$\mathsf{Verify}(\delta, \mathbf{c}, \hat{\mathbf{c}}, \Gamma)$**.** 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 4.4.4. *If the shuffler performs correctly, our public shuffle scheme is correct.*

Proof. Suppose that the shuffler follows the above algorithm properly. We take each transformation T_i $(1 \leq i \leq n)$ 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 ElGamal ciphertexts $(\prod_{i=1}^{n} v_{ij}, \prod_{i=1}^{n} u_{ij})$ for $1 \leq j \leq \ell$, and

outputs

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

$M = (m_1 m_2 \cdots m_n)^2 \mod p_1 \cdots p_\ell$ 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 $(\prod_{i=1}^n v_{ij}, \prod_{i=1}^n u_{ij})$ 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 ℓ and the shuffler takes as input $n\ell$ ciphertexts and outputs ℓ ciphertexts.

Small Message Case. 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 $n = 10^4$.

Chapter 5

Conclusion and Further Work

In this dissertation, we describe the definition of verifiable secret shuffles, its security model, and limitations that they have. Then 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 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.

Bibliography

[Abe98] Masayuki Abe. Universally verifiable mix-net with verification work indendent of the number of mix-servers. In Kaisa Nyberg, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1403, pages 437–447, 1998.

[Abe99] Masayuki Abe. Mix-networks on permutation networks. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology-AsiaCrypt*, LNCS 1716, pages 258–273, 1999.

[AH01] Masayuki Abe and Fumitaka Hoshino. Remarks on mix-network based on permutation networks. In Kwangjo Kim, editor, *Public Key Cryptography*, LNCS 1992, pages 317–324, 2001.

[AW07] Ben Adida and Douglas Wikström. How to shuffle in public. In Salil Vadhan, editor, *TCC*, LNCS 4392, pages 555–574, 2007.

[BG12] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology-EuroCrypt*, LNCS 7237, pages 263–280, 2012.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC*, LNCS 3378, pages 325–341, 2005.

[BH62] Paul Bateman and Roger Horn. A heuristic asymptotic formula concerning the distribution of prime numbers. *Mathematics of Computation*, 16:363–367, 1962.

[Bon98] Dan Boneh. The decision Diffie-Hellman problem. In Joe Buhler, editor, *ANTS*, LNCS 1423, pages 48–63, 1998.

[BS62] Paul Bateman and Rosemarie 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.

[BY86] Josh Benaloh and Moti Yung. Distributing the power of a government to enhance the privacy of voters. In *PODC*, pages 52–62, 1986.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[CP92] David Chaum and Torben Pedersen. Wallet databases with observers. In Ernest Brickell, editor, *Advances in Cryptology-Crypto*, LNCS 740, pages 89–105, 1992.

[CZ81] David Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In Joan Feigenbaum, editor, *Advances in Cryptology-Crypto*, LNCS 576, pages 445–456, 1991.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In Kwangjo Kim, editor, *Public Key Cryptography*, LNCS 1992, pages 119–136, 2001.

[DK00] Yvo Desmedt and Kaoru Kurosawa. How to break a practical mix and design a new one. In Bart Preneel, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1807, pages 557–572, 2000.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router onion router. In *USENIX Security Symposium*, pages 303–320, 2004.

[El 84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. Blakely and David Chaum, editors, *Advances in Cryptology-Crypto*, LNCS 196, pages 10–18, 1984.

[FMM+02] Jun Furukawa, Hiroshi Miyauchi, Kengo Mori, Satoshi Obana, and Kazue Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In Matt Blaze, editor, *Financial Cryptography*, LNCS 2357, pages 16–30, 2002.

[FMS10] Jun Furukawa, Kengo Mori, and Kazue Sako. An implementation of a mix-net based network voting scheme and its use in a private organization. In David Chaum, Markus Jakobsson, Ronald Rivest, Peter Ryan, Josh Benaloh, Miroslaw Kutylowski, and Ben Adida, editors, *Towards Trustworthy Elections*, LNCS 6000, pages 141–154, 2010.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself. practical solutions to identification and signature problems. In Andrew Odlyzko, editor, *Advances in Cryptology-Crypto*, LNCS 263, pages 186–189, 1987.

[FS01] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In Joe Kilian, editor, *Advances in Cryptology-Crypto*, LNCS 2139, pages 368–387, 2001.

[Fur05] Jun Furukawa. Efficient and verifiable shuffling and shuffle-decryption. *IEICE transactions*, 88-A(1):172–188, 2005.

[GI08] Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In Nigel Smart, editor, *Advances in Cryptology-EuroCrypt*, LNCS 4965, pages 379–396, 2008.

[GL07] Jens Groth and Steve Lu. Verifiable shuffle of large size ciphertexts. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography*, LNCS 4450, pages 377–392, 2007.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Harry Lewis, Barbara Simons, Walter Burkhard, and Lawrence Landweber, editors, *STOC*, pages 365–377, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[Gro03] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In Yvo Desmedt, editor, *Public Key Cryptography*, LNCS 2567, pages 145–160, 2003.

