Shuffling in secret (and proving it too)

Chris Patton

March 20, 2016

1 Introduction

A mixnet (sometimes called a "mix-network") is a service designed to facilitate anonymous communications over the internet [1]. Each of a set of users $\mathcal{Q} = \{P_1, \ldots, P_n\}$ wish to send a message to a user in $\mathcal{Q}$. Each encrypts their message M and the identity of the intended recipient P_i under the public key of the mixnet server (heereafter referred to as the "mix") and transmits the ciphertext. The mix server waits until it receives $m \leq n$ ciphertexts from m distinct users. It then decrypts each message and destination pair under its secret key and transmits each message to its intended destination. Crucially, the mix transmits the messages in a random order so as to hide the correspondence between senders and recipients. As long as the mix is trusted and the user's ciphertexts are faithfully transmitted to the mix, this simple approach suffices to preserve anonymity of its participants.

Normally we think of the mixnet being composed of a cascade of mix servers, each decrypting, shuffling, then transmitting to the next mix in the cascade until finally the last server outputs the users' messages. This is meant to distribute trust so that as long as the messages pass through at least one honest mix, anonymity is achieved. Still, the trust model here is rather strong. For example, the mix could send a fake message on behalf of one of the senders, or redirect it to a different recipient. We consider the problem of publically verifying that the outputs are indeed a permutation of the inputs. Moreover, we require that the correspondence not be revealed. Andrew Neff describes in [6] a method for proving a shuffle in zero-knowledge.

Neff's solution is based on the ElGamal public key encryption scheme, which is known to be semantically secure assuming the discrete log problem (decisional Diffie-Hellman, or DDH) is hard [2]. Based on this assumption, Neff proves his protocol achieves a property known as honest verifier zero-knowledge (HVZK) [5]. Informally speaking, this means that as long as the verifier faithfully executes the protocol, the mix can prove the outputs are a permutation of the inputs without revealing the permutation.

We formulate and define security for the problem and present Neff's protocol. For simplicity, we limit the discussion to a mixnet with a single mix server. In this scenario, users encrypt their messages under the public key of the mix. The mix waits until it receives a full batch of ciphertexts, decrypts them under its private key, and shuffles the output. In section 2, we specify ElGamal encryption, describe the DDH assumption, and

sketch the HVZK notion. In section 3, we present the protocol and sketch its security. In section 4 we discuss the details of the implementation.

2 Preliminaries

If X is a finite set, let $x \twoheadleftarrow X$ denote uniformly sampling an element x from the set. Let $y \leftarrow A(x)$ denote running A on input x and assigning its output to y. Let $y \leftarrow A(x;r)$ denote the execution of a probabilistic algorithm with the sequence of coins $r \in \{0,1\}^\infty$. Let $y \twoheadleftarrow A(x)$ denote choosing $r \twoheadleftarrow \{0,1\}^\infty$ and executing $y \leftarrow A(x;r)$. Let $[i..j]$ where $i \leq j$ denote the set of integers from i to j inclusively. Let $[n] = [1..n]$. Let $\mathsf{Perm}(n)$ denote the set of permutations on vectors of length n. Let $y \twoheadleftarrow \langle P, V\rangle(x)$ denote the probabilistic execution of interactive Turing machines P and V on common input x. When the protocol finishes, the output of V is assigned to y. x is called the common input of P and V. Finally a function $\Delta(n)$ is negligible if for all positive polynomials $p(\cdot)$ and sufficiently large n, $\Delta(n) < \frac{1}{p(n)}$.

