Report on **Mix Networks**

# Introduction

Mix networks (mixnets) were designed to provide anonymity, also known as untraceability, in a network communication preventing tracing back from a receiver to a sender. A popular example is a secure electronic voting where voters need to keep their identities secret after sending their encrypted votes to the system.

Over the last few decades, advances in communication networks have led to network- based personalized services that involve sensitive information.

The concept of mix networks -introduced in 1981 by David Chaum- allowed anonymity preserving application in many network communications such as electronic voting, anonymous emailing, confidentiality of medical records, prevention of online user profiling by market researchers and many other applications where privacy and anonymity matter in a great deal.

Continuous attacks on these networks have raised privacy concerns. An adversary would do a passive attack by eavesdropping and correlating messages with their senders. However, an adversary can also carry out an active attack by modifying messages (adding or deleting) in addition to tracing them back to their senders.

Weaknesses in Chaum’s mixnet “also called Decryption mixnets” led to proposing Re- encryption mixnets by Park et al, where the mix-servers use the homomorphic property of the cryptosystem to re-encrypt cipher texts instead of decrypting.

In this paper we concentrate on Re-encrption mix nets description, construction, and universal verification in the context of a voting system.

# Mix network

According to Wikipedia definition: “mix networks are routing protocols that create hard- to-trace communications by using a chain of proxy servers known as *mixes* which take in messages from multiple senders, shuffle them, and send them back out in random order to the next destination (possibly another mix node)”.

We learned that the design of a mixnet is actually based on the design of the main component which is the mix itself and its performance. To a mix be effectively providing anonymity for a batch of inputs it should do the following:

* + Use either encryption or decryption to change the appearance of inputs, and
  + A permutation on the batch of the transformed inputs.
  + Forward the batch in parallel to the next destination which removes the information about the time of arriving for each input.

The previous procedures hide the correspondence between inputs and outputs from observers, which achieves the goal of the mixnet by keeping user’s anonymity.

# Mixnet Classifications

Mixes (also called stages) are the main components in a mixnet as transformation and mixing operations take place inside them. Consequently, researchers have always classified mixnets according to:

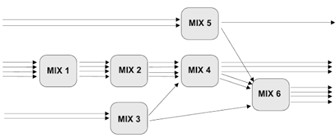
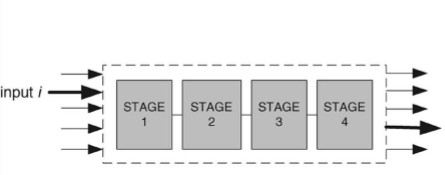
* + Mixnet Topology: how stages are interconnected, and
  + Mixing Strategy: what cryptographic operations they perform.

Topologies and strategies are explained almost similarly in different resources. We find [1] a good reference to summarize them in sections 3.1 and 3.2.

## Mixnet Topologies

Basically there are two different topologies.

* + - **Cascade mixnet**: consists of stages connected in a fixed and sequential order. All inputs will pass through each stage in order. When a batch of inputs arrives in the first stage they will be mixed and then sent to the next stage. This operation is repeated in each stage all the way to the last stage until the input batch is outputted. All messages follow exactly same path. Inputs may arrive at the first stage during an interval, but after the mixing operation the stage forwards them to the next stage in parallel, which eliminates the order of arriving. See figure (1.a).
    - **Free-route mixnet**: instead of passing a single path, input messages can pass different paths (routes) of stages. This operation requires each stage be independent from other stages -unlike cascade mixnets – which involves important operating conditions:

1. Any stage expects receiving inputs directly from the sender at different times.
2. The inputs to a stage may come from more than one connected stage. For example; in figure (1.b) stage 6 receives inputs from stages 3 and 4.
3. Any stage can forward directly to the addressed receiver. For example; in (1.b) stages 4, 5, and 6 send their outputs directly to the final receiver.
4. Each stage may wait for a batch of a fixed size or wait for a fixed amount of time before forwarding the inputs to the next stage.

**Figure 1.a and 1.b [1] [3]**

## Mixing strategies

Mixnets perform cryptographic concealment in different strategies:

## Decryption mixnets

All mix net papers refer to D. Chaum as he was the first to outline this type of mix nets, thus they are in his name called Chaumian, and this paragraph describes the point of it [2]. Each mix in this mixnet has its own pair of keys. Before sending a message, the sender must encrypt the message with the public keys for all stages. Each mix peels oﬀ one of these encryption using its private key with the decryption algorithm. The term “onion” –which was later used- visualizes this structure; the sender builds the onion, and each mix peels oﬀ one layer of the onion. Peeling off a layer changes the appearance of the message, then the mix applies a secret random permutation before forwarding to another mix.

