

Attacks on cMix - Some Small Overlooked Details

Herman Galteland¹, Stig F. Mjølunes², and Ruxandra F. Olimid²

¹ Department of Mathematical Sciences, NTNU, Norwegian University of Science and Technology, Trondheim, Norway

`herman.galteland@math.ntnu.no`

² Department of Telematics, NTNU, Norwegian University of Science and Technology, Trondheim, Norway

`stig.mjolsnes@item.ntnu.no`, `ruxandra.olimid@ntnu.no`

Abstract Chaum et al. have very recently introduced cMix as the first practical system that offers senders-recipients unlinkability at scale. cMix is claimed by its authors to be secure unless all nodes collude. We argue their assertion does not hold for the basic description of the protocol and sustain our statement by two different types of attacks: tagging attack and insider attack. For each one, we discuss the settings that make it feasible and possible countermeasures. By this, we highlight the necessity of implementing additional mechanisms that at first have been overlooked or have only been mentioned as additional features.

Keywords: cryptographic protocols, sender-recipient unlinkability, anonymity, mixnets, attacks

1 Introduction

1.1 cMix

The cMix protocol by Chaum et al. [1] is an improved mixing network [2] which aims to provide an anonymous communication tool for its users at large scales. The mixing should be such that no one is able to relate an output message to a user input message, that is, no one is able to link a sender with a recipient. An important advantage over its predecessors is that cMix moves expensive computations (like public key encryption) to a precomputation phase, keeping the real-time phase, which is in charge with actual message delivery, fast. The protocol is meant to be a part of a larger system, called Privategrity, but its authors describe cMix independently.

cMix is claimed by its authors to be the first practical system that provides sender-recipient unlinkability, unless all nodes collude. We argue their assertion is false for the basic description of the protocol (as given in [1], Section 4) and sustain our statement by two different types of attacks. Each of them has its own influence on the design of the original protocol and highlights the necessity of implementing additional commitments or mechanisms that at first have been overlooked or have only been mentioned as additional features.

1.2 Related Work

cMix is designed to be resistant to most standard mix network attacks. As our work focuses on the cryptanalysis of cMix, we will present here a selection of proposed attacks on anonymous overlay networks, and we will later introduce in detail, in Subsection 2.2, the adversarial model from [1].

Tagging attacks are a potential threat to all mix networks [3]. Given that it is possible to recognize a valid output message, an adversary can tag a message before it is sent through the mix network and observe the tag in the output. Hence, the adversary can break the anonymity of a specific user. We show in Section 3 that cMix is vulnerable to a tagging attack.

Replay attacks are network attacks in which an adversary maliciously re-transmits a valid data transmission several times, making it possible to analyze the outgoing traffic [4]. We do not analyze replay attacks against cMix system, as they are eliminated by the adversarial model (see Subsection 2.2).

Intersection attacks and *statistical disclosure attacks* use information given by observing mix networks where the users can freely choose the mix node for their messages (free mix nodes) [4–6]. In such systems, different batches can be distinguished since they come from different mix nodes. Assuming a user use the same mix nodes for every message, the adversary can separate the routes by analyzing the network flow.

Traffic analysis attacks is a family of attacks that observes the network traffic in order to deduce informational patterns in communication and targets connection-based systems. Unlike message-based system like cMix, connection-based systems use free mix nodes that do not batch and permute messages. By *counting packets* [7] and *timing communication* [8] the adversary is able to distinguish between different paths in the (free) mix network. *Contextual attacks* [9] (or *traffic confirmation attacks* [10], or *intersection attacks* [11]) analyze the traffic when specific users and recipients use a protocol, their communication pattern, and how many messages they send and receive.

The authors of cMix acknowledge that their proposal is potentially vulnerable to attacks that make anonymous systems fail, like the broadband intersection attacks, contextual attacks, or DoS (Denial of Service) [1].

1.3 Results

We focus on the security analysis of cMix as described in [1], Section 4 and show that it is susceptible to two attacks, which differ by adversarial power and action type.