2.1 ElGamal encryption

Fix a prime number p and let $\mathbb{Z}_p$ denote the modular ring of order p. In what follows, we assume arithmetic is performed in the modular ring $\mathbb{Z}_p$. Let $\mathbb{Z}_p^*$ denote the corresponding multiplicative group. Let $g \in \mathbb{Z}_p$ such that $g^q = 1$ where q is prime. Then $\langle g\rangle = \mathbb{Z}_q^*$ is a subgroup of $\mathbb{Z}_p^*$ and $q|(p-1)$. We call (p,q,g) the *public parameters*. The *secret key* is chosen uniformly from the set of powers of g mod p: let $k \twoheadleftarrow [q-1]$. Let $K = g^k$ be the *public key*. Key generation is denoted $(K,k) \twoheadleftarrow \mathcal{G}(p,q,g)$. Encryption and decryption operate on group elements.[1] Encryption of $M \in \mathbb{Z}_p^*$ proceeds as follows: Let $r \twoheadleftarrow [q-1]$, $R = g^r$, $S = K^r$, and $C = M \cdot S$. Output the tuple (C,R). This is denoted $(C,R) \twoheadleftarrow \mathcal{E}_K(M)$. Decryption of a ciphertext $(C,R) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ proceeds as follows: let $S = R^k$ (S is called the *shared secret*) and $M = C \cdot S^{-1}$. Normally the decryption algorithm would output just M, but for the purpose of verification, we demand that it also returns the shared secret. Thus, decryption is denoted $(M,S) \leftarrow \mathcal{D}_k(C,R)$.

ElGamal is known to be *semantically secure* under the decisional Diffie-Hellman assumption [2, 4]. Informally speaking, this means that if given K, it is infeasible for any adversary to determine k, then given (C,R), it is infeasible for any adversary to learn anything meaningful about M. As we're not interested in the security of ElGamal from a privacy standpoint, we won't formalize semantic security. However, we define the DDH assumption in the next subsection.

The mixnet operates as follows: let $M_1, \ldots, M_n \in \mathbb{Z}_p^*$ be a sequence of plaintexts. Let $(C_1,R_1), \ldots, (C_n,R_n) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ be a sequence of corresponding ElGamal ciphertexts encrypted under the public-key K. The mix server, who possesses the secret-key k, takes as input the ciphertexts and outputs a sequence $(M_{\pi(1)}, S_{\pi(1)}), \ldots, (M_{\pi(n)}, S_{\pi(n)})$. where π is a permutation on $[n]$. We remark that there is a natural extension of this scheme

[1] Public-key encryption schemes are typically defined on bit strings. It is easy to messages as elements of $\mathbb{Z}_p^*$, but the number of messages that can be encoded depends on the modulus p.

for mixnets with a cascade of servers, each possessing its own decryption key [7]. Indeed, Camenisch and Mityagin use the Neff proof in order to verify the output of each mix in the cascade. If a dishonest mix is detected, the protocol is run without without using it.

2.2 Decisional Diffie-Hellman

Let (p, q, g) be the public parameters of the ElGamal cryptosystem. The game the adversary D plays is defined as follows: let $a, b, r \twoheadleftarrow [q-1]$ and let $A = g^a$, $B = g^b$, $Y_0 = g^r$, and $Y_1 = g^{ab}$. Choose a random bit $b \twoheadleftarrow \{0,1\}$. On input (A, B, Y_b), the adversary outputs a bit $b' \in \{0,1\}$ and wins if $b = b'$. The advantage of D is defined as

$$\Delta(q) = |\Pr[D(A, B, Y_1) = 1] - \Pr[D(A, B, Y_0) = 1]|$$

The DDH assumption is that, for every polynomial-time adversary D, the function $\Delta(q)$ is negligible.

2.3 Honest verifier zero-knowledge

Zero-knowledge is defined in [5] with respect to the class of formal languages with interactive proof systems. A language L has an interactive proof system if there exists a pair of probabilistic algorithms (P, V) such that for all $x \in L$, $\Pr[\langle P, V\rangle(x) = 1] \geq 2/3$ (this property is called *completeness*), and for all $x \notin L$, $\Pr[\langle P, V\rangle(x) = 1] \leq 1/3$ (this property is called *soundenss*). The class of languages with interactive proof systems is denoted **IP**.