[Gro10] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of Cryptology*, 23(4):546–579, 2010.

[GT03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the fiat-shamir paradigm. In *FOCS*, pages 102–113, 2003.

[Hux72] Martin Huxley. On the difference between consecutive primes. *Inventiones Math.*, 15:164–170, 1972.

[JJR02] Markus Jakobsson, Ari Juels, and Ronald Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security Symposium*, pages 339–353, 2002.

[Mai85] Helmut Maier. Primes in short intervals. *Michigan Mathematical Journal*, 32(2):221–225, 1985.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In Dan Boneh, editor, *Advances in Cryptology-Crypto*, LNCS 2729, pages 96–109, 2003.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[Nef03] C. Andrew Neff. Verifiable mixing (shuffling) of ElGamal pairs, 2003.

[NSK04] Lan Nguyen, Reihaneh Safavi-Naini, and Kaoru Kurosawa. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS*, LNCS 3089, pages 61–75, 2004.

[NSK06] Lan Nguyen, Reihaneh Safavi-Naini, and Kaoru Kurosawa. Verifiable shuffles: a formal model and a paillier-based three-round construction with provable security. *International Journal of Information Security*, 5(4):241–255, 2006.

[OS05] Rafail Ostrovsky and William Skeith III. Private searching on streaming data. In Victor Shoup, editor, *Advances in Cryptology-Crypto*, LNCS 3621, pages 223–240, 2005.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1592, pages 223–238, 1999.

[PBDV04] Kun Peng, Colin Boyd, Ed Dawson, and Kapalee Viswanathan. A correct, private, and efficient mix network. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *Public Key Cryptography*, LNCS 2947, pages 439–454, 2004.

[Ped91] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology-Crypto*, LNCS 576, pages 129–140, 1991.

[PRT12] Udaya Parampalli, Kim Ramchen, and Vanessa Teague. Efficiently shuffling in public. To appear in PKC'12, 2012.

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sho09] Victor Shoup. *A conputational introduction to number theory and algebra*. Cambridge University Press, 2nd edition, 2009.

[SK95] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme – a practical solution to the implementation of a voting booth. In Louis Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology-EuroCrypt*, LNCS 921, pages 393–403, 1995.

[Sta05] Heiko Stamer. Efficient electronic gambling: An extended implementation of the toolbox for mental card games. In Christopher Wolf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC*, LNI 74, pages 1–12, 2005.

[TW10] Björn Terelius and Douglas Wikström. Proofs of restricted shuffles. In Daniel Bernstein and Tanja Lange, editors, *Africacrypt*, LNCS 6055, pages 100–113, 2010.

[Wik02] Douglas Wikström. The security of a mix-center based on a semantically secure cryptosystem. In Alfred Menezes and Palash Sarkar, editors, *Indocrypt*, LNCS 2551, pages 368–381, 2002.

[Wik05] Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In Bimal Roy, editor, *Advances in Cryptology-AsiaCrypt*, LNCS 3788, pages 273–292, 2005.

[Wik09] Douglas Wikström. A commitment-consistent proof of a shuffle. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, LNCS 5594, pages 407–421, 2009.

[Wik10] Douglas Wikström. Verificatum. In *http://www.verificatum.com/*, 2010.

국문초록

본 논문은 공개 셔플 (Shuffle) 을 설계하고 구축하는 방법에 대해서 제시한다. 일반적으로 표준 셔플은 메시지를 셔플하는 주체가 랜덤 치환 (Permutation) 과 랜덤값을 안전하게 비밀로 유지하는 것을 요구한다. 그러나, 이러한 비밀값들을 항상 안전하게 유지할 수 있다고 기대하는 것은 경우에 따라 매우 강한 가정일 수 있으며, 심지어 현실에서는 적용할 수 없는 경우가 있다. 그래서 본 연구에서 이러한 가정을 약화시키면서 안전하게 셔플을 수행할 수 있는 방법에 대해서 제시한다.

셔플의 대표적인 응용인 믹스넷 (Mix-net) 은 익명화 (Anonymization) 을 위한 보편적 수단의 하나이다. 믹스넷은 입력된 암호화된 값을 치환하고 재암호화하거나 복호화하는 방법을 많이 이용된다. 만약 믹스넷을 구성하는 믹스서버가 적어도 하나 정직하게 행동한다면 입력과 출력의 연결성은 안전하게 숨겨질 것이다. 그래서 연결성이 안전하게 숨겨지는 것을 보장되기 위해서는 믹스를 담당하는 서버가 믹스 프로토콜을 따라 동작하도록 보장할 필요가 있다. 이를 위한 별도의 수단으로 영지식증명을 사용하는데, 영지식 증명은 연산량과 통신량의 복잡도가 높은 것으로 알려져 있다.