Tagging Attack. The cMix paper [1] considers tagging attacks and introduces commitments [12] to overcome this issue. However, it also states that “*tagging attacks do not work before the exit node*”, and “*if a tagging attack is detected, at least the last node should be removed from the cascade*” (see [1], Section 4.3); therefore, the authors might be aware of a possible attack performed by the exit node and if so, they do not consider any prevention. We introduce a simple

tagging attack launched by the last node. Although a prevention mechanism is immediate (by adding an extra commitment), we consider it for completeness, as an example of a possible tagging attack against the system. At the time of writing, the authors of cMix acknowledged that the actual design of the system adds the additional commitment we refer to as a countermeasure [13].

Insider Attack. Attacks are claimed unsuccessful unless all nodes collude. We contradict this and prove that the last node can break the unlinkability by, essentially, creating a mix network consisting of only itself. To succeed, the last node will deviate from the protocol rules and choose its own output. We argue that this attack remains undetected in the original version of cMix and becomes detectable only if additional checks like RPC (*Randomized Integrity Checking*, see Subsection 2.3) are considered (suggested by the authors of [1] as a special feature). On the other hand, an inappropriate use of RPC could allow a coalition of nodes (all except one) to link a large fraction of the senders to their recipients.

1.4 Outline

Section 2 describes cMix and presents the adversarial model. The two following sections contain our results: Section 3 describes a simple tag attack which is similar to the tag attack described by Chaum et al. in the original cMix paper; Section 4 presents the insider attack where the adversary controls the last node and makes the overall mixing process independent of the preceding nodes. Section 5 concludes and indicates possible future research directions.

2 Preliminaries

2.1 cMix Description

Figure 1 describes the cMix protocol from [1]. We ignore the return steps, since they are irrelevant for our attacks. Note that this does not restrict the applicability of our results, since the same permutation is used for both forward and return path. Once the permutation is disclosed, both directions of communication are compromised.

cMix has 2 phases: a precomputation phase and a real-time phase. By design, heavy public key computations are performed in the precomputation phase, which can run on a separate hardware (for each node). Since the precomputation phase does not require any input from the users, it can be performed offline. Splitting the process into two phases gives cMix an increased performance, as the heavy public key operations are being computed during the (offline) precomputation phase and can be performed while a batch is being filled up with messages.

The scheme consists mainly of a sequence of n mix nodes that process β messages at a time (a *batch* of messages); made simple, each node performs a permutation on the input and blinds the output by multiplying it with a random

Table 1. Notations

U_j	user j
\mathbf{M}	a batch of β messages $\mathbf{M} = (M_1, \dots, M_\beta)$, each M_i sent by a distinct user
N_i	node i from the set of n mix nodes $\{N_1, \dots, N_n\}$
e_i	the share of node N_i of the secret key e
d	the public key of the system, where $d = \prod_{i=1}^n g^{e_i}$
$\mathcal{E}(\cdot)$	a multi-party group-homomorphic encryption under the system public key d
π_i	a random permutation on a batch, applied by node N_i
Π_i	the composed permutation performed by all nodes from N_1 to N_i
$k_{i,j}$	the derived secret key shared between node N_i and the sending user of slot j
\mathbf{k}_i	the vector of derived secret keys shared between node N_i and all users in a batch, i.e. $\mathbf{k}_i = (k_{i,1}, \dots, k_{i,\beta})$
K_j	the product of all shared keys for the sending user of slot j , i.e. $K_j = \prod_{i=1}^n k_{i,j}$
$\mathbf{r}_i, \mathbf{s}_i$	random values of node N_i for the batch, where $\mathbf{r}_i = (r_{i,1}, \dots, r_{i,\beta})$, respectively $\mathbf{s}_i = (s_{i,1}, \dots, s_{i,\beta})$
$\mathbf{R}_i, \mathbf{S}_i$	the direct product of the first i values, i.e. $\mathbf{R}_i = \prod_{j=1}^i \mathbf{r}_j$, respectively $\mathbf{S}_i = \prod_{j=1}^i \mathbf{s}_j$

value. The last node N_n makes an exception, as it usually behaves differently from the other nodes (see Figure 1).