Let $L \in \mathbf{IP}$ exhibited by (P, V). We define security with respect to a game associated to adversary D, verifier V^*, and simulator M^*. Suppose that L is a finite set[2] and let $x \twoheadleftarrow L$. Let $y_1 \twoheadleftarrow \langle P, V^*\rangle(x)$ and $y_0 \twoheadleftarrow M^*(x)$ choose a bit $b \twoheadleftarrow \{0,1\}$. On input (x, y_b), adversary D outputs a bit b' and wins if $b = b'$. Let $k = |L|$ and define the advantage of D as

$$\Delta(k) = |\Pr[D(x, y_1) = 1] - \Pr[D(x, y_0) = 1]|$$

We say that (P, V) is *computational zero-knowledge* (or just *zero-knowledge*) if for every probabilistic polynomial-time adversary D and for every probabilistic polynomial-time verifier V^*, there exists probabilistic polynomial-time simulator M^* such that $\Delta(k)$ is negligible. The security notion achieved by Neff's protocol is somewhat weaker: a proof system (P, V) is *honest verifier zero-knowledge* (HVZK) if for every probabilistic polynomial-time adversary D, there exists a probabilistic polynomial-time simulator M^* such that

$$|\Pr[D(x, \langle P, V\rangle(x)) = 1] - \Pr[D(x, M^*(x)) = 1]|$$

is a negligible function of k.

[2] This is without loss of generality, since we consider polynomial-time adversaries.

3 The Neff shuffle

We now give the syntax of an interactive proof system for mixnets. The prover P (nominally the mix server) possesses the secret key k and the verifier V possesses the public key K. The common input of P and V is all of the messages passed through the mix server. The mix takes as input a sequence of ciphertexts $(C_1, R_1), \ldots, (C_n, R_n) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and outputs a sequence of plaintexts $(M_1, S_1), \ldots, (M_n, S_n) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$. Let $x \in \{0,1\}^*$ denote the encoding of the input ciphertexts and output plaintexts. We say $x \in L$ if and only if there exists a bijection $\pi : [n] \to [n]$ such that for every $i \in [n]$, $(M_{\pi(i)}, S_{\pi(i)}) = \mathcal{D}_k(C_i, R_i)$.

A trivial proof system for L is one in which the prover transmits to the verifier its secret key and the verifier decrypts each of the ciphertexts. Of course, this completely exposes π. Our goal is to specify a proof system (P, V) in which P convinces V of the existence of π without revealing the permutation or the secret key. A proof system that is HVZK is sufficient to achieve this goal, which we present in section 3.3. This protocol is built up from simpler protocols presented in sections 3.1 and 3.2. Finally, in section 3.4, we show how to apply the main protocol to mixnets.

3.1 The basic protocol and its security

The main building block for the protocol is a zero-knowledge proof for the *iterated logarithmic multiplication problem* (ILMP). Let $n \in \mathbb{N}$, $x_1, \ldots, x_n \in [q-1]$, and $y_1, \ldots y_n \in [q-1]$. Let $X_i = g^{x_i}$ and $Y_i = g^{y_i}$ for each $i \in [n]$. Given the sequences $(X_1, \ldots X_n)$ and $(Y_1, \ldots, Y_n)$, the problem is to decide if $g^{x_1 \cdots x_n} = g^{y_1 \cdots y_n}$. Neff points out this problem is closely related to the Chaum-Pederson signature scheme [3] and the corresponding solution is nothing more than a multidimensional generalization of their proof. The following is an interactive proof for ILMP.

$\underline{\textsc{Silmpp}_{\langle P,V\rangle}\big((X_1, \cdots, X_n), (Y_1, \ldots, Y_n)\big)}$:

P1. Let $\theta_0 = \theta_1 = 0$ and $\theta_1, \ldots, \theta_{n-1} \twoheadleftarrow [q-1]$. Let $A_i = X_i^{\theta_{i-1}} \cdot Y_i^{\theta_i}$ for every $i \in [n]$. Send the sequence $A_1, \ldots A_n$ to V.

V1. Let $\gamma \twoheadleftarrow [q-1]$. Send γ to P.

P2. Compute $r_1, \ldots, r_{n-1} \in [q-1]$ satisfying the set of equations:

$$\begin{aligned} Y_1^{r_1} &= A_1 \cdot X_1^{-\gamma} \\ X_i^{r_{i-1}} \cdot Y_i^{r_i} &= A_i \qquad (\forall\, i \in [2..n-1]) \\ X_n^{r_{n-1}} &= A_n \cdot Y_n^{(-1)^{(n-1)}\gamma} \end{aligned} \tag{1}$$

Send the sequence $r_1, \ldots r_{n-1}$ to V.

