Efficient Shuffle Arguments

Research Seminar Project

Janno Siim

Introduction

Anonymous communication is useful in variety of online applications. E-voting, online chat software and data collection for surveying are some of the applications where anonymity is an important feature.

One way to guarantee anonymous network communication is using a mix network (mix-net) protocol. A mix network is a protocol that contains several mix servers. A mix server collects user's ciphertexts, cryptographically transforms and permutes them and then sends the ciphertexts to the destination or to another mix server.

Shuffling guarantees that an adversary eavesdropping on the network cannot connect source of the ciphertext to the destination as long as at least one of the mix servers is honest.

It is highly important that a mix server does the permutation correctly. For example in e-voting it would disastrous if a malicious mix server could change votes at will. At the same time revealing the permutation would defeat the purpose of a mix network.

Solution is that the mix server must produce a zero-knowledge proof that the permutation was done correctly. Mix server must prove that the output ciphertexts are rerandomized and permuted input ciphertexts without revealing anything about the permutation.

This report gives an overview of two efficient zero-knowledge arguments for shuffles. Section 1 gives a summary of mix-nets. Section 2 covers preliminaries needed for zero-knowledge arguments. In section 3 we look at interactive argument for shuffles introduced by Neff in article [1]. In section 4 we study a shuffle protocol by Groth from [3].

1 Mix Network

Mix networks were first introduced as a way to provide anonymity by Chaum in 1981 in [4]. Since then mix networks have been an active area of research.

Mix networks can have different topologies. Simplest is the cascade topology where there is a fixed sequence mix servers. First mix server collects a batch of ciphertexts, permutes and obfuscates them and sends to a next mix server that does the same. Last mix server sends ciphertexts to the recipient. This means that every message always travels through the mix network using the same path of servers.

Alternative to cascade topology is the free-routing topology where a message has the possibility to travel through different paths in the mix network. In this report only the cascade topology is considered.

Another way to categorize mix-nets is through the obfuscation phase in the mix servers. Two common ways of doing it is either by reencrypting or decrypting the ciphertexts.

To describe those two methods let us first introduce some notation. We assume that mix network uses some public key cryptosystem.

Let there be k mix servers and n users who want to send a message. Let c_i be the message that i-th user wants to send to the recipient with address a_i where $i = 1, \ldots, n$. We assume that c_i is a ciphertext already encrypted with recipient a_i's public key.

1.1 Decryption Mix Network

Let pk_j be the public key of the j-th mix server for $j = 1, \ldots, k$.

In the decryption mix-net each user first encrypts his message with the recipients public key and then appends recipients address to the ciphertext. After that he encrypts the result with each of the mix server's public keys. We get the following ciphertexts for $i = 1, \ldots, n$

$$c_i = E_{pk_1}(E_{pk_2}(\ldots(E_{pk_k}(c_i \| a_i)))).$$

All the ciphertexts are sent to the first server. Server decrypts the first layer and then shuffles the resulting ciphertexts. New ciphertexts are sent to the second mix server that repeats the same process. When k-th server decrypts his layer of the ciphertext then also the recipient of the message is revealed. Illustration of this method can be seen on Figure 2.

In decryption network sender of the message is able to trace his message through the mix network because he knows all the intermediate ciphertext layers. Depending on the application this might or might not be a preferable property.