Besides the last node, there is another entity with a special role in the system - the *network handler* - that interacts both with the users and the whole set of nodes. The network handler receives messages from the users and arranges them into batches; once a batch is full it is sent to the first node in the mix network. After the last node performs its mixing, it sends the batch back to the network handler, which can then forward or broadcast the messages to the destination. The mixing should be such that no one is able to relate an output message to a user input message, that is, no one is able to link a sender with a recipient.

Before using the system, each sender U_j must establish a private symmetric key with each of the nodes N_i , which they use as a seed in a pseudorandom generator to derive the secret keys $k_{i,j}$. To blind a message M_j , before it is sent to the network handler, user U_j multiplies M_j with a key composed by the derived keys shared with each of the nodes $K_j = \prod_{i=1}^n k_{i,j}$. The network handler arranges messages into a batch and sends it through the mix network. Each node will apply its permutation to the batch and the last node will send it back to the network handler. The output is a permuted batch of messages.

During the mixing step of the precomputation phase, each node performs encryption under a public key of the system; the related private key is split across all nodes in the network. The encryption scheme suggested by the authors of [1] is the multi-party group-homomorphic encryption based on ElGamal [14] described by Benaloh [15]. Moreover, all computations of the protocol are performed in a prime order cyclic group G that satisfies the decisional Diffie-Hellman security assumption. We denote by G^* the set of nonidentity elements in G .

Precomputation Phase

Step 1 (preprocessing). Each node N_i , $1 \leq i \leq n$, selects a random \mathbf{r}_i , computes the encryption $\mathcal{E}(\mathbf{r}_i^{-1})$ and sends it to the network handler. The network handler computes the product of all the received values, produces $\mathcal{E}(\mathbf{R}_n^{-1}) = \prod_{i=1}^n \mathcal{E}(\mathbf{r}_i^{-1})$ and sends it to the first node.

Step 2 (mixing). Each node N_i , $1 \leq i \leq n$, computes $\pi_i(\mathcal{E}(\Pi_{i-1}(\mathbf{R}_n^{-1}) \times \mathbf{S}_{i-1}^{-1})) \times \mathcal{E}(\mathbf{s}_i^{-1})$, where Π_0 is the identity permutation and $\mathbf{S}_0^{-1} = \mathbf{1}$. The last node sends the vector of random components (i.e. the first component) of the ciphertext $(\mathbf{x}, \mathbf{c}) = \mathcal{E}((\Pi_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ to the other nodes and stores the vector of message components (i.e. the second component) locally for the real-time phase.

Step 3 (postprocessing). Using the random component \mathbf{x} , each node N_i , $1 \leq i \leq n$, computes its individual decryption share for (\mathbf{x}, \mathbf{c}) as $\mathcal{D}_i(\mathbf{x}) = \mathbf{x}^{-e_i}$, stores it locally to use in the real-time phase and publicly commits to it.

Real-Time Phase

Step 0. Each user constructs its message MK_j^{-1} (for slot j) by multiplying the message M_j with the inverse of the key K_j and it sends it to the network handler, which collects all messages and combines them to get a vector $\mathbf{M} \times \mathbf{K}^{-1}$.

Step 1 (preprocessing). Each node N_i , $1 \leq i \leq n$, sends $\mathbf{k}_i \times \mathbf{r}_i$ to the network handler, which uses them to compute $\mathbf{M} \times \mathbf{R}_n = \mathbf{M} \times \mathbf{K}^{-1} \times \prod_{i=1}^n \mathbf{k}_i \times \mathbf{r}_i$ and sends the result to N_1 .

Step 2 (mixing). Each node N_i , $1 \leq i \leq n$, computes $\pi_i(\Pi_{i-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{i-1}) \times \mathbf{s}_i$, where Π_0 is the identity permutation and $\mathbf{S}_0 = \mathbf{1}$. The last node N_n sends a commitment to its message $\Pi_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n$ to every other node.

