Multiparty Computation to Generate Secret Permutations

Chris Studholme and Ian Blake
University of Toronto

September 6, 2007

Abstract

We make use of a universal re-encryption mixnet to efficiently perform a secure multiparty computation to generate a secret permutation. When complete, the permutation is shared among the players in such a way that each player knows his share of the permutation but no others. Such a permutation is useful in dining cryptographers networks (DC-nets) to determine in which slot each player should transmit. We also see this primitive as being useful in online gaming for either shuffling cards or ordering players without the need for a trusted dealer or other third party.

Keywords: secure multiparty computation, secret permutation, re-encryption, ElGamal encryption

1 Introduction

The multiparty computation described in this paper allows n players to choose a random permutation, of themselves or some other set, and share that permutation among themselves in such a way that each player knows only their part of the permutation. Since the permutation is not completely secret, we consider this type of sharing a weak form of secret sharing. If a group of players collude, they will learn each others' shares of the permutation and they can narrow down the possibilities for the remaining shares, but as long as at least two players are honest, the coalition will be unable to learn the entire permutation.

We envision two primary uses of this secret permutation generation technique. The first is to generate a secret ordering of the players participating in some form of anonymous message delivery system. In both a dining cryptographers network (DC-net) [3] and a public decryption shuffle [1] the participating players must know which slot their message is to appear in after decryption. Indeed, in "Dining Cryptographers Revisited", Golle and Juels [8] make the following comment regarding collisions and the possibility of generating a secret permutation:

> "The problem can be avoided through techniques like secure multiparty computation of a secretly distributed permutation of slots among players, but this is impractical."

Our contribution is an interactive protocol to generate the required secretly distributed permutation which has complexity that is, to within a constant, optimal. Adida and Wikström describe in Section 7 of [1] an alternate method of generating the required permutation; however, we note that their method requires that an encrypted $n \times n$ matrix be passed from one player to the next. Since our method involves only passing an n-vector, our protocol has a factor of n lower complexity.

The other potential use of our protocol is in the ordering of players or in making other random choices in online games. Sweeney and Shamos consider this problem in the non-secret setting [9].

While not many games require a secret random ordering of the players, we imagine that with an efficient protocol for generating such orderings, perhaps such a game might be designed in the future. Our protocol is useful, however, for one common component of many games, the shuffling of cards.

Our approach to generating a secret permutation is to construct a mixnet where each of the n parties is also a node in the mixnet. The parties take turns shuffling and re-encrypting an n-vector of ciphertexts, and the final order of these ciphertexts determines the permutation. The protocol is complicated both by the need to ensure that each party has a chance to shuffle the vector (i.e. malicious parties cannot route around the honest ones) and that no party can recognize their own ciphertext until the final ordering has been fixed.

Other possible approaches to generating this sort of permutation include having each party choose some large number at random and then comparing those numbers using a multiparty greater than protocol. Care would need to be taken to ensure a party cannot unfairly influence their share of the permutation by making a non-uniform choice. To be successful, $\mathcal{O}(n \log n)$ instances of the greater than protocol would need to be performed, and in the end the permutation would not be entirely secret. A better approach might be to make use of a k^{th}-ranked element computation [2]. While each party would learn what the k^{th}-ranked element is, no one except the holder of that element would know who has it. Unfortunately, with each k^{th}-ranked element computation requiring $\log n$ rounds with overhead $\mathcal{O}(n \log n)$ per round, the total complexity would be $\mathcal{O}\left(n^2 \log^2 n\right)$. We will show that our protocol has a communication and computation complexity of $\mathcal{O}(n \log n)$ per player.

In the next section we review the ElGamal cryptosystem and universal re-encryption. Then, in Section 3, we present our secret permutation sharing protocol and in Section 4 we prove several security properties. Finally, in Section 5, we discuss the application of our protocol to games.

2 ElGamal Cryptosystem

At the core of our secret permutation sharing scheme is a mixnet utilizing the ElGamal probabilistic public key encryption scheme [5]. Let G be some cyclic group $\langle g \rangle$ generated by $g \in G$ and let $q = |G|$. We use the operator $\in_R$ to denote a uniform random selection. The three essential algorithms provided by the ElGamal cryptosystem are:

- **Key generation**: Output $(PK, SK) = (y = g^x, x)$ for random $x \in_R \mathbb{Z}_q$.
- **Encryption**: Input comprises a message $m \in G$, a public key y and a random encryption factor $r \in_R \mathbb{Z}_q$. The output is a ciphertext $C = (\alpha, \beta) = (my^r, g^r)$.
- **Decryption**: Input is a ciphertext $C = (\alpha, \beta)$ under public key y and the corresponding secret key x. The output is plaintext $m = \alpha/\beta^x$.

In addition to these basic operations, a re-encryption algorithm is also commonly used. Re-encryption takes as input a ciphertext C and the public key the ciphertext was encrypted under y, and then outputs an alternate ciphertext C' that is an encryption of the same plaintext message under the same public key.

Golle, et. al., extend the ElGamal cryptosystem to provide an algorithm for doing universal re-encryption [7]. Their new algorithms require a two-fold increase in ciphertext size but allow re-encryption without knowledge of the public key. To encrypt they choose a random encryption factor $r = (r_0, r_1) \in_R \mathbb{Z}_q^2$ and then form the ciphertext (4-tuple) $C = [(my^{r_0}, g^{r_0});\ (y^{r_1}, g^{r_1})]$. To re-encrypt the ciphertext $C = [(\alpha_0, \beta_0);\ (\alpha_1, \beta_1)]$, choose a random re-encryption factor

$r' = (r'_0, r'_1) \in_R \mathbb{Z}_q^2$ and compute $C' = [(\alpha_0 \alpha_1^{r'_0}, \beta_0 \beta_1^{r'_0});\ (\alpha_1^{r'_1}, \beta_1^{r'_1})]$. Decryption is the same as it is for standard ElGamal (using only the first 2-tuple), but it allows one to also verify that the ciphertext was encrypted using the correct public key. Such verification is done by decrypting the latter 2-tuple and checking for the identify element.

As will be seen in the next section, our protocol makes use of universal re-encryption; however, as will also be seen, we are only interested in recognizing ciphertexts encrypted under a certain key and not in sending messages. Therefore, all of our ciphertexts are encryptions of the identity element message. Since the latter half of a Golle ciphertext 4-tuple is actually an encryption of the identity element, we can define a universal re-encryption algorithm for use on standard ElGamal encryptions of the identity element, thus avoiding the two-fold increase in ciphertext size. We propose the following algorithm:

- **Universal identity re-encryption**: Input is a ciphertext $C = (\alpha, \beta)$ and a random re-encryption factor $r' \in_R \mathbb{Z}_q$. The output is an alternate ciphertext $C' = (\alpha^{r'}, \beta^{r'})$.

If this re-encryption operation is performed on a ciphertext that is not an encryption of the message $m = 1$, the re-encryption will corrupt the message — preventing successful decryption.

For secret key x, we define the set of all possible encryptions of the identity element under the public key g^x as $\mathcal{E}(x) = \{\ (\alpha, \beta) \mid \alpha = \beta^x\ \}$. The ciphertext $(1, 1)$ is an element of every such set, but without this element, the sets are disjoint. Furthermore, for every C with $\beta \neq 1$, there exists at most one x for which $C \in \mathcal{E}(x)$.

The final operations we need are a means of altering the key under which a ciphertext is encrypted. The following two algorithms accomplish this:

- **Key addition**: Input is a ciphertext $C = (\alpha, \beta)$ and an offset δ. The output is a ciphertext message $C' = (\alpha\beta^\delta, \beta)$, which is an encryption of the same plaintext message but now under public key $y' = yg^\delta$, where y is the public key C is encrypted under.