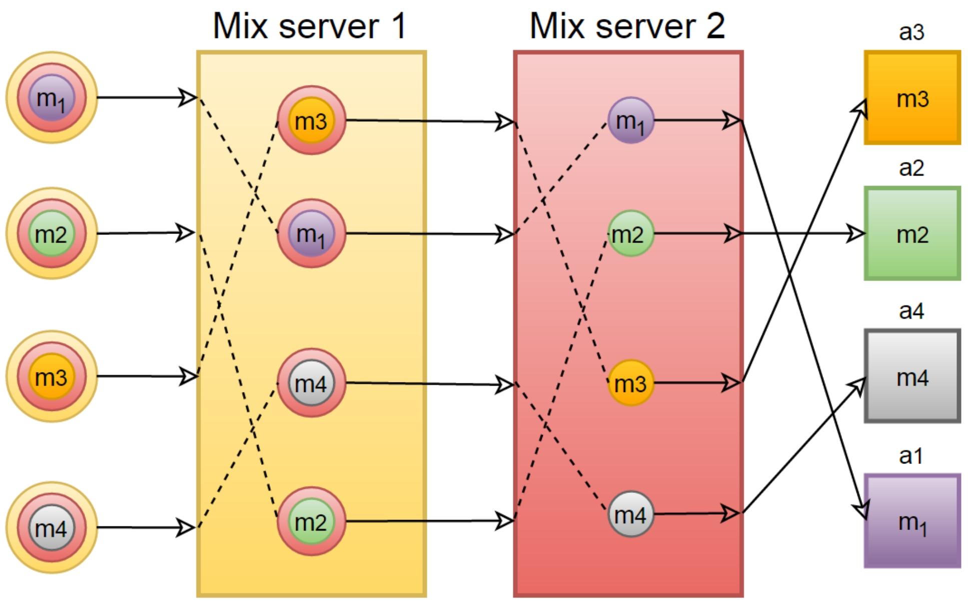

Figure 1: Illustration of the idea of decryption mix-net. Messages m_1, m_2, m_3, m_4 are sent to corresponding recipients a_1, a_2, a_3, a_4. Circles represent ciphertexts and each color corresponds to a public key.

1.2 Rerandomization Mix Network

In the following we assume that the cryptosystem has homomorphic encryption function i.e.

$$E_{pk}(m_1, r_1) \cdot E_{pk}(m_2, r_2) = E_{pk}(m_1 \cdot m_2, r_1 + r_2).$$

ElGamal is one cryptosystem that has this property.

Let pk be the mix servers public key. Users $i = 1, \ldots, n$ send the following ciphertexts to the first mix server

$$c_i^* = E_{pk}(c_i || a_i, r_i)$$

where r_i is some randomization value.

First mix sever shuffles the ciphertexts and then rerandomizes by calculating

$$c_i^* \cdot E_{pk}(1, r_1^*) = E_{pk}((c_i || a_i) \cdot 1, r_i + r_1^*) = E_{pk}(c_i || a_i, r_i + r_1^*).$$

Rerandomized ciphertexts are sent to the next mix server that repeats the process. In the k-th server ciphertexts are decrypted in some distributed manner. This reveals the recipients and the messages can be delivered.

Rest of the report assumes that mix-nets use rerandomization. Previously described protocol does not guarantee that mix servers do the shuffling and rerandomization correctly. This report covers two protocols that allow shuffling and rerandomization to be done verifiably without revealing anything excepts that mix server followed the protocol.

2 Preliminaries

In the following we denote with Σ_k the set of permutations on the set $\{1, \ldots, k\}$ i.e.

$$\Sigma_k := \{\pi : \{1, \ldots, k\} \to \{1, \ldots, k\} \mid \pi \text{ is a bijective function}\}.$$

We are going to extensively use the cryptosystem defined next.

Definition 1. *ElGamal encryption system contains the following four components:*

> ***Setup*** *Suitable cyclic group $\mathcal{G}$ of order q is chosen. Generator G of a group $\mathcal{G}$ is chosen.*
>
> ***Key generation*** *Secret key s is picked randomly from $\mathbb{Z}_q$. Public key $h = G^s$ is published.*
>
> ***Encryption*** *Let $m \in \mathcal{G}$ be the message. Random value r from $\mathbb{Z}_q$ is chosen. Encryption function E is defined as*
>
> $$E(m, r) := (m \cdot h^r, G^r).$$
>
> ***Decryption*** *Let (c_1, c_2) be a ciphertext. Decryption function D is defined as*
>
> $$D(c_1, c_2) := \frac{c_1}{c_2^s} = \frac{m \cdot h^r}{(G^r)^s} = \frac{m \cdot h^r}{h^r} = m.$$

ElGamal k-Shuffle Problem: Suppose $(X_1, Y_1), \ldots, (X_n, Y_n)$ are the input ElGamal pairs and $(\bar{X}_1, \bar{Y}_1), \ldots, (\bar{X}_n, \bar{Y}_n)$ output pairs known both to prover P and verifier V. Input pairs are encrypted using public generator g and public key h. Prover knows in addition values $\beta_1, \ldots, \beta_n$ and permutation π such that for $i = 1, \ldots, n$

$$(\bar{X}_i, \bar{Y}_i) = (g^{\beta_i} \cdot X_{\pi(i)}, h^{\beta_i} \cdot Y_{\pi(i)}).$$

P must prove to verifier that these relations hold without revealing anything about $\pi \in \Sigma_k$ or $\beta_1, \ldots, \beta_n$.