A message *m* that would be sent through a mix net of five stages will be prepared by the sender to look like:

*m*enc = Epk1 (r1, Epk2 (r2 , Epk3 (r3 , Epk4 (r4 , Epk5 (r5 , *m*)))))

## Hybrid mixnets

In the previous decryption mixnet, a stage peels off a layer by decrypting with its private key. Hybrid mixnet is presented to achieve efficient integration of public-key and symmetric-key operations. Each stage obtains a symmetric key that it uses to decrypt the received onion before forwarding it to the next stage.

## Re-encryption mixnets

Research projects published later listed some weaknesses in Decryption and Hybrid mix nets that imposed more efficient ways to encrypt messages. Main weaknesses are:

1. The size of the onions decreases as the stages are traversed.
2. The sender is required to encrypt for each stage.
3. The decryption is performed in a predetermined (by sender of onion) sequence of stages Re-encryption mixnet is a more efficient mixnet was proposed to address the weaknesses

listed.

# Re-encryption mixnet: Description and Underlying Cryptosystem

## General Description

In this paragraph, we briefly summarize the work of re-encryption mix nets as described in [4]. On a high level, a mixnet works as follows. A sender only needs to perform a single encryption for all the stages using the public key of the mixnet, computed from the public keys of the stages. The senders submit their inputs encrypted with a homomorphic cryptosystem, e.g., ElGamal, to the mixnet.

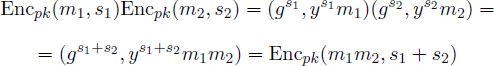
After sending the message, instead of decrypting a previously encrypted message, each stage starting from the first stage, re-encrypts and permutes the list of the ciphertexts before passing the list on to the next stage in the chain. Therefore, in this model if one node is honest in keeping the permutation secret, the passed messages will be definitely anonymous.

Once the last stage has published his output list on the bulletin board, the verification phase starts. A verifier partitions the input to each stage into a number of blocks. The stage then reveals the output block corresponding to each input block without revealing how the individual ciphertexts are shuffled. Then the server proves that the product of all the ciphertexts in each output block is a re-encryption of the product of the ciphertexts in the corresponding input block. If the verification is passed, then the stage jointly decrypt the final list of ciphertexts and otherwise the mixing restarts.

## Homomorphic Property [4]

The homomorphic property of the El Gamal Cryptosystem makes it powerful to be used in Re-encryption mixnets. In the following we briefly describe this property.

* + - Consider Gq as a group, then Gq × Gq is also a group.
    - Let’s define the group operation as (a,b (c,d) = (ac, bd) for any (a, b),(c, d) ∈Gq × Gq.
    - From this and the definition of El Gamal cryptosystem, we can deduce that it is homomorphic.
    - This means that for any two messages m1, m2 ∈Gq and randomness s1, s2 ∈Zq:



ie, the encryption of the product of two messages equals the product of the encryptions of the messages. By choosing m1 = m and m2 = 1 we get



* + - This property is used to re-encrypt an already encrypted message.
    - Since, s1 ∈ Zq and s2 ∈ *Z*q are chosen with uniform randomness, s1 + s2 ∈ Zq will be uniformly random as well.
    - So the distribution of ciphertexts encrypted once will be indistinguishable from the distribution of ciphertexts that have been reencrypted.

## Underlying Cryptosystem

In this section we will be explaining the whole operation and how El Gamal cryptosystem is used. This will be in a context of a voting system where voter’s privacy is of paramount importance. We refer to [5] and [6] for the details we include here as they were the most convenient to us, and we apply our understanding of homomorphic on the equations.

The cryptosystem is composed by three public parameters: *p, q, g*, a recipient’s public key *h,* and a recipient’s private key *x* defined in the following way:

- The modulo *p* is chosen as a large safe prime, that is *p=2q+1* and *q* is a prime number.

* *g* is a generator of *Gq*, the q-order subgroup of *Zp\**
* The private key *x* is selected from *Zq*

-The public key *h* is calculated as *h=gx mod p*

Voting options *v* are chosen to be all from the quadratic residue or quadratic non-residue *modulo p*, and this is to make them indistinguishable after decrypting. Padding could be added to a voting option that does not fit in the set.