- **Key product**: Input is a ciphertext $C = (\alpha, \beta)$ and a coefficient c. The output is a ciphertext message $C' = (\alpha^c, \beta)$. If C is an encryption of m under the public key y, then C' is an encryption of m^c under the public key $y' = y^c$.

Note that while key addition can be used on any ciphertext, key product will alter the message if the ciphertext is not an encryption of the identity element. These operations can be performed without knowledge of the key a ciphertext is encrypted under, but they will not provide any information about said key. Also, if a message C is encrypted under g^x, then key addition with an offset of $-x$ will decrypt the message, yielding (m, β).

Key product will only be used to **negate the key** of a ciphertext. If a ciphertext $C \in \mathcal{E}(x)$, then performing key product with $c = -1 \bmod q$ will yield a ciphertext $C^- \in \mathcal{E}(-x)$. This operation can be used to turn any black box performing key addition with (secret) offset δ into a black box which performs key addition with offset $-\delta$. Simply negate the key of the ciphertext before input into the black box and again after output.

Note that the key addition operation can be thought of as a partial encryption or decryption operation and is closely related to the threshold decryption techniques described by Desmedt and Frankel [4].

2.1 Security of ElGamal

ElGamal is known to have *semantic security* if the group G is one for which the *Decisional Diffie-Hellman (DDH) assumption* holds. Semantic security is a property that limits an adversary's ability to derive information about a plaintext message from its corresponding ciphertext.

Typically, instead of testing a cryptosystem for semantic security, one tests for *ciphertext indistinguishability*, a property that has been shown to be equivalent to semantic security [6]. This latter property asserts an adversary's inability to determine which of two plaintext messages, chosen by him, has been given back to him in the form of a ciphertext message.

The DDH assumption asserts that no computationally bounded adversary can distinguish between the distributions (g^x, g^y, g^{xy}) and (g^x, g^y, g^z), where $x, y, z \in_R \mathbb{Z}_q$. Alternatively, given (g^x, g^y, g^z), the adversary cannot determine if $z = xy$. By *computationally bounded* we mean an adversary who runs in time polynomial in κ, a security parameter. For this DDH assumption, and ElGamal, we choose the group G such that $q = \mathcal{O}(2^\kappa)$.

Semantic security limits an adversary's ability to derive information about a message from its corresponding ciphertext; however, since the ciphertexts we are interested in are all encryptions of the identity element, semantic security has little meaning in our setting. Instead, we provide a definition of *key indistinguishability* (KI). Informally, this is the inability of an adversary to distinguish between ciphertexts encrypted with distinct public keys.

To test an adversary, as defined by the algorithm $\mathcal{A}$, later referred to as an *adversarial algorithm*, we define an experiment for KI as follows. Two private ElGamal keys are generated along with two random encryption exponents. Then, an encryption of the identity element is formed using each key and the corresponding random exponent. The adversary is given these two ciphertexts in some randomly chosen order along with the two public keys and asked to guess the order of the ciphertexts. If the adversary guesses correctly the experiment terminates with an output of '1'. Otherwise the output is '0'. As mentioned above, ElGamal is parametrized under the security parameter κ.

Experiment 1. $Exp_{\mathcal{A}}^{KI}(EG, \kappa)$

$u_0, u_1 \in_R \mathbb{Z}_q$;
$r_0, r_1 \in_R \mathbb{Z}_q$;
$C_0 \leftarrow (g^{u_0 r_0}, g^{r_0})$; $C_1 \leftarrow (g^{u_1 r_1}, g^{r_1})$;
$\mathfrak{b} \in_R \{0, 1\}$;
$\mathfrak{b}' \leftarrow \mathcal{A}(g^{u_0}, g^{u_1}, C_{\mathfrak{b}}, C_{1-\mathfrak{b}}, \text{“guess”})$;
if $\mathfrak{b} = \mathfrak{b}'$ **then** output '1' **else** output '0' **fi**

Definition 1. *The ElGamal cryptosystem EG provides KI if for any adversary $\mathcal{A}$ with resources polynomial in κ the probability $pr[Exp_{\mathcal{A}}^{KI}(EG, \kappa) = \text{'1'}] - 1/2$ is negligible in κ.*

Theorem 1. *Under the DDH assumption, ElGamal provides KI.*

Proof: By contradiction, assume adversary $\mathcal{A}$ is successful in breaking KI. We will show how this adversary can be used to break the DDH assumption.

Consider the following slightly altered experiment:

$Exp_{\mathcal{A}}^{altki}(EG, \kappa)$
$u_0, u_1, u_1' \in_R \mathbb{Z}_q$;
$r_0, r_1 \in_R \mathbb{Z}_q$;
$C_0 \leftarrow (g^{u_0 r_0}, g^{r_0})$; $C_1 \leftarrow (g^{u_1 r_1}, g^{r_1})$;
$\mathfrak{b} \in_R \{0, 1\}$;
$\mathfrak{b}' \leftarrow \mathcal{A}(g^{u_0}, g^{u_1'}, C_{\mathfrak{b}}, C_{1-\mathfrak{b}}, \text{“guess”})$;
if $\mathfrak{b} = \mathfrak{b}'$ **then** output '1' **else** output '0' **fi**

The only difference between this experiment and Experiment 1 is that, in the latter, one of the public keys passed to the adversary is corrupt (alternatively, one of the ciphertexts is

corrupt). The adversary may, however, still have sufficient information to determine $\mathfrak{b}$. This gives us two cases to consider.

Case 1: $pr[Exp_{\mathcal{A}}^{altki}(EG, \kappa) = \text{'1'}] - 1/2$ is negligible in κ. The adversary is not capable of handling corrupt parameters. We can construct an adversary $\mathcal{A}'$ successful against DDH as follows. Recall that the DDH test is to determine if $z = xy$.

funct $\mathcal{A}'(g^x, g^y, g^z, \text{“guess”}) \equiv$
$u_0, r_0 \in_R \mathbb{Z}_q$;
$C_0 \leftarrow (g^{u_0 r_0},\ g^{r_0})$; $C_1 \leftarrow (g^z,\ g^y)$;
$\mathfrak{b} \in_R \{0, 1\}$;
$\mathfrak{b}' \leftarrow \mathcal{A}(g^{u_0}, g^x,\ C_{\mathfrak{b}}, C_{1-\mathfrak{b}},\ \text{“guess”})$;
if $\mathfrak{b} = \mathfrak{b}'$ **then** output '1' **else** output '0' **fi.**

If $z = xy$ then $\mathcal{A}$ will see parameters constructed as in Experiment 1; however, if $z \neq xy$ the ciphertext C_1 will not be a properly formed encryption under the key g^x and $\mathcal{A}$ will be unable to determine $\mathfrak{b}$.

Case 2: $pr[Exp_{\mathcal{A}}^{altki}(EG, \kappa) = \text{'1'}] - 1/2$ is not negligible in κ. In this case the adversary is clever enough to handle partially corrupt input, however, we can still construct an adversary $\mathcal{A}'$ successful against DDH.

funct $\mathcal{A}'(g^x, g^y, g^z, \text{“guess”}) \equiv$
$u_1, u_1', r_1 \in_R \mathbb{Z}_q$;
$C_0 \leftarrow (g^z,\ g^y)$; $C_1 \leftarrow (g^{u_1 r_1},\ g^{r_1})$;
$\mathfrak{b} \in_R \{0, 1\}$;
$\mathfrak{b}' \leftarrow \mathcal{A}(g^x, g^{u_1'},\ C_{\mathfrak{b}}, C_{1-\mathfrak{b}},\ \text{“guess”})$;
if $\mathfrak{b} = \mathfrak{b}'$ **then** output '1' **else** output '0' **fi.**

If $z = xy$ then $\mathcal{A}$ will see partially corrupt parameters yet does have an advantage when guessing $\mathfrak{b}$. When $z \neq xy$, no information about $\mathfrak{b}$ is passed to $\mathcal{A}$, and therefore, the adversary cannot possibly determine $\mathfrak{b}$ with a probability significantly greater than 1/2.