Step 3 (postprocessing). Each node N_i , $1 \leq i \leq n-1$, sends its precomputed decryption share for $(\mathbf{x}, \mathbf{c}) = \mathcal{E}((\Pi_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ to the network handler, while the last node N_n sends its decryption share multiplied by the value in the previous step and the message component: $\Pi_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \mathcal{D}_n(\mathbf{x}) \times \mathbf{c}$. Finally, the network handler retrieves the permuted message as $\Pi_n(\mathbf{M}) = \Pi_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \prod_{i=1}^n \mathcal{D}_i(\mathbf{x}) \times \mathbf{c}$.

Figure 1. The cMix Protocol (forward path) [1]

Refer to Figure 1 for the detailed description of cMix, which is self-contained by the notations in Table 1.

2.2 Adversarial Model

The adversarial model in [1] assumes authenticated channels among the mix nodes and between the network handler and each mix node. This implies that the adversary can eavesdrop, forward and delete messages, but he cannot modify, inject or replay messages without detection. The adversary can compromise both users (up to all except two) and mix nodes (up to all except one). Compromised

nodes can behave malicious but cautious, as the attacker aims to remain undetected. Within this attacker model, the authors of cMix claim that the outputs are unlinkable to the inputs, even if the adversary knows the set of senders and the set of recipients for every batch of messages.

The security analysis in Appendix A of the cMix paper assures secure authenticated channels, for which the adversary can eavesdrop on the sender, the recipient, and the length of a message, but cannot read its content. All our attacks hold under these stronger security assumptions.

2.3 Features and Extensions of cMix

The cMix paper [1] dedicates a section to special features and extensions of the system, where it shortly discusses the utility of adding RPC (*Randomized Integrity Checking*) to cMix. Introduced in 2002 by Jacobsson, Juels and Rivest [16], RPC is an integrity check mechanism that has continued to be analyzed and developed even recently [17, 18]. The idea of RPC is that each node commits to a randomly chosen permutation, publishes its input and output, and proves that it have followed the protocol correctly by giving a (large) fraction of its secret information, validating that an input/output pair is correct. That is, show that a (large) fraction (selected by the other nodes or by a random oracle) of their permutations is correct. As it is described in the cMix paper, two adjacent nodes can be paired up and reveal their secret information such that none of the messages can be followed though a pair as an input of the first node and as the output of the second node. This method will probabilistically detect whether any node changes the messages of a batch, while it keeps secret the permutation through the two nodes.

3 The Tagging Attack

Our first attack is similar to the tag attack described in the cMix paper [1], but it uses a different value to remove the tag. During the precomputation phase the nodes compute the value $(\mathbf{x}, \mathbf{c}) = \mathcal{E}((\Pi_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$, where the last node stores the vector of message components \mathbf{c} locally and sends the vector of random components \mathbf{x} to all other nodes. Each node computes its decryption share using \mathbf{x} and commits to this value. Note that \mathbf{c} is not being committed to in the original protocol.

The authors of cMix introduce commitments to detect potential tagging attacks, exposing any attempt of using the decryption shares to remove the tag. However, the commitments are independent of \mathbf{c} , so it is possible to perform a similar attack which uses \mathbf{c} instead of $\mathcal{D}_n(\mathbf{x})$ to remove the tag. The downside is that the adversary needs to corrupt the last node (since it is the only node that has access to the message component) and the network handler (under the assumption of secure authorized channels). Figure 2 describes the tag attack performed by N_n on one of the users.

Goal: Tag a message M_j belonging to user U_j and recognize it in the permuted batch of messages, linking the sender U_j to its recipient.

Step 1. The corrupted node N_n creates a tag vector \mathbf{t} which consists of $\beta - 1$ ones and one tag $t \in G^*$ in slot j (i.e. $\mathbf{t} = (1, \dots, 1, t, 1, \dots, 1)$), computes $\mathbf{k}_n \times \mathbf{r}_n \times \mathbf{t}$ and sends it to the network handler (**Real-time Phase.Step 1**).