V2. Accept if equation (1) holds; otherwise reject.

Figure 1: SILMP protocol, an interactive proof for ILMP. The prover P knows $x_1, \ldots x_n$ and $y_1, \ldots, y_n$.

COMPLETENESS. Assume that for every $i \in [n]$, it holds that $x_i \neq 0$ and $y_i \neq 0$. (Otherwise, the verifier could see by inspection whether or not $(X_1, \ldots X_n), (Y_1, \ldots, Y_n)$ is in the language by inspection. If $(\exists\, i \in [n])\, X_i = 1$, then $x_i = 0$ and $g^{x_1 \cdots x_n} = 1$. The same is true for Y_i.) Taking the log base-g of both sides of equation (1) yields the following system of linear equations:

$$\begin{bmatrix} y_1 & 0 & 0 & 0 & \cdots & 0 \\ x_2 & y_2 & 0 & 0 & \cdots & 0 \\ 0 & x_3 & y_3 & 0 & \cdots & 0 \\ & & & & \ddots & \\ 0 & 0 & \cdots & 0 & x_{n-1} & y_{n-1} \\ 0 & 0 & \cdots & 0 & 0 & y_n \end{bmatrix} \begin{bmatrix} r_1 - \theta_1 \\ r_2 - \theta_2 \\ r_3 - \theta_3 \\ \vdots \\ r_{n-2} - \theta_{n-2} \\ r_{n-1} - \theta_{n-1} \end{bmatrix} = \begin{bmatrix} -\gamma x_1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ (-1)^{(n-1)}\gamma \end{bmatrix} \tag{2}$$

It can be shown that the $(n \times n)$ matrix in the left-hand side of equation (2) is non-singular. This implies there exists a unique solution $(r_1, \ldots, r_{n-1})$ satisfying this system. Neff gives the solution explicitly:

$$r_i - \theta_i = (-1)^{(n-i-1)}\gamma \prod_{j=i+1}^{n} y_i {x_i}^{-1} \tag{3}$$

(The inverse of x_i is computed in the group $\mathbb{Z}_q^*$. This is found using Euclid's algorithm.) Thus, if the prover isn't cheating, then the verifier accepts with probability 1.

SOUNDNESS. Suppose the prover is cheating and $g^{x_1 \cdots x_n} \neq g^{y_1 \cdots y_n}$. For any $r_1, \ldots, r_{n-1} \in [q-1]$, there is at most one $\gamma \in [q-1]$ such that equation (1) holds. If γ is chosen randomly, then the verifier accepts with probability at most $1/q$.

SECURITY. Let x be an instance of ILMP. The claim is that if the DDH assumption holds, then for every probabilistic polynomial D, there exists a probabilistic, polynomial time simulator M^* such that

$$\Delta(q) = |\Pr[D(x, \langle P, V\rangle(x)) = 1] - \Pr[D(x, M^*(x)) = 1|$$

is negligible. To prove this claim, one would argue the contrapositive: suppose there exists adversary D such that for every simulator M^*, its advantage in distinguishing the distribution $\langle P, V\rangle(x)$ and $M^*(x)$ is non-negligible. Then there exists an adversary A whose advantage in the DDH game outlined in section 2.2 is non-negligible. We omit the details of the proof here, but claim this can be shown via a reduction from A to D.

3.2 Simple n-shuffle

Let $n \in \mathbb{N}$, $x_1, \ldots, x_n \in [q-1]$, and $y_1, \ldots y_n \in [q-1]$. Let $X_i = g^{x_i}$ and $Y_i = g^{y_i}$ for each $i \in [n]$. Let $c, d \in [q-1]$, $C = g^c$, and $D = g^d$. We say that $x \in$ SHUFFLE0 if x is an encoding of $(X_1, \ldots X_n)$, $(Y_1, \ldots Y_n)$, C, and D and there exists a permutation $\pi : [n] \to [n]$ such that for every $i \in [n]$, it holds that $Y_i^d = X_{\pi(i)}^c$. A zero-knowledge proof for SHUFFLE0 is specified in figure 2.