These constructions prove that KI follows from the DDH assumption. ■

We also note that key addition with offset δ cannot be performed if one only has g^δ. This is obvious if one recalls that, for $C \in \mathcal{E}(u)$, key addition with offset $-u$ is equivalent to decryption and that g^{-u} can be easily computed from g^u. If one could do this, one could decrypt using only the public key.

Theorem 2. *Assuming KI and given a ciphertext $C \in \mathcal{E}(u)$ and a public key g^x, for any adversary $\mathcal{A}$ with resources polynomial in κ the probability that $\mathcal{A}$ outputs a ciphertext $C' = (\alpha', \beta')$ for which $\beta' \neq 1$ and $C' \in \mathcal{E}(u + x)$ is negligible in κ.*

Proof: Given any ciphertext C and a public key g^u, a successful adversary $\mathcal{A}$ can determine if $C \in \mathcal{E}(u)$ as follows. Choose $\delta \in_R \mathbb{Z}_q$ and compute $g^{-u+\delta}$. Then use $\mathcal{A}$ to compute key addition with offset $-u + \delta$. If this last step is successful, the resulting ciphertext will be an element of $\mathcal{E}(\delta)$ (which is easily checked) if and only if $C \in \mathcal{E}(u)$. Since $\mathcal{A}$ is not always successful but is assumed to be successful with a probability that is non-negligible in κ, the above steps must be repeated many, but no more than polynomial in κ, times to attain confidence in the result. If any of these repetitions results in an element of $\mathcal{E}(\delta)$, we assume that $C \in \mathcal{E}(u)$. A test of this form is sufficient to break KI, thus proving the theorem. ■

Corollary 1. *Given as input ciphertexts $C_1 \in \mathcal{E}(u_1)$ and $C_2 \in \mathcal{E}(u_2)$, for any adversary $\mathcal{A}$ with resources polynomial in κ the probability that $\mathcal{A}$ outputs a ciphertext $C' = (\alpha', \beta')$ for which $\beta' \neq 1$ and $C' \in \mathcal{E}(u_1 + u_2)$ is negligible in kappa.*

Finally, we define a generalized form of KI, called *generalized oracle key indistinguishability* (GOKI). KI is generalized in two ways: there are $h \geq 2$ keys and the adversary is provided with an oracle that can recognize ciphertexts encrypted under one of these keys. The following experiment tests an adversary $\mathcal{A}$ for GOKI. Note that $\pi(C_1, \ldots, C_h)$ produces a permuted list where C_i is located in position $\pi(i)$.

Experiment 2. $Exp_{\mathcal{A}}^{GOKI}(EG, \kappa, h)$
$u_1, \ldots, u_h \in_R \mathbb{Z}_q$;
$r_1, \ldots, r_h \in_R \mathbb{Z}_q$;
for $i := 1$ **to** h **do**
  $C_i \leftarrow (g^{u_i r_i}, g^{r_i})$;
od
$\pi : [1 \ldots h] \rightarrow [1 \ldots h] \longleftarrow$ random permutation;
$(p, q) \leftarrow \mathcal{A}(g^{u_1}, \ldots, g^{u_h}, \pi(C_1, \ldots, C_h), \mathcal{O}_{u_1,\ldots,u_h}$, "guess");
if $p = \pi(q)$ **then** output '1' **else** output '0' **fi**

The oracle $\mathcal{O}_{u_1,\ldots,u_h}$, on input a ciphertext $C = (\alpha, \beta)$, determines if $\alpha = \beta^{u_i}$ for some i. If index i exists, the oracle outputs '1', otherwise it outputs '0'. Obviously, the oracle does not reveal which key matched.

Definition 2. *The ElGamal cryptosystem EG provides GOKI if for all h and any adversary $\mathcal{A}$ with resources polynomial in κ, the probability $pr[Exp_{\mathcal{A}}^{GOKI}(EG, \kappa, h) = \text{'1'}] - 1/h$ is negligible in κ.*

Conjecture 1. *Under the DDH assumption, ElGamal provides GOKI.*

3 Secret Permutation Sharing

Before stating the protocol, it is useful to have in mind, at least informally, the properties this protocol seeks to provide. They are:

- **Correctness.** Each of the n players outputs a unique share of the secret permutation (say, an integer in $\{1, \ldots, n\}$). Against a malicious adversary we require that either each of the honest players outputs a unique share or at least one outputs a distinct failure signal.
- **Privacy.** No coalition of $n - 2$ or fewer players can learn the complete permutation. A coalition will know which shares are held by the honest players, but as long as there are at least 2 honest players, the coalition will be unable to determine the mapping of shares held by honest players to those players with a probability greater than that of random guessing.
- **Uniformity.** The positions of the honest players in the final permutation are distributed uniformly.

After stating the protocol we will define these properties formally and prove that they hold under suitable assumptions against an honest but curious (semi-honest) adversary. The protocol, as stated, is not secure against a malicious adversary, but we will discuss what is required to make it secure against such an adversary and describe our efforts to prove security in this case.

Assume $g \in G$ and $q = |G|$ are publicly known. The players are initially in some order and numbered 1 to n. The protocol for generating a permutation of n items is as follows:

1. Each player chooses at random two private keys $x_i \in_R \mathbb{Z}_q$ and $u_i \in_R \mathbb{Z}_q$. The public key g^{u_i} is sent to Player 1.

2. Player 1 uses the n public keys to form an initial list of ciphertexts $[(g^{u_1}, g), \ldots, (g^{u_n}, g)]$. Note that each of these ciphertexts is an encryption of the message $1 \in G$ under the corresponding public key.

3. Each player in turn, starting with Player 1, will perform key addition with offset x_i on each ciphertext, re-encrypt each ciphertext and shuffle the list. Suppose Player i is processing the ciphertext $C_j = (\alpha_j, \beta_j)$. The player will, with random re-encryption factor $r_j \in_R \mathbb{Z}_q$, compute
$$\bar{C}_j = (\alpha_j^{r_j} \beta_j^{x_i r_j},\ \beta_j^{r_j}) \ . \tag{1}$$
After processing each ciphertext in this manner, a random permutation is chosen and the list of ciphertexts is reordered. The new list is then sent to the next player for similar processing.

4. After Player n has completed the above step, she broadcasts the list of ciphertexts to all players.

5. After receiving the list, each player broadcasts their x_i value to all players.

6. Finally, each player will attempt to decrypt all messages using the secret key $w_i = u_i + \sum_j x_j$. If Player i locates exactly one ciphertext for which $\alpha/\beta^{w_i} = 1$, $\beta \neq 1$, then the player takes the position of this message within the list as his share of the secret permutation.

If one wishes a permutation on a larger number of items, say a multiple of n, each player can simply act as multiple players; however, to improve efficiency slightly, each player only needs to choose a single x_i and perform Step 3 once.

3.1 Complexity

The computational complexity of the protocol is $\mathcal{O}(n\kappa)$ group operations per player ($\mathcal{O}(n)$ exponentiations), while the communication complexity is $\mathcal{O}(n\kappa)$ bits per player ($\mathcal{O}(n)$ group elements). This assumes that in Step 4 the final list is either efficiently broadcast or passed through the players in a daisy chain manner (for example, from one player to the next in the reverse order of Step 3).

To ensure the probability that two players choose the same key is negligible, the key space must have at least n^2 elements. This implies that the security parameter κ must be at least proportional to $\log n$, and thus, the communication complexity is $\mathcal{O}(n \log n)$ bits per player.