Step 2. The network handler sends the set of all decryption shares $\{\mathcal{D}_i(\mathbf{x}) | 1 \leq i < n\}$ to the last node (**Real-time Phase.Step 3**). Node N_n can retrieve the tagged and permuted messages as $\Pi_n(\mathbf{M} \times \mathbf{t}) = \Pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{t}) \times \mathbf{S}_n \times \prod_{i=1}^n \mathcal{D}_i(\mathbf{x}) \times \mathbf{c}$ and recognize the tagged message in slot j' .

Step 3. The corrupted node N_n creates the inverse tag vector \mathbf{t}^{-1} , which consists of $\beta - 1$ ones and one tag $t^{-1} \in G^*$ in slot j' , computes $\mathbf{c}' = \mathbf{c} \times \mathbf{t}^{-1}$ and sends $\Pi_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \mathcal{D}_n(\mathbf{x}) \times \mathbf{c}'$ to the network handler.

Figure 2. The Tagging Attack

In order for our tag attack to be successful we need to assume that it is possible to recognize valid messages in the output. To tag a message M_j the last node create a tag vector $\mathbf{t} = (1, \dots, t, \dots, 1)$, multiplies it with the keys and random values $\mathbf{k}_n \times \mathbf{r}_n \times \mathbf{t}$, and sends the result to the network handler. The tag then goes through the mixnet attached to message M_j and arrives at the last node as $\Pi_{n-1}(\mathbf{M} \times \mathbf{R}_n \times \mathbf{t}) \times \mathbf{S}_{n-1}$. The last node can then permute, do the computations accordingly to the protocol, and publish its commitment to the value $\Pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{t}) \times \mathbf{S}_n$. This triggers all other nodes to send their decryption share to the network handler, which forwards them to N_n . The last node can then retrieve the batch of permuted messages and find the invalid message $M_j t$ in slot j' of the permuted batch. The last node creates the inverse tag \mathbf{t}^{-1} , which has t^{-1} in slot j' , and replaces the message components with the altered value $\mathbf{c}' = \mathbf{c} \times \mathbf{t}^{-1}$. The network handler then computes

$$\Pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{t}) \times \mathbf{S}_n \times \mathbf{c}' \times \prod_{i=1}^n \mathcal{D}_i(\mathbf{x}) =$$

$$\Pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{t}) \times \mathbf{S}_n \times (\Pi_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1} \times \mathbf{t}^{-1} = \Pi_n(\mathbf{M} \times \mathbf{t}) \times \mathbf{t}^{-1} = \Pi_n(\mathbf{M})$$

and deliver the permuted batch as normal. That is, the adversary has successfully linked a sender with a recipient without being detected.

To make this attack detectable, the last node should publish a commitment to the vector of message components \mathbf{c} in the **Precomputation Phase.Step 3** or the system should implement RPC as an integrity check mechanism. Although prevention can be simply achieved by natural solutions like the ones mentioned, we introduce the attack for completeness; it stands as an example of tagging attack performed by the last node, a type of attack the authors of cMix seem to be aware of (see [1], Section 4.2: *"tagging attacks do not work before the exit*

node” and “if a tagging attack is detected, at least the last node should be removed from the cascade”).

At the time of writing, the authors of cMix acknowledged that the actual design of the system implements the countermeasure we refer to and commits to the vector of message components \mathbf{c} , as explained above [13].

4 The Insider Attack

Our second attack allows the last node to cancel all permutations introduced by the previous nodes and perform the overall mixing process by itself. Hence, the output of the real-time phase will be a batch of messages permuted with a known permutation, making it easy to link all senders and recipients. To succeed, the adversary needs to corrupt the last node (which knows the permutation) and the network handler (which knows the content of the values $\mathcal{E}(\mathbf{R}_n^{-1})$ and $\mathbf{M} \times \mathbf{R}_n$, under the assumption of secure authenticated channels). Figure 3 describes the insider attack.