2007년 TCC에서 Adida와 Wikström에 의해서 공개적으로 셔플을 할 수 있는 안전성이 증명가능한 방법이 제시되었다. 그들의 방법은 셔플을 수행하는 주체가 일체의 비밀정보를 모르는 상태에서 셔플을 할 수 있도록 한다. 이러한 목적을 위해서 그들의 기법은 암호화된 치환행렬 (Permutation Matrix) 을 공개하면 셔플을 필요로 하는 사용자들이 공개된 치환행렬을 이용하여 암호화된 값들을 셔플할 수 있다. 즉석 (On-the-fly) 에서 셔플을 해야 하는 기존의 기법에 비해서 그들의 방법은 믹스넷을 구축할때 복호화하는 알고리즘과 복잡도만 고려하는 것을 허용한다. 그러나, 그들의 기법이 공개적으로 셔플을 하는 것을 가능하게 했지만, 랜덤 치환과 랜덤값

은 여전히 셔플을 생성하는 제 3의 신뢰기관이 알아야 하는 상황으로 변경되었을 뿐 비밀값 자체를 시스템에서 제거한 것은 아니다. 또한 셔플을 공개할 때 정당하게 생성된 셔플이라는 것을 증명하기 위해 여전히 영지식 증명도 필요하다.

본 논문에서는 기존 셔플–비밀셔플이나 Adida와 Wikström의 기법–과 달리 랜덤 치환과 랜덤값을 사용하지 않고 공개적으로 셔플을 허용하는 공개 셔플 (Public Shuffle) 을 제안한다: n개의 암호문 $(c_1, \ldots, c_n)$이 주어지면 우리가 제시한 기법은 $f_i(c_1, \ldots, c_n), 1 \leq i \leq \ell$, 을 계산한다, 여기서 $f_i(x_1, \ldots, x_n)$는 $x_1, \ldots, x_n$을 매개변수로 하는 대칭함수이다. 본 논문은 적용되는 대칭함수에 따라 크게 두 가지 설계가 가능함을 보인다. 첫번째는 환준동형 (Ring Homomorphic) 암호시스템을 사용하는 것이고 장점으로는 상수크기의 암호문만 요구한다는 것이다. 두번째는 군준동형 (Group Homomorphic) 암호시스템을 사용하는 것으로 사용자수에 선형 크기의 암호문 개수를 요구한다. 그러나 두 기법 모두 랜덤 치환이나 랜덤값을 요구하지 않으며, 아울러 영지식증명을 사용하지 않고 공개적으로 셔플이 옳게 이루어졌는지 확인할 수 있는 방법을 제공한다.

주요어휘: 셔플, 검증가능한 비밀셔플, 공개셔플, 믹스넷, ElGamal 암호
학번: No. 20008-30081

감사의 글

아직도 많이 부족한 저의 지도교수가 되어 주시고, 박사학위기간 동안 공부하고 연구할 수 있도록 물심양면(物心兩面)으로 지원하고 가르침을 주신 천정희(千丁熙) 선생님께 먼저 진심으로 감사드립니다. 바쁘신 와중에 심사위원으로 참여하여 주신 김명환 선생님, 이인석 선생님, 이향숙 선생님, 특히 먼곳에서 기꺼이 참여하여 주신 김용대 선생님께 감사드립니다.

암호학적 난제 연구단의 모든 동료들도 고맙습니다: 김민규, 김성욱, 김홍태, 이형태, 김태찬, 류한솔, 김진수, 홍현숙, 정희원, 그리고 이주은씨: 모두 보고 싶을 겁니다! Minnesota에서 지내는 동안 여러가지 도움을 준 박동철, Abadelaziz Mohaisen에게도 감사의 마음을 전합니다.

함께 연구하고 여러가지 助言을 아끼지 않았던 입학동기 이형태와 박사학위기간 내내 성실하고 열정적인 모습으로 자극을 준 배영진, 그리고 사소한 일까지 챙겨준 팀장 김태찬에게 이 글을 빌어 고맙다는 말을 전합니다. 또한 자연스런 삶을 모습을 보여주시는 형석형님과 대인이신 종승형님께도 감사하는 마음을 잊을 수 없습니다. 함께 있는 것만으로도 힘이 되는 친구들–재철,재형과 문선도 고맙다는 말을 받아야 합니다. 열심히 살고 있는 학수와 지영도 마찬가지입니다. 힘들때 이들의 그늘이 있어 쉬었다 갈 수 있었습니다.

끝으로 공부하는 동안 내내 애정과 격려의 말로 묵묵히 기다려준 제 아내, 은영에게 온 마음으로 고맙다는 말을 하고 싶습니다. 또한 함께 힘들어 하신 어머니와 누이–세종엄마, 그리고 묵묵히 지켜봐 주신 장인과 장모님께도 감사드립니다.

學而時習之면 不亦說乎아!

《論語-學而》