$\underline{\textsc{Shuffle0}_{\langle P,V\rangle}\big((X_1,\cdots,X_n),(Y_1,\ldots,Y_n),C,D\big)}$:

V1. Let $t \twoheadleftarrow [q-1]$, $U = D^t$, and $W = C^t$. Let $\hat{X}_i = X_i \cdot U^{-1}$ and $\hat{Y}_i = Y_i \cdot W^{-1}$ for every $i \in [n]$. Send t to P.

P1. Let $U = g^{dt}$ and $W = g^{ct}$. Let $\hat{X}_i = X_i \cdot U^{-1}$ and $\hat{Y}_i = Y_i \cdot W^{-1}$ for every $i \in [n]$. Run

$$\textsc{Silmpp}_{\langle P,V\rangle}\big((\hat{X}_1,\ldots,\hat{X}_n,C,\ldots,C),(\hat{Y}_1,\ldots,\hat{Y}_n,D,\ldots,D)\big)$$

where the number of C's and D's is n.

V2. Accept if V accepts in step P1.

Figure 2: The simple n-shuffle, an interactive proof system for Shuffle0. The prover knows $x_1,\ldots x_n$ and $y_1,\ldots,y_n$, c, and d. (Here, inverse is computed in $\mathbb{Z}_p^*$.)

3.3 General n-shuffle

The simple n-shuffle is *almost* what we need; its only limitation is that it is required that the prover knows the log of each X_i and Y_i. The general n-shuffle, outlined in this section, overcomes this deficiency. Let $X_1,\ldots X_n, Y_1,\ldots,Y_n \in \mathbb{Z}_p^*$, $k \in [q-1]$, and $K = g^k$. We say $(Y_1,\ldots,Y_n)$ is a shuffling of $(X_1,\ldots X_n)$ if there exists a permutation $\pi : [n] \to [n]$ such that for every $i \in [n]$, it holds that $Y_i = X_{\pi(i)}^k$. We say $x \in \textsc{Shuffle}$ if x is an an encoding of $X_1,\ldots X_n, Y_1,\ldots,Y_n$, and K where $Y_1,\ldots Y_n$ is a shuffling of $X_1,\ldots,X_n$. The protocol in figure 3 specifies a zero-knowledge proof for Shuffle.

3.4 Applying the protocol to mixnets

Consider the following algorithm for a mixnet constructed from the ElGamal encryption scheme:

$\underline{\text{Algorithm } \textsc{Mix}_k\big((C_i,R_i)_{i=1}^n\big)}$

$\pi \twoheadleftarrow \mathsf{Perm}(n)$
For $i \leftarrow 1$ to n do
  $(M_i,S_i) \leftarrow \mathcal{D}_k(C_{\pi(i)},R_{\pi(i)})$
Return $(M_i,S_i)_{i=1}^n$

Let $K = g^k$ denote the corresponding public key. For every $i \in [n]$, we have that $S_i = R_{\pi(i)}^k$ and $M_i = C_{\pi(i)} \cdot S_i^{-1}$. To prove that the mix didn't cheat, that is C_i is the ciphertext corresponding to the message $M_{\pi(i)}$, it suffices to prove $S_i = R_{\pi(i)}^k$. This is nothing more than an instance of Shuffle. The output of the mix can be verified by running $\textsc{Shuffle}_{\langle P,V\rangle}\big((R_1,\ldots,R_n),(S_1,\ldots,S_n),K\big)$.

4 Implementation

I implemented the simple n-shuffle of section 3.2 in Go. The code is available online at *github.com/cjpatton/shuffle*. I wrote a pair of functions `Shuffle0Prove` and `Shuffle0Verify`, which implement the prover and verifier roles of the SHUFFLE0 proof respectively. They use a standard Go channel for communication. This makes it convenient to adapt the code for a real-world application using a network socket for example.

My goal for this project was to implement the general n-shuffle and adapt it for the mixnet setting; I skipped it because of time constraints. Coding this algorithm ended up being technically difficult. In particular, there were a number of details about the computations left implicit in the paper that I had to figure out. For example, the system of equations in equation (1) in my implementation differ slightly from those presented in the paper, so I ended up doing some math-hacking to get it to work. Once I got SILMPP straightened out, implementing SHUFFLE0 was pretty straight-forward.