During **Precomputation Phase.Step 1** the corrupted network handler computes and sends $\mathcal{E}(\mathbf{R}_n^{-1})$ to the first and the last node. The honest nodes operate as normal, where the last, dishonest, node discards the input it receives from the previous node and choose its own output. The last node draws a random vector $\mathbf{A} = (A_1, \dots, A_\beta)$, encrypts the inverted values, $\mathcal{E}(\mathbf{A}^{-1})$, and computes $\pi_n(\mathcal{E}(\mathbf{R}_n^{-1}) \times \mathcal{E}(\mathbf{A}^{-1})) = \pi_n(\mathcal{E}(\mathbf{R}_n^{-1} \times \mathbf{A}^{-1}))$. The last node publishes the random components of $\pi_n(\mathcal{E}(\mathbf{R}_n^{-1} \times \mathbf{A}^{-1}))$ to the other nodes such that they can prepare their decryption shares.

In **Real-Time Phase.Step 1** the network handler sends $\mathbf{M} \times \mathbf{R}_n$ to the first and the last node. In the mixing step the last node will discard what it receives from the previous node, selects its output $\pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A})$, commits to this batch of messages, and send $\pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A}) \times \mathbf{c} \times \mathcal{D}_n(\mathbf{x})$ to the network handler. As the network handler receives the decryption shares from the other nodes, it can retrieve the permuted messages and forward them correctly:

$$\pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A}) \times \mathbf{c} \times \prod_{i=1}^n \mathcal{D}_i(\mathbf{x}) = \pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A}) \times \pi_n(\mathbf{R}_n^{-1} \times \mathbf{A}^{-1}) = \pi_n(\mathbf{M}).$$

Note that the output batch is only permuted with the permutation π_n which is known to the last node. Hence, the adversary can easily deanonymize all of the senders by applying π_n^{-1} to the output.

The RPC mechanism ensures with high probability that each node follows its instructions; hence, this will prevent the last node from deviating from the protocol. Since our insider attack changes the entire batch, RPC will detect the attack with overwhelming probability.

Notes on the implementation of RPC. In addition to the conditions from the cMix paper, the implementation of RPC need to be done with care. For example, if the adversary corrupts all except one node, the remaining honest node will

Goal: Perform the mixing process with only the last node, using only a known permutation to permute the batch of messages.

Step 1. The network handler computes and sends $\mathcal{E}(\mathbf{R}_n^{-1})$ to the first and last node (**Precomputation Phase.Step 1**). The last node discards the input it is given from the previous node and publishes the component of random elements of $\pi_n(\mathcal{E}(\mathbf{R}_n^{-1} \times \mathbf{A}^{-1}))$, for a random and invertible \mathbf{A} (**Precomputation Phase.Step 3**).

Step 2. The network handler computes and sends $\mathbf{M} \times \mathbf{R}_n$ to the first and last node (**Real-Time Phase.Step 1**). The last node discards the input it is given from the previous node, publishes a commitment to $\pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A})$ and sends $\pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A}) \times \mathbf{c} \times \mathcal{D}_n(\mathbf{x})$ to the network handler (**Real-Time Phase.Step 3**).

Step 3. The network handler retrieves the permuted batch of messages as $\pi_n(\mathbf{M}) = \pi_n(\mathbf{M} \times \mathbf{R}_n \times \mathbf{A}) \times \pi_n(\mathbf{R}_n^{-1} \times \mathbf{A}^{-1})$ and publishes it. The adversary now can easily reveal \mathbf{M} by applying π_n^{-1} .

Figure 3. The Insider Attack

reveal a (large) portion of its permutation, which exposes the senders of the corresponding messages. To avoid this, we propose a few changes to the original cMix protocol: let each node N_i perform two permutations, π_{2i-1} and π_{2i} , and generate two random vectors to hide both permutations, \mathbf{s}_{2i-1} and \mathbf{s}_{2i} . Thus, each node can follow the cMix protocol as if it were two nodes and form a pair all by itself. This ensures that no message can be followed as the input and output of a single node in the RPC checks. Note that each node N_i still needs only one key vector \mathbf{k}_i and one random vector \mathbf{r}_i .