2.1 Zero-Knowledge

Definition 2. *Let κ be a function from bit strings to the interval $[0, 1]$. The protocol (P, V) is said to be a proof of knowledge for the relation R with knowledge error κ, if following conditions hold:*

> ***Completeness*** *On common input x, if the honest prover P has a private input ω such that $(x, \omega) \in R$, then the verifier V always accepts.*
>
> ***Knowledge soundness*** *There exists a probabilistic algorithm M called knowledge extractor. This M gets input x and rewindable black-box access to the prover and attempts to compute ω such that $(x, \omega) \in R$. We require that the following holds: for any prover P^*, let $\epsilon(x)$ be the probability that V accepts on input x. There exists a constant c such that whenever $\varepsilon(x) > \kappa(x)$, M will output a correct ω in expected time at most*
>
> $$\frac{|x|^c}{\varepsilon(x) - \kappa(x)}$$
>
> *where access to P^* counts as one step only.*

Definition 3. *Proof of knowledge protocol (P, V) for relation R is said to be zero-knowledge if for every probabilistic polynomial time verifier V^*, there is a simulator M_{V^*} running in expected probabilistic polynomial time, such that we have $M_{V^*} \sim^c (P, V)$ on input x and some value δ. Value δ is accessible to V^* only.*

Here $\sim^c$ denotes that two distribution are computationally indistinguishable.

3 Neff's Protocol

In this section we study a verifiable shuffle protocol proposed by Neff in [1] and [2]. Both papers contain the same protocol idea but [1] contains some significant mistakes that are addressed in [2].

First let us solve a simplified version of the previously described ElGamal shuffle problem.

Simple k-Shuffle Problem: Let $<G>= \mathcal{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order q where both p and q are primes and G is a publicly known generator of $\mathcal{G}$. Suppose Γ, input sequence $X_1, \ldots, X_k$ and output sequence $Y_1, \ldots, Y_k$ are publicly known elements of $\mathcal{G}$.

Prover P knows in addition values $\gamma = \log_G \Gamma$ and $x_i = \log_G X_i$, $y_i = \log_G Y_i$ for $i = 1, \ldots, k$. Prover also knows a permutation $\pi \in \Sigma_k$.

Prover P is required to convince verifier V that for $i = 1, \ldots, k$

$$Y_i = X_{\pi(i)}^{\gamma} \tag{1}$$

without revealing anything about γ, π, x_i or y_i.

This problem can be solved with the following 4 round public coin protocol.

Simple k-Shuffle Protocol:

1. V picks a random value t from $\mathbb{Z}_q$ and sends it to P.

2. For $i = 1, \ldots, k$ prover P computes

$$\hat{x}_i = x_i - t$$

$$\hat{y}_i = y_i - t\gamma.$$

P picks randomly and independently $2k-1$ elements $\theta_1, \ldots \theta_{2k-1}$ from $\mathbb{Z}_q$ and computes for $i = 1, \ldots, 2k$ values

$$\Theta_i = \begin{cases} G^{-\theta_1 \hat{y}_1} & \text{if } i = 1 \\ G^{\theta_{i-1}\hat{x}_i - \theta_i \hat{y}_i} & \text{if } 2 \le i \le k \\ G^{\gamma\theta_i - \theta_{i+1}} & \text{if } k+1 \le i \le 2k-1 \\ G^{\gamma\theta_{2k-1}} & \text{if } i = 2k. \end{cases}$$

P sends $\Theta_1, \ldots, \Theta_{2k}$ to V.