Each voter i computes an initial **encryption** *c0,i* of its vote option *v* using a random exponent *r* in *Zq*, and posts the ciphertext on a bulletin board:

*Ench*(*vi,ri*)*= c0,i =* (*vi*·*hri mod p, gri mod p*) *=* (*c*1*0,i, c*2*0,i*)

Therefore, an encrypted voting option can be **decrypted** as

*vi = c*1*0,i* . *c*2 *-x mod p*

*0,i*

When all voters have submitted their encrypted vote options, the stages agree on an initial list *L*0=(c0,1, …, c0,N) of ciphertexts. Technical details on removing duplicates and elimination of ciphertexts with invalid proofs are not included in this report as they are not of our concern here. We assume the list contains *N* different ciphertexts.

The *j*th mix stage reads the list of ciphertexts *L*j-1 from the bulletin board, chooses a random permutation πj and random exponents *rj,1, …, rj,N* , and *m*j = 1 to maintain the integrity of the voter’s message, to **re-encrypt** the previously encrypted votes using the homomorphic property as follows

*cj,i* = *Ench*(*cj*-1*,i* , *rj,*πj (*i*)) *= Ench*(1, *rj,*πj (*i*)) . *cj*-1*,*πj (*i*)

and according to our understanding of the homomorphic property this permutation and re- encryption operation is expressed as

*cj,i* = *Ench*(1, *rj,*πj (*i*)) . *Ench*(*vi,ri*) = *Ench*(*vi*, *ri* + *rj,*πj (*i*))

= (*vi* . *h ri+rj,πj(i) mod p* , *gri+rj,πj(i) mod p*) = (*c*1*j,i* , *c*2*j,i*)

Finally, the *j*th mix stage writes *L*j = (cj,1, …, cj,N) on the bulletin board.

And that doesn’t change the **decryption** as it remains

*vi = c*1*j,i* , *c*2*j,i-x mod p*

# Universal Verifiability

While working on this report we learned that any mixing verification protocol is strongly related to the mixing scheme in the mixnet. In addition, to the same mixing scheme there might be more than one verification method. In some mixnets the verification process is implemented simultaneously with the mixing process at each stage, while in some others the verification is done after the mixing is fully completed and just before decryption. In this section we summarize the universal verification process depending on [6].

In the mixing phase, each mix stage stores in a secret way the permutation and re- encryption values applied for each vote. When the last stage has mixed and re-encrypted its inputs the decryption process can start. However, before disclosing any significant information, the correct performance of the whole mixnet is universally verified.

The concept “universal verifiability” was introduced in 1995 by Sako and Kilian. This verifiability comes from the fact that the verification of the proofs can be performed by any external verifier. This verifier can guarantee correctness of the mixnet output even if all stages have been compromised, since any incorrect proofs can be detected. The main idea is that each stage must prove that its output batch corresponds to a unique input batch, without disclosing the relationship.

## Verification Protocol

We first list the steps required for the process and then describe the whole protocol in the subsection.

1. For the first mix stage, the verifier divides randomly the input votes in groups using a grouping array that is sent to the prover (the stage).
2. Then, the verifier calculates an *Input Integrity Proof* for each group.
3. The verifier asks the prover for the output destination of the votes belonging to each group and calculates an *Output Integrity Proof* for each group.
4. The prover calculates a NIZKP (Non-interactive Zero-Knowledge)based on the re- encryption random factor in order to demonstrate that the *Output Integrity Proof* is the re- encryption of the *Input Integrity Proof* of the same group.
5. For the next node, the groups are redefined in such a way that each new group is composed of votes from different output groups in the previous node, and repeats the steps 2–5 until the correctness of the last stage is verified.

## Creating the group [6]

As said before, this group organization is done before decrypting the votes, preventing the disclosure of sensitive information to any mixing stage, which otherwise leads to cheating the verification process. When the verification process starts, the input encrypted votes of each node are divided into several independent groups following a random organization proposed by the verifier. This is done by sending an array with the indexes of the position of the votes to be grouped:

For *m* input votes: *{v1, v2, v3, ..., vm}*.

An example of a grouping array is: *{v3, vm-1, v5, ..., v2}*.

The prover organizes the input votes following the grouping array order to define each vote group contents. Then, using the pre-saved mixing permutation information, the prover indicates to the verifier for each stage output vote the group to which it belongs to. Since this is only group affiliation, it is not possible to individually correlate input and output votes.

For the next nodes, the new input groups are created by taking votes from different output groups of the previous stage.

When creating the group the right group size should be chosen as smaller groups are of high probability to detect manipulation of any vote, however, if the group is too small the votes may not be equally distributed at the last stage as the groups at the last stage are required to be composed of one vote from each group defined in the first stage.