Consider the communication costs associated with either the players deciding on a non-secret permutation or having a third party choose the permutation and send it to each player. Encoding a permutation of n items requires $\mathcal{O}(n \log n)$ bits. In this sense, our permutation generation protocol is optimal (to within a constant).

Of course, if a third party were to generate the permutation and only send each player his share of the permutation, the communication complexity would be $\mathcal{O}(\log n)$ per player, but $\mathcal{O}(n \log n)$ for the third party. In this sense, our protocol is not quite optimal.

4 Security

We consider 3 types of adversary:

1. **Global passive.** The global passive adversary has access to all communication and wishes to learn something about the generated permutation.

2. **Semi-honest player.** This player follows the protocol as stated but attempts to learn any additional information she can from the messages she sees and the secrets she has.

3. **Malicious adversary.** This adversary has complete control over one or more players, and as such, may not follow the protocol. Of course, if any other player detects a problem with the messages sent out by the adversary, they may terminate the protocol. The goal of the malicious adversary is to break one of the security properties while avoiding premature termination.

All players are computationally bounded and run in time polynomial in the security parameter κ. Furthermore, we assume that all communications are secure in the sense that if one honest player sends/broadcasts a message to some other honest player, the message will arrive intact.

Our protocol does not require that any of the communication be private. That is, all communication can be considered to be broadcast to all players. Because of this, the above list is totally ordered on the strength of the adversary and proof of security against a malicious adversary implies security against the others.

We now formally define each of the three desired security properties, prove our SPG is secure against a semi-honest adversary, and discuss security against a malicious adversary.

4.1 Correctness

Player i, assumed to be following the protocol, will take his position in the final permutation to be j if and only if the j^{th} ciphertext in the final list, say $C_j = (\alpha_j, \beta_j)$, has the property that $\alpha_j = \beta_j^{w_i}$ and $\beta_j \neq 1$. Recall that $w_i = u_i + \sum_k x_k$. We say an *invalid permutation* has been selected if Player i and Player i', $i \neq i'$, take the same final position j. This can occur if and only if one of the following holds: $w_i = w_{i'}$ or the two players see a different ciphertext at position j, say $C_j \neq C'_j$, respectively.

The probability that $u_i = u_{i'}$ is negligible in the security parameter so we assume this is not the case. The sum of the x_k can only be different (for different players) if when broadcasting x_k in Step 5 some player sends different values to different players. This cannot happen in the case of a semi-honest adversary or in the case where a reliable broadcast channel is used. For the case where the adversary is malicious and uses point-to-point transmission of messages, we discuss below the changes necessary to ensure all players receive the same x values, thus preventing the adversary from causing an invalid permutation to be generated in this manner.

We also note that all players will see the same ciphertext in a given position, say ciphertext C_j in position j, both in the case of a semi-honest adversary and in the case where a reliable broadcast channel is used in Step 4. If a malicious adversary is present and the final list is distributed via point-to-point transmission (either directly from Player n or daisy chained through all players) we suggest a simple verification of the final list. Given a collision resistant hash function[1] $\mathcal{H} : G^{2n} \to G$, broadcast in Step 5 both x_i and the hash value computed by applying $\mathcal{H}$

[1]For example, apply `SHA-512` to a bit-string representation of the element of the domain, and then map the result x to an element of G by treating x as an integer and computing g^x. To scale with κ, substitute a hash function with a sufficiently large range for `SHA-512`.

to the final list received in Step 4. Then, in Step 6, every player verifies that the $n-1$ hash values they received match the hash value they computed. If the values do not match, the permutation generation fails.

4.2 Privacy

To formally define privacy we first design an experiment which tests an adversary's ability to correctly determine an honest player's share of the permutation. We assume that either the adversary is semi-honest and may obtain the secret material held by an arbitrary subset of the players or the adversary is malicious and may control an arbitrary subset of the players. The following experiment could be simplified somewhat if only a semi-honest adversary were being considered, but we wish to use the same definition for both types of adversary and will consider the proof of privacy in the semi-honest case to be a sketch of the proof for the malicious adversary case.

The experiment that follows is intended to simulate the protocol but we have made a few simplifications which are justified as follows:

- Since the order in which the players process the list in Step 3 is arbitrary, we may assume that both Player 1 and Player n are adversary controlled. This gives the adversary the most power. In addition to this, we will assume that some number of adversary controlled players go first, then all of the honest players, and finally the remaining adversary controlled players. Furthermore, the actions of these groups of players may be aggregated. Note that even in the case where there are one or zero adversary controlled players, we let the adversary process the list both first and last.

- In aggregating the actions of the honest players in Step 3 we let $x = \sum x_i$.

- We force the adversary's x_i values to sum to zero. The adversary may initially perform key addition with some non-zero offset and then later undo this by performing key addition with the additive inverse. This does not weaken a semi-honest adversary as the adversary may also perform key addition with any desired offset on any private copies of ciphertexts as they desire.

- Forcing the adversary's x_i values sum to zero does weaken a malicious adversary against the protocol as stated. If such an adversary is able to choose his x_i values as a function of the honest player's x_i values, the adversary will have control over the sum $\sum_j x_j$ computed in Step 6 and can break privacy. To prevent this, we describe later how a non-mailable commitment scheme can be used to ensure that the value of the sum $\sum_j x_j$ is distributed uniformly. For the purposes of this definition, we simply assume this is the case.

- Finally, in aggregating the honest players, we assume they all see the same final list. This list must have either been distributed using reliable broadcast or a hash function has been employed as described in the correctness section above.

With these simplifications in mind, we now design an experiment for a probabilistic adversarial algorithm $\mathcal{A}$. Assume the secret permutation generator is parametrized by κ, the security parameter, n, the number of players, and h, the number of honest players. In the case of a semi-honest adversary, $n - h$ is the number of players open to the adversary. We restrict the adversarial algorithm to running in time polynomial in κ.

The experiment proceeds as follows. First the honest players choose their secret keys and send the public keys to the adversary. The adversary generates and returns a "first list" of ciphertexts

along with two polynomial time algorithms to be invoked later. This first list includes the actions of the adversary in Step 3 of the protocol. The experimenter then performs Step 3 on behalf of the honest players, using the aggregated x, and passes the resulting list to the adversary provided algorithm A which generates the "final list" of ciphertexts. This final list is tested to ensure it is valid, i.e. it contains exactly one ciphertext for each honest player. If the list is not valid, the experiment terminates with the special output value '$\perp$'. This output value distinguishes this outcome from a failed guess, which produces an output of '0'. Finally, after being given the value of x, the other adversary provided algorithm, A', guesses an honest player mapping, and if this guess is correct, the output is '1'. Note that $\pi(\bar{C}_1, \ldots, \bar{C}_n)$ produces a permuted list where $\bar{C}_i$ is located in position $\pi(i)$.