3. V pick a random value c from $\mathbb{Z}_q$ and sends it to P.

4. P computes values $\alpha_1, \ldots, \alpha_{2k-1}$ as follows

$$\alpha_i = \begin{cases} \theta_i + c\prod_{j=1}^{i}(\frac{\hat{x}_j}{\hat{y}_j}) & \text{if } 1 \le i \le k \\ \theta_i + c\gamma^{i-2k} & \text{if } k+11 \le i \le 2k-1 \end{cases}$$

and sends them to V.

5. V computes

$$U = G^{-t}$$
$$W = \Gamma^{-t}$$
$$\hat{X}_i = X_i U$$
$$\hat{Y}_i = Y_i W$$

for $i = 1, \ldots k$.

V accepts only if all of the following equations hold

$$\begin{aligned}
\hat{X}_1^{c} \hat{Y}_1^{-\alpha_1} &= \Theta_1 \\
\hat{X}_1^{\alpha_1} \hat{Y}_1^{-\alpha_2} &= \Theta_2 \\
&\vdots \\
\hat{X}_i^{\alpha_{i-1}} \hat{Y}_i^{-\alpha_i} &= \Theta_i \\
&\vdots \\
\hat{X}_k^{\alpha_{k-1}} \hat{Y}_k^{-\alpha_k} &= \Theta_k \\
\Gamma^{\alpha_k} G^{-\alpha_{k+1}} &= \Theta_{k+1} \\
&\vdots \\
\Gamma^{\alpha_{2k-2}} G^{-\alpha_{2k-1}} &= \Theta_{2k-1} \\
\Gamma^{\alpha_{2k-1}} G^{c} &= \Theta_{2k}.
\end{aligned}$$

Figure 1 shows the messages sent between prover and verifier.

Theorem 1. *Simple k-shuffle protocol is an honest verifier zero-knowledge proof of knowledge protocol for relation (1).*

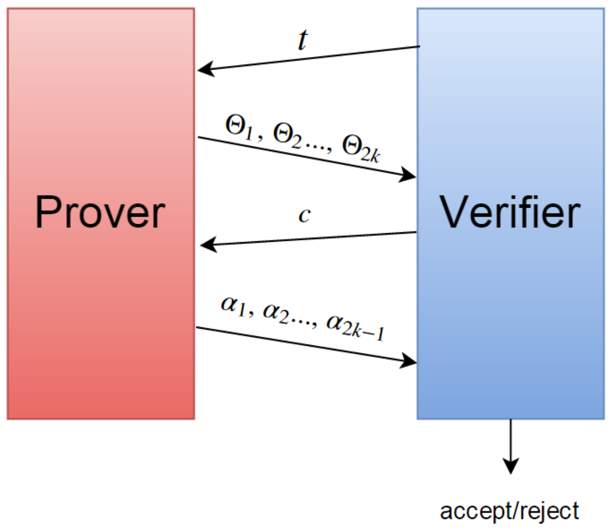
Prover
Verifier
t
$\Theta_1, \Theta_2 \ldots, \Theta_{2k}$
c
$\alpha_1, \alpha_2 \ldots, \alpha_{2k-1}$
accept/reject

Figure 2: Drawing shows the messages passed between prover and verifier in the Simple k-shuffle protocol.

4 Groth's Protocol

Conclusion

References

[1] C. Andrew Neff. 2001. *A verifiable secret shuffle and its application to e-voting.* In Proceedings of the 8th ACM conference on Computer and Communications Security (CCS '01), Pierangela Samarati (Ed.). ACM, New York, NY, USA, 116-125. DOI=`http://dx.doi.org/10.1145/501983.502000`

[2] C. Andrew Neff. 2003. *Verifiable mixing (shuffling) of elgamal pairs.* In proceedings of PET '03, LNCS series.

[3] Jens Groth. 2010. *A Verifiable Secret Shuffle of Homomorphic Encryptions.* J. Cryptol. 23, 4 (October 2010), 546-579. DOI=`http://dx.doi.org/10.1007/s00145-010-9067-9`

[4] David L. Chaum. 1981. *Untraceable electronic mail, return addresses, and digital pseudonyms.* Commun. ACM 24, 2 (February 1981), 84-90. DOI=`http://dx.doi.org/10.1145/358549.358563`