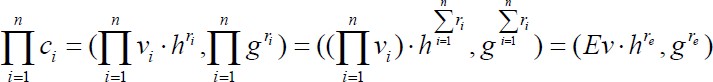
The paper suggests a formula to compute the right group size, that takes into consideration the number of the stages in the mixnet. It preserves the voters privacy and optimizes manipulation detection rates. If *t* is the number of mixnet stages (at least two) and *m* the total number of votes, the number of *n* votes inside a group should be at least:

*n* = √*t m*

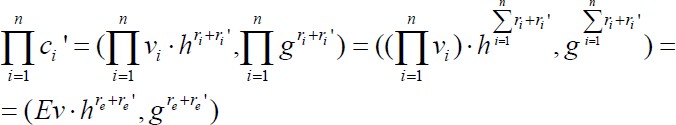
## ZKP of the Integrity Proofs [5]

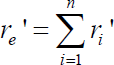
Again shows up the homomorphic property of ElGamal encryption in the verification process. Using this property, two types of integrity checks can be performed at each stage. the check is called *Integrity Proof* and it is the result of multiplying a group of votes.

When creating groups, the verifier also multiplies the votes (of number *n*) of the same group in the input of a stage to obtain an *Input Integrity Proof* as follows



After the prover indicates which votes in the output of the stage belong to which input group, the verifier can multiply the votes of the same group to obtain an *Output Integrity Proof* as follows



The accumulated re-encryption factor can be calculated using all the saved individual re- encryption factors of the votes of each group

Having this accumulated factor, the stage proves that the *Output Integrity Proof* is the re- encryption of the *Input Integrity Proof* , which proves the correctness of the stage output. This operation is called a Non-Interactive Zero Knowledge Proof of Re-encryption (NIZKP-RE) and it is repeated at each stage.

# Security

Paper [5] – which is a study on [6]- affirms that the universal verification proposed in [6]- is significantly faster than the most efficient proofs of shuffles, but it is not perfectly secure. It demonstrates attacks against both the correctness and the privacy of the mixnet. Their attacks allow replacing a few inputs or breaking privacy of a few voters, but it suggests that is enough to consider the mixnet not secure.

The paper included 2 attacks on privacy and an attack on correctness. We briefly explain the attack on correctness. We use parameters: *b* denotes the number of blocks in the verification and *l* = *N*/*b* denotes the block size, where *N* is the number of voters.

The attack requires only one corrupt stage and shows if *l* ≥*b* then it is possible to replace R= *1 b* -1 votes without being detected. The attacker corrupts a stage j so the stage replaces

√

*3*

cj-1,1, …, cj-1,R by ciphertexts of its choice u1,…uR, and replaces cj-1,R+1 by



forming a modified ciphertext list *L’*j-1 . using the equation above we notice that the products of the ciphertexts in the real and the modified lists are equal,

ie:

and the stage now re-encrypts and permutes *L’*j-1 to form *L*j. The rest of the attack contains a few details that we didn’t fully understand them but we learned that it explains how the attack goes undetected by proving that the revealed randomness is valid for both *L’*j-1 and *L*j-1.

# Conclusion

Mixnetworks is a broad concept in cryptography. We meant to stick mainly to concepts we encountered in INSE6110, namely El Gamal encryption and its homomorphic property, and Zero-Knowledge Proof that is based on Schnorr protocol, in an example of real world. Many details in the example are valid for a cascade mix-net and does not apply for free-rout mix-nets.

# References

1. Sampigethaya, Krishna, and Radha Poovendran. "A survey on mix networks and their secure applications." *Proceedings of the IEEE* 94.12 (2006): 2142-2181.
2. Chaum, David L. "Untraceable electronic mail, return addresses, and digital pseudonyms."

*Communications of the ACM* 24.2 (1981): 84-90.

1. Geir M. Koien, Vladimir A. Oleshchuk. Book: Aspects of Personal Privacy in Communications: Problems, Technology and Solutions.
2. Larsson, Erik, and Carl Svensson. "El Gamal Mix-Nets and Implementation of a Verifier." (2013).
3. Khazaei, Shahram, Björn Terelius, and Douglas Wikström. "Cryptanalysis of a Universally Verifiable Efficient Re-encryption Mixnet." *IACR Cryptology ePrint Archive* 2012 (2012): 100.
4. Allepuz, Jordi Puiggalí, Sandra Guasch Castelló, and Scytl Secure Electronic Voting. "Universally verifiable efficient re-encryption mixnet." *Electronic Voting* 167 (2010): 241-254.