Experiment 3. $Exp_{\mathcal{A}}^{priv}(SPG, \kappa, n, h)$

for $i := 1$ **to** h **do**
  $u_i \in_R \mathbb{Z}_q$;
od
$(C_1, \ldots, C_n;\ A, A') \longleftarrow \mathcal{A}(g^{u_1}, \ldots, g^{u_h})$; // "first list"
$x \in_R \mathbb{Z}_q$;
for $i := 1$ **to** n **do**
  if $\alpha_i \notin G$ **or** $\beta_i \notin G$ **or** $\beta_i = 1$ **then** output '$\perp$' **fi** // $C_i = (\alpha_i, \beta_i)$
  $r_i \in_R \mathbb{Z}_q$;
  $\bar{C}_i \longleftarrow (\alpha_i^{r_i}\beta_i^{xr_i},\ \beta_i^{r_i})$;
od
$\pi : [1 \ldots n] \to [1 \ldots n] \longleftarrow$ random permutation;
$(C'_1, \ldots, C'_n) \longleftarrow \pi(\bar{C}_1, \ldots, \bar{C}_n)$;
$(C''_1, \ldots, C''_n) \longleftarrow A(C'_1, \ldots, C'_n)$; // "final list"
for $i := 1$ **to** h **do**
  $z_i \longleftarrow \left\{\ j \mid \alpha''_j = (\beta''_j)^{u_i + x}\ \right\}$; // $C''_j = (\alpha''_j, \beta''_j)$
  if $|z_i| \neq 1$ **then** output '$\perp$' **fi**
od
$(p, q) \longleftarrow A'(x, C'_1, \ldots, C'_n)$; // "the guess"
if $\alpha''_p = (\beta''_p)^{u_q + x}$ **and** $\beta''_p \neq 1$ **then** output '1' **else** output '0' **fi**

Definition 3. *The secret permutation generator SPG with n players, $h \geq 2$ of which are honest, has* privacy *if for any adversary $\mathcal{A}$ with resources polynomial in κ, the probability $pr[Exp_{\mathcal{A}}^{priv}(SPG, \kappa, n, h) = \text{'1'}] - 1/h$ is negligible in κ.*

A semi-honest adversary has additional restrictions not inherent in the above experiment that will be discussed in the proof of the following theorem. Also, notice that with a semi-honest adversary, the experiment will never terminate with an output of '$\perp$', and therefore, the tests that lead to this output could be removed if one is only interested in privacy against a semi-honest adversary.

Theorem 3. *Assume ElGamal has GOKI. If we consider only semi-honest adversaries $\mathcal{A}$ then our secret permutation generation protocol has privacy.*

Proof: By contradiction, we suppose there exists a semi-honest adversary $\mathcal{A}$ for which $pr[Exp_{\mathcal{A}}^{priv}(SPG, \kappa, n, h) = \text{'1'}] - 1/h$ is *non-negligible* in κ and use this algorithm to construct an adversary $\mathcal{A}'$ that can break GOKI.

Throughout this proof we will assume the h keys $g^{u_1}, \ldots, g^{u_h}$ provided by Experiment 2 are distinct. The probability that they are not is negligible in κ.

There exists a vector $(v_1, \ldots, v_n) \in \mathbb{Z}_q^n$ for which each $C_i \in \mathcal{E}(v_i)$ and $\bar{C}_i(x) \in \mathcal{E}(v_i + x)$. We denote $\bar{C}_i$ as $\bar{C}_i(x)$ to emphasis the dependence on x.

Since the adversary is semi-honest, it must be the case that there exists $\delta \in \mathbb{Z}_q$ such that for each j there exists a unique i for which $C_i \in \mathcal{E}(u_j - \delta)$ (i.e. $v_i = u_j - \delta$). Furthermore, the algorithm A must perform a key addition with offset δ on each input ciphertext, a re-encryption of each ciphertext, and some permutation (shuffling) of the ciphertexts. We note that the shuffling of the ciphertexts cannot help the adversary in any way and so, without loss of generality, we assume algorithm A does not shuffle its output.

We saw in Section 2.1 that an algorithm which performs key addition with offset δ can be transformed into an algorithm that performs key addition with offset $-\delta$ by simply negating the key of both the input ciphertext and the output ciphertext. In this way, we construct algorithm A^- from A which performs key addition with offset $-\delta$ (and re-encryption) on each of the input ciphertexts. Note that the composition of A and A^- in either order is an algorithm which, on input a list of n ciphertexts, simply re-encrypts each ciphertext and outputs them.

Now we describe the construction of the adversary $\mathcal{A}'$. In Experiment 2, the adversary is invoked with the line

$$(p, q) \longleftarrow \mathcal{A}'(g^{u_1}, \ldots, g^{u_h}, \hat{C}_1, \ldots, \hat{C}_h, \mathcal{O}_{u_1,\ldots,u_h}, \text{“guess”}) \ .$$

The first thing to do is simply pass the keys to the adversary

$$(C_1, \ldots, C_n;\ A, A') \longleftarrow \mathcal{A}(g^{u_1}, \ldots, g^{u_h}) \ ,$$

and then follow Experiment 3, choosing a x and π, until we have completed the computation of the “final list”

$$(C_1'', \ldots, C_n'') \longleftarrow A(C_1', \ldots, C_n') \ .$$

By performing key addition on each of these ciphertexts with offset $-x$ and passing the result to the oracle, the set $H = \{\ i \mid C_i'' \in \mathcal{E}(u_j + x) \text{ for some } j\ \}$ can be computed. Since the adversary is semi-honest, this set must have exactly h elements.

Now, construct a new final list $(\hat{C}_1'', \ldots, \hat{C}_n'')$ as follows. For each i, if $i \notin H$ then set $\hat{C}_i'' = C_i''$, otherwise $\hat{C}_i''$ is $\hat{C}_j$ with the offset x added to the key. Each $\hat{C}_j$ must be used exactly once so we choose a bijection $\mu : [1 \ldots h] \to H$ and set $j = \mu^{-1}(i)$.

With this new final list, apply A^- to compute a new intermediate list

$$(\hat{C}_1', \ldots, \hat{C}_n') \longleftarrow A^-(\hat{C}_1'', \ldots, \hat{C}_n'') \ .$$

Finally, the new intermediate list is simply passed to algorithm A' to obtain a mapping

$$(p, q) \longleftarrow A'(x, \hat{C}_1', \ldots, \hat{C}_n') \ .$$

If this mapping is an obviously incorrect guess, i.e. $p \notin H$, then $\mathcal{A}'$ is to output a random guess and terminate. Otherwise, the pair $(\mu^{-1}(p), q)$ is output to indicate that $\hat{C}_{\mu^{-1}(p)}$ is an element of $\mathcal{E}(u_q)$.

Guesses made by $\mathcal{A}$ fall into three categories: correct, wrong and obviously wrong ($p \notin H$). Since an obviously wrong guess is turned into a valid guess for $\mathcal{A}'$ we have that the probability $\mathcal{A}'$ outputs a correct guess is $P_{correct} + P_{bad}(1/h)$, where P_{bad} is the probability that a guess is obviously wrong. Since $P_{correct} \geq 1/h + k^{-\epsilon}$ for some $\epsilon > 0$ and infinitely many κ (the security parameter), we conclude that $\mathcal{A}'$ is correct with at least this same probability, thus breaking GOKI. ■

4.2.1 Privacy against a malicious adversary

As mentioned, our protocol does not provide privacy against a malicious adversary without a few modifications. The most significant change here is to make use of a *non-mailable commitment scheme* to ensure that no player can compute his x_i value as a function of the other player's values. The commitment scheme is a method that allows a player to commit to their value while keeping it hidden. Later, the player can reveal his value and all other players can verify that it is the one committed to. The non-mailable aspect of such a commitment is discussed below. The modifications required to the steps of the protocol are as follows:

Step 1: Using a non-mailable commitment scheme $\mathcal{C}$, each player broadcasts a commitment $\mathcal{C}(x_i)$ to x_i.

Step 3: Each player must verify that all of the ciphertexts in the list they receive are properly formed. This is done by checking that for all j, $\alpha_j \in G$, $\beta_j \in G$ and $\beta_j \neq 1$. If any of these checks fail, the protocol is terminated.

Step 5: In broadcasting x_i to all players, the commitment $\mathcal{C}(x_i)$ is opened. Also, as described in the above section on correctness, if the use of a hash function is required the hash value is broadcast in this step.

Step 6: In addition to the other checks described, the validity of the commitments is checked, and if a hash function was used, the hash values are also checked. If any of these checks fail, the protocol terminates with failure.