References

[1] David Chaum, 1981. "Untraceable electronic mail, return addresses, and digital pseudonyms." `http://freehaven.net/anonbib/cache/chaum-mix.pdf`

[2] Taher Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms." Appeared in *CRYPTO'84.*

[3] D. Chaum and T.P. Pedersen. "Wallet databases with observers." Appeared in *CRYPTO'92.*

[4] Yiannis Tsiounis and Moti Yung 1998. "On the security of ElGamal based encryption." Appeared in *PKC'98.*

[5] Oded Goldreich, 2001. *Foundations of Cryptography.*

[6] Andrew Neff. "A verifiable secret shuffle and its application to e-voting." Proceedings of the 8th ACM conference on Computer and Communications Security. ACM, 2001.

[7] Jan Camenisch and Anton Mityagin, 2006. "Mix-network with stronger security."

$\textsc{Shuffle}_{\langle P,V\rangle}\big((X_1,\cdots,X_n),(Y_1,\ldots,Y_n),K\big)$:

P1. Let $e_{1,1},\ldots,e_{1,n},e_{2,1},\ldots,e_{2,n},d \twoheadleftarrow [q-1]$. For each $i\in[n]$, let $E_{1,i}=g^{e_{1,i}}$, $E_{2,i}=g^{e_{2,i}}$, and $D=g^d$. Send $(E_{1,1},\ldots E_{1,n})$, $(E_{2,1},\ldots E_{2,n})$, and D to V.

V1. Let $f_{1,1},\ldots,f_{1,n},f_{2,1},\ldots,f_{2,n} \twoheadleftarrow [q-1]$ and send $(f_{1,1},\ldots f_{1,n})$ and $(f_{2,1},\ldots f_{2,n})$ to P.

P2. For every $j\in[2]$ and $i\in[n]$, let $\alpha_{j,i}=d\cdot f_{j,\pi^{-1}(i)}\cdot e_{j,\pi^{-1}(i)}$, and $F_{j,i}=g^{\alpha_{j,i}}$. Send $F_{1,1},\ldots,F_{1,n}$ and $F_{2,1},\ldots,F_{2,n}$ to V.

V2. Let $\gamma \twoheadleftarrow [q-1]$. Send γ to P.

P3. For every $j\in[2]$ and $i\in[n]$, let $\beta_{j,i}=f_{j,i}\cdot e_{j,i}$, $U_i=g^{\beta_{1,i}+\gamma\beta_{2,i}}$ and $V_i=g^{\alpha_{1,i}+\gamma\alpha_{2,i}}$. Run

$$\textsc{Shuffle0}_{\langle P,V\rangle}\big((U_1,\ldots,U_n),(V_1,\ldots,V_n),g,D\big)$$

P4. Let $a_1,\ldots,a_n,b_1,\ldots,b_{n-1} \twoheadleftarrow [q-1]$ and

$$b_n=v_n^{-1}\Big(\sum_{i=1}^{n-1}b_iv_i-d\sum_{i=1}^{n-1}a_iu_i\Big)$$

where $u_i=\log U_i$ and $v_i=\log V_i$. For every $i\in[n]$, let $A_i=g^{a_i}$ and $B_i=g^{b_i}$. Send $(A_1,\ldots,A_n)$ and $(B_1,\ldots,B_n)$ to V.

V3. Let $\lambda \twoheadleftarrow [q-1]$. Send λ to P.

P5. For every $i\in[n]$, let $s_i=a_i+\lambda u_i$, $r_i=b_i+\lambda v_i$, $P=\prod_{i=1}^n X_i^{r_i}$, and $Q=\prod_{i=1}^n Y_i^{s_i}$. Send P and Q to V.

P6. Run $\textsc{Silmpp}_{\langle P,V\rangle}\big((P,Q),(D,K)\big)$

V4. Accept if V accepts in steps P3 and P6 and for every $i\in N$, $g^{s_i}=A_i\cdot U_i^{\lambda}$ and $g^{r_i}=B_i\cdot V_i^{\lambda}$.

Figure 3: The general n-shuffle, an interactive proof system for Shuffle. The prover knows the secret key k and the permuation π.