One must carefully choose the commitment scheme to use. If one player, say Player i, commits to x_i with $\mathcal{C}(x_i)$, some other player could commit to the same value, but since that player does not know x_i, that player would be unable to correctly perform Step 3. The result would be failure of the protocol. However, if a player, given the commitment $\mathcal{C}(x_i)$, can generate either $\mathcal{C}(-x_i)$ or $\mathcal{C}(x_j - x_i)$, for some x_j, the privacy of the protocol can be broken. For this reason using g^{x_i} as a commitment to x_i will not work. We believe committing to each bit in the binary representation of x_i should suffice. Other, more efficient, non-mailable commitment schemes may work as well. From this point on we assume a suitable commitment scheme has been selected.

With these modifications to the protocol in mind, we now prove our SPG has privacy against an almost malicious player. The adversary is malicious but has one restriction placed upon it; the algorithm $A(\bar{C}_1, \ldots, \bar{C}_n)$ is implemented as n algorithms $A_1(\bar{C}_1), \ldots, A_n(\bar{C}_n)$. We are working to remove this restriction.

Before stating our theorem, we prove two important lemmas. The first allows us to prove that for each public key u_j, at least one ciphertext in the "first list" is a function of that key and that key alone. The second is needed to prove that each of the h ciphertexts in the "first list" that are functions of the public keys have a specific form.

The following experiment tests an adversary's ability to produce two final ciphertexts from one intermediate ciphertext. We refer to this experiment as the *dual* experiment to emphasize the generation of two ciphertexts from one.

Experiment 4. $Exp_{\mathcal{A}}^{dual}(\kappa)$

$u_1, u_2 \in_R \mathbb{Z}_q$;
$(C_0, A) \longleftarrow \mathcal{A}(g^{u_1}, g^{u_2})$; // "initial ciphertext"
if $\alpha_0 \notin G$ **or** $\beta_0 \notin G$ **then** output '0' **fi** // $C_0 = (\alpha_0, \beta_0)$
$x_1, x_2 \in_R \mathbb{Z}_q$;
$r_1, r_2 \in_R \mathbb{Z}_q$;
$C_1' \longleftarrow (\alpha_0^{r_1} \beta_0^{x_1 r_1}, \beta_0^{r_1})$;

$C'_2 \longleftarrow (\alpha_0^{r_2}\beta_0^{x_2 r_2}, \beta_0^{r_2})$;
$(C''_1, C''_2) \longleftarrow A(C'_1, C'_2)$; // "final ciphertext"
if $\beta''_1 = 1$ **or** $\beta''_2 = 1$ **then** output '0' **fi** // $C'' = (\alpha'', \beta'')$
if $\alpha''_1 = (\beta''_1)^{u_1+x_1}$ **and** $\alpha''_2 = (\beta''_2)^{u_2+x_2}$ **then** output '1' **else** output '0' **fi**

Lemma 1. *Assuming KI (as in Theorem 2), the probability* $pr[Exp_{\mathcal{A}}^{dual}(\kappa) = \text{'1'}]$ *is negligible in* κ.

Proof: By contradiction, suppose $\mathcal{A}$ is an adversary that is successful in the experiment with non-negligible probability. We show how this adversary can be used to break Theorem 2.

We are given as input a key g^x and a ciphertext $C \in \mathcal{E}(u)$, and we must show how to compute a ciphertext $C' = (\alpha', \beta')$ for which $\beta' \neq 1$ and $C' \in \mathcal{E}(u + x)$.

Choose $u_1 \in_R \mathbb{Z}_q$, compute g^{u_1}, and set $g^{u_2} = g^x$. These two public keys are given to $\mathcal{A}$ to get a ciphertext C_0. There exists some v_0 for which this ciphertext is an encryption under the key g^{v_0}.

During a "normal" invocation of A, the ciphertext C'_1, an encryption under the key $g^{v_0+x_1}$, is transformed into C''_1, an encryption under the key $g^{u_1+x_1}$. This is key addition with offset $u_1 - v_0$. Likewise, the ciphertext C'_2 is transformed into C''_2 via key addition with offset $u_2 - v_0$.

In Section 2 we noted that an algorithm that performs key addition with offset δ can be easily transformed into an algorithm to perform key addition with offset $-\delta$. Simply negate the key of both the input ciphertext and the output ciphertext. Let A^- be A transformed as described.

To break the lemma, we need two invocations of A. For the first, we provide C as input to A^-, first parameter, to obtain an element of $\mathcal{E}(u - u_1 + v_0)$. Using key addition, this ciphertext is transformed into an element of $\mathcal{E}(u + v_0)$. Finally, this latter ciphertext is given to A as the second parameter to obtain an element of $\mathcal{E}(u + u_2) = \mathcal{E}(u + x)$.

If each invocation of A yields an appropriate result with probability greater than $k^{-\epsilon}$, then our final probability of success at least $k^{-2\epsilon}$, which is non-negligible in κ. ■

We further note that this lemma holds even if the probability is only computed over experiments for which $x_1 = x_2$ as the proof does not require that these two values be independent.

Lemma 2. *Suppose* h *coloured balls are tossed (randomly) into* n *coloured bins with at most one ball allowed in each bin. The colours are chosen from some set and duplicates are allowed. Unless all of the balls are the same colour, the probability that each ball matches the colour of the bin it lands in is at most* $1/h$.

Proof: Suppose d of the balls are, say, red and $h - d$ are black. Also, to ensure the probability of a match is non-zero, we assume that at least d bins are red and at least $h - d$ are black. Some bins (up to $n - h$) may be painted some third colour. We consider a toss of the balls to have been a *success* if the colour of each ball matches the colour of the bin it landed in.

Let the bins be lined up in some order, toss the balls, and consider the h bins that have a ball in them. If any of these bins are a third colour, the toss was a failure. Assume this is not the case. Map the colours of the h bins to a string of bits with red mapping to 1 and black mapping to 0. If this bit string (of length h) is not of weight d, the toss was a failure, so we assume this is not the case. Label this bit string X.

Now consider the h balls and map their colours to the bits of a bit string labelled Y. The toss is only a success if the two bit strings match exactly (i.e. $X = Y$).

The bit string Y has weight d and was selected uniformly from the set of all weight d strings. This set is of size $\binom{h}{d}$. Therefore, the probability that $X = Y$ is $1/\binom{h}{d}$, and since we made some

assumptions about X above, this probability is an upper bound. Notice that for $0 < d < h$, $1/\binom{h}{d} \leq 1/h$. Only if $d = 0$ or $d = h$ is a probability greater than $1/h$ possible.

Finally, we note that if there are more than two distinct ball colours, the probability of success is further reduced, thus proving the lemma. ■

Theorem 4. *Assume ElGamal has GOKI. If we consider only malicious adversaries $\mathcal{A}$ for which the adversary provided algorithm $A(\bar{C}_1, \ldots, \bar{C}_n)$ is implemented as n algorithms $A_1(\bar{C}_1), \ldots, A_n(\bar{C}_n)$, then our (modified) secret permutation generation protocol has privacy.*

Proof: By contradiction, we suppose there exists an adversary $\mathcal{A}$ for which $pr[Exp^{priv}_{\mathcal{A}}(SPG, \kappa, n, h) = \text{'1'}] - 1/h$ is *non-negligible* in κ and use this algorithm to construct an adversary $\mathcal{A}'$ that can break GOKI.

Throughout this proof we will assume the h keys $g^{u_1}, \ldots, g^{u_h}$ provided by Experiment 2 are distinct. The probability that they are not is negligible in κ.

There exists a vector $(v_1, \ldots, v_n) \in \mathbb{Z}_q^n$ for which each $C_i \in \mathcal{E}(v_i)$ and $\bar{C}_i(x) \in \mathcal{E}(v_i + x)$. We denote $\bar{C}_i$ as $\bar{C}_i(x)$ to emphasis the dependence on x.

Claim: If, for some i, j, k, and non-negligibly many x, algorithm A_k, on input $\bar{C}_i(x)$, outputs an element of $\mathcal{E}(u_j + x)$, then for all $j' \neq j$ and k', the number of x values for which algorithm $A_{k'}$, also on input $\bar{C}_i(x)$, outputs an element of $\mathcal{E}(u_{j'} + x)$, is negligible.

This follows from Lemma 1 and establishes that each ciphertext C_i can be used to generate elements of $\mathcal{E}(u_j + x)$ for at most one j. Therefore, each of at least h of the n ciphertexts output by $\mathcal{A}$ as the "first list" must have been constructed for the purpose of generating elements from a particular set $\mathcal{E}(u_j + x)$.

Each algorithm A_k is probabilistic and therefore its action is difficult to predict. Despite this, we will consider each of these algorithms, on any particular run, to be performing key addition with some offset; however, the offset may vary from one run to the next. Our justification for this is that the algorithm cannot (with non-negligible probability) output an element of one of the sets $\mathcal{E}(u_j + x)$ without performing key addition (even if it is with an offset of 0). Indeed, on any particular run, the probability that at least h of the algorithms are performing key addition must be at least $1/h$.

Claim: There exists $\delta \in \mathbb{Z}_q$ such that for all j there exists an i for which $C_i \in \mathcal{E}(u_j - \delta)$ (i.e. $v_i = u_j - \delta$).

Suppose, during some run of the algorithms, each algorithm A_k performs key addition with the offset δ'_k. Furthermore, each $C_i \in \mathcal{E}(u_j - \delta_i)$ for some j and δ_i. Actually, with the latter, for all j there exists some δ_i for which $C_i \in \mathcal{E}(u_j - \delta_i)$, but we assume the adversary had some j in mind when he generated C_i. If algorithm A_k is to process the input $\bar{C}_i(x)$ and output an element of $\mathcal{E}(u_j + x)$, then it is necessary that $\delta'_k = \delta_i$. Because of this we consider each ciphertext C_i to be coloured by δ_i and each algorithm to be coloured by δ'_k. A valid final list can only be produced if the permutation π maps h of the ciphertexts to similarly coloured algorithms.

Lemma 2 establishes that unless at least h of the ciphertexts are of the same colour, the probability that a valid final list is generated is at most $1/h$. Let δ be this colour. Furthermore, because of the specific requirements for a valid final list, it must be the case that for every j there exists an i for which $C_i \in \mathcal{E}(u_j - \delta)$.

Now we describe the construction of the adversary $\mathcal{A}'$. In Experiment 2, the adversary is invoked with the line

$$(p, q) \longleftarrow \mathcal{A}'(g^{u_1}, \ldots, g^{u_h}, \hat{C}_1, \ldots, \hat{C}_h, \mathcal{O}_{u_1,\ldots,u_h}, \text{"guess"}) \ .$$

The first thing to do is simply pass the keys to the adversary

$$(C_1, \ldots, C_n;\ A_1, \ldots, A_n, A') \longleftarrow \mathcal{A}(g^{u_1}, \ldots, g^{u_h}) \ ,$$

and then follow Experiment 3, choosing a x and π, until we have completed the computation of the "final list"

$$(C''_1, \ldots, C''_n) \longleftarrow (A_1(C'_1), \ldots, A_n(C'_n)) \ .$$

By performing key addition on each of these ciphertexts with offset $-x$ and passing the result to the oracle, the set $H = \{\ i \mid C''_i \in \mathcal{E}(u_j + x) \text{ for some } j\ \}$ can be computed. If this set does not have exactly h elements, then $\mathcal{A}'$ is to output a random guess and terminate. From the claims proven earlier, we know that with probability at least $1/h$, H has exactly h elements, for each j there exists an $i \in H$ such that $C''_i \in \mathcal{E}(u_j + x)$, and there exists a δ such that for $k \in H$, A_k, during this run, performed key addition with an offset of δ.

Now, let algorithm A_k^- be A_k but with both the input ciphertext and output ciphertext modified by key negation. Recall that the result of this is that if A_k performs key addition with offset δ, then A_k^- will perform key addition with offset $-\delta$.

To finish the construction of the adversary, we now compute a new intermediate ciphertext list $(\hat{C}'_1, \ldots, \hat{C}'_n)$. For each i, if $i \notin H$ then set $\hat{C}'_i = C'_i$, otherwise $\hat{C}'_i$ is $\hat{C}_j$ with the offset x added to the key and processed with A_i^-. Each $\hat{C}_j$ must be used exactly once so we choose a bijection $\mu : [1 \ldots h] \to H$ and set $j = \mu^{-1}(i)$. As above, with probability at least $1/h$, the h algorithms used here all performed key addition with offset δ.

Finally, the new intermediate list is simply passed to algorithm A' to obtain a mapping

$$(p, q) \longleftarrow A'(x, \hat{C}'_1, \ldots, \hat{C}'_n) \ .$$

If this mapping is an obviously incorrect guess, i.e. $p \notin H$, then $\mathcal{A}'$ is to output a random guess and terminate. Otherwise, the pair $(\mu^{-1}(p), q)$ is output to indicate that $\hat{C}_{\mu^{-1}(p)}$ is an element of $\mathcal{E}(u_q)$.

The probability that $\mathcal{A}'$ makes it to this last step and the list $(\hat{C}'_1, \ldots, \hat{C}'_n)$ is a well formed ciphertext list (i.e. there exists some π that would yield this intermediate list in Experiment 3) is at least $1/h^2$. If P_{guess} is the probability that $\mathcal{A}$ correctly guesses a mapping given that the final list is valid, then the probability that $\mathcal{A}'$ outputs a correct mapping is at least

$$\left(1 - \frac{1}{h^2}\right)\frac{1}{h} + \frac{1}{h^2}P_{guess} \ .$$

If $P_{guess} \geq 1/h + k^{-\epsilon}$ for some $\epsilon > 0$ and infinitely many κ (the security parameter), then the probability that $\mathcal{A}'$ is correct is at least $1/h + k^{-\epsilon}/h^2$ which is non-negligibly greater than $1/h$, thus breaking GOKI. ■

It is our intention to show, in future work, that the restricted form of the algorithm A does not in fact weaken the adversary. Not only does re-arrangement of the ciphertexts, using an intermediate ciphertext more than once, and duplicating output ciphertexts not provide the adversary with any advantage, we also note the infeasibility of creating an algorithm that takes as input two ciphertexts and combines them to create a single meaningful ciphertext as output. With this in mind, we state the following conjecture.

Conjecture 2. *Assuming ElGamal has GOKI, our (modified) SPG protocol has privacy against a malicious adversary.*

4.3 Uniformity

We stated earlier that uniformity is the property that the positions of the honest players in the final permutation are distributed uniformly. If there are h honest players and n positions, we have $\binom{n}{h}$ distinct arrangements of honest players. It is from this set the positions are to be selected uniformly.

To test an adversary's ability to break uniformity, we consider the following experiment. Let κ be the security parameter, n the number of players, h of which are honest, and $H \subseteq \{1, \ldots, n\}$, $|H| = h$, the set of positions the adversary is aiming to put the honest players into. The bulk of the following experiment is identical to Experiment 3 with the only difference being after the final list has been tested for validity. If the final list is valid the experiment simply checks to see if the honest players are in the positions specified by the set H. If so, the experiment terminates with output '1'. If not, the output is '0'.

Experiment 5. $Exp_{\mathcal{A}}^{fair}(SPG, \kappa, n, h, H)$

for $i := 1$ **to** h **do**
  $u_i \in_R \mathbb{Z}_q$;
od
$(C_1, \ldots, C_n;\ A) \longleftarrow \mathcal{A}(g^{u_1}, \ldots, g^{u_h})$; // "first list"
$x \in_R \mathbb{Z}_q$;
for $i := 1$ **to** n **do**
  if $\alpha_i \notin G$ **or** $\beta_i \notin G$ **or** $\beta_i = 1$ **then** output '$\perp$' **fi** // $C_i = (\alpha_i, \beta_i)$
  $r_i \in_R \mathbb{Z}_q$;
  $\bar{C}_i \longleftarrow (\alpha_i^{r_i} \beta_i^{x r_i},\ \beta_i^{r_i})$;
od
$\pi : [1 \ldots n] \to [1 \ldots n] \longleftarrow$ random permutation;
$(C'_1, \ldots, C'_n) \longleftarrow \pi(\bar{C}_1, \ldots, \bar{C}_n)$;
$(C''_1, \ldots, C''_n) \longleftarrow A(C'_1, \ldots, C'_n)$; // "final list"
for $i := 1$ **to** h **do**
  $z_i \longleftarrow \left\{ j \mid \alpha''_j = (\beta''_j)^{u_i + x} \right\}$; // $C''_j = (\alpha''_j, \beta''_j)$
  if $|z_i| \neq 1$ **then** output '$\perp$' **fi**
od
$H' \longleftarrow \left\{ j \mid \alpha''_j = (\beta''_j)^{u_i + x} \text{ for some } i \right\}$;
if $H = H'$ **then** output '1' **else** output '0' **fi**

Definition 4. *The secret permutation generator SPG with n players, h of which are honest, has uniformity if for any adversary $\mathcal{A}$ with resources polynomial in κ and any $H \subseteq \{1, \ldots, n\}$, $|H| = h$, the probability $pr[Exp_{\mathcal{A}}^{fair}(SPG, \kappa, n, h, H) = \text{'1'}] - 1/\binom{n}{h}$ is negligible in κ.*

While our SPG does provide uniformity against a semi-honest adversary, even the modified version does not provide uniformity against a malicious one. There is a malicious adversary that can break uniformity, but only with a high probability that the experiment fails with an invalid "final list". Such an adversary operates as follows. Each ciphertext C_i is an encryption under one of the public keys u_j, where each public key is used roughly the same number of times. Then, algorithm A is constructed to randomly choose exactly h of the intermediate ciphertexts and combine them with $n - h$ randomly generated ciphertexts. The chosen intermediate ciphertexts take the positions described by H in the final list. If the final list is valid, the honest player ciphertexts are in exactly the right positions.

We calculate that the probability the final list is valid is approximately $(n/h)^h/\binom{n}{h}$ (if n is a multiple of h, this expression is exact). Clearly, for all $h < n$, this probability is strictly greater than $1/\binom{n}{h}$. We conjecture, however, that in an environment where the adversary has a strong incentive to not cause the protocol to fail with an invalid final list, our SPG has uniformity.

Conjecture 3. *Assume ElGamal has GOKI. If we consider only malicious adversaries $\mathcal{A}$ for which the probability $pr[Exp_{\mathcal{A}}^{fair}(SPG, \kappa, n, h, H) = '\perp']$ is negligible in κ, then our (modified) secret permutation generation protocol has uniformity.*

Furthermore, we suspect this restriction is too strong and we are exploring the possibility that a higher bound on the probability that the final list is invalid may be allowed.

Note that for one of our stated uses of an SPG, the ordering of players in an anonymous message delivery system, uniformity is not necessary as maximum anonymity is only achieved when the permutation is uniformly selected, and therefore, the players have a strong motivation to ensure such uniformity.

As for its use in online games, there is a straightforward way to add uniformity to any SPG protocol.

4.3.1 Adding Uniformity

The uniformity property is not as essential to a SPG protocol as privacy because uniformity can be easily added with one additional round of communication and a constant factor increase in communication complexity. The additional steps simply require that, after the secret permutation has been generated, the players participate in a protocol to generate a non-secret uniform permutation and then compose the two permutations to yield a secret and uniform final permutation.

One example of a non-secret uniform permutation generation protocol is as follows. Each player chooses an integer from $\mathbb{Z}_{n!}$. The players then commit to and reveal their integers. The commitment is to ensure the integers are independent. Finally, the players sum the integers modulo $n!$ and map the result to a permutation on n items.

5 Application to Games

A secret permutation sharing scheme can be used in a variety of games to either select a secret ordering of the players or to shuffle a deck of cards. Some games (e.g. the board game Civilization) require the players to draw numbers from a hat to determine an initial secret ordering. Many other games require a deck of cards to be shuffled and dealt. For games where an entire deck is to be dealt out (e.g. Bridge or Hearts), the application of our secret permutation sharing scheme is straightforward. Suppose 4 players are to be dealt 13 cards each. In Step 2, Player 1 forms an initial set of 52 ciphertexts from the 4 public keys received

$$[(g^{u_1}, g), \ldots, (g^{u_1}, g),\ (g^{u_2}, g), \ldots, (g^{u_2}, g),\ (g^{u_3}, g), \ldots, (g^{u_3}, g),\ (g^{u_4}, g), \ldots, (g^{u_4}, g)].$$

This list contains 13 copies of each of the 4 ciphertexts. Then, in Step 6, each player can expect to find exactly 13 ciphertexts for which $\alpha/\beta^{w_i} = 1$. The 52 list positions are associated with the cards of a standard deck and each player takes the cards associated with the 13 such ciphertexts as their hand. We are also working on an extension of our protocol to allow the cards to be revealed to players slowly over time.

6 Conclusions

A secure multiparty protocol for computing a secret permutation has been formulated and analyzed. Notions of privacy and uniformity were introduced and the protocol evaluated in terms of these notions. In particular, it was shown the protocol achieves privacy under a weak condition on the behaviour of the adversary. Work to remove this condition continues.

References

[1] Ben Adida and Douglas Wikström, *How to shuffle in public*, Proceedings of the Theory of Cryptography 2007, February 2007.

[2] Gagan Aggarwal, Nina Mishra, and Benny Pinkas, *Secure computation of the k^{th}-ranked element.*, EUROCRYPT (Christian Cachin and Jan Camenisch, eds.), Springer, 2004, LNCS 3027, pp. 40–55.

[3] David Chaum, *The dining cryptographers problem: Unconditional sender and recipient untraceability*, Journal of Cryptology **1** (1988), 65–75.

[4] Yvo G. Desmedt and Yair Frankel, *Threshold cryptosystems*, CRYPTO '89: Proceedings on Advances in cryptology (New York, NY, USA), Springer-Verlag New York, Inc., 1989, LNCS 435, pp. 307–315.

[5] Taher El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, Proceedings of CRYPTO 84 on Advances in cryptology (New York, NY, USA), Springer-Verlag New York, Inc., 1985, LNCS 196, pp. 10–18.

[6] Shafi Goldwasser and Silvio Micali, *Probabilistic encryption.*, J. Comput. Syst. Sci. **28** (1984), no. 2, 270–299.

[7] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson, *Universal re-encryption for mixnets*, RSA Conference Cryptographer's Track, 2004, LNCS 2964, pp. 163–178.

[8] Philippe Golle and Ari Juels, *Dining cryptographers revisited*, Proceedings of Eurocrypt 2004, May 2004, LNCS 3027, pp. 456–473.

[9] Latanya Sweeney and Michael Shamos, *Multiparty computation for randomly ordering players and making random selections*, Tech. Report CMU-ISRI-04-126, Carnegie Mellon University, School of Computer Science, Pittsburgh, July 2004.