5 Conclusions

We demonstrate by examples that the cMix protocol, as initially defined in its basic settings allows linkability between senders and recipients, therefore compromising the anonymity. For both classes of attacks (the tagging attack and the insider attack) we detail the actions the adversary should follow to succeed its goal and discuss possible countermeasures. All our attacks succeed in the secure authenticated channels settings, under the assumption that the adversary can corrupt the network handler (a natural assumption the authors of cMix made by default).

By discussing the attacks, we highlight the compulsory aspect of mechanisms that at first have been overlooked (e.g.: additional commitments) or have only been mentioned as additional features (e.g.: RPC integrity mechanism). At the moment of writing, the authors of cMix acknowledged the actual system implements the commitment that prevents our tagging attack.

Our current work is restricted to the theoretical exposure of the attacks against cMix as a standalone protocol. Interesting future work might include a

practical checking of the attack feasibility against the current implementation of cMix. A more ambitious work would be to analyze cMix security within the context of the larger system Privategrity.

Acknowledgements. Herman Galteland is funded by Nasjonal sikkerhetsmyndighet (NSM), www.nsm.stat.no.

References

1. Chaum, D., Das, D., Javani, F., Kate, A., Krasnova, A., de Ruiter, J., Sherman, A.T.: cmix: Anonymization by high-performance scalable mixing. Cryptology ePrint Archive, Report 2016/008 (2016) <http://eprint.iacr.org/>, version 20160530:183553 from May, 30 2016.
2. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM* **24**(2) (February 1981) 84–90
3. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding routing information. In Anderson, R., ed.: *Proceedings of Information Hiding: First International Workshop*, Springer-Verlag, LNCS 1174 (May 1996) 137–150
4. Berthold, O., Pfitzmann, A., Standtke, R. In: *The Disadvantages of Free MIX Routes and How to Overcome Them*. Springer Berlin Heidelberg, Berlin, Heidelberg (2001) 30–45
5. Danezis, G., Diaz, C., Troncoso, C. In: *Two-Sided Statistical Disclosure Attack*. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 30–44
6. Danezis, G., Serjantov, A. In: *Statistical Disclosure or Intersection Attacks on Anonymity Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 293–308
7. Serjantov, A., Sewell, P. In: *Passive Attack Analysis for Connection-Based Anonymity Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg (2003) 116–131
8. Danezis, G. In: *The Traffic Analysis of Continuous-Time Mixes*. Springer Berlin Heidelberg, Berlin, Heidelberg (2005) 35–50
9. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. In: *International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, New York, NY, USA, Springer-Verlag New York, Inc. (2001) 10–29
10. Syverson, P.F., Goldschlag, D.M., Reed, M.G.: Anonymous connections and onion routing. In: *Proceedings of the 1997 IEEE Symposium on Security and Privacy. SP '97*, Washington, DC, USA, IEEE Computer Society (1997) 44–54
11. Berthold, O., Langos, H. In: *Dummy Traffic against Long Term Intersection Attacks*. Springer Berlin Heidelberg, Berlin, Heidelberg (2003) 110–128
12. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.* **37**(2) (October 1988) 156–189
13. de Ruiter, J.: Personal communication in e-mail from July, 28 2016.
14. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: *Proceedings of CRYPTO 84 on Advances in Cryptology*, New York, NY, USA, Springer-Verlag New York, Inc. (1985) 10–18
15. Benaloh, J.: Simple verifiable elections. In: *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop. EVT'06*, Berkeley, CA, USA, USENIX Association (2006) 5–5

16. Jakobsson, M., Juels, A., Rivest, R.L.: Making mix nets robust for electronic voting by randomized partial checking. In: Proceedings of the 11th USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2002) 339–353
17. Khazaei, S., Wikström, D.: Randomized partial checking revisited. In: Proceedings of the 13th International Conference on Topics in Cryptology. CT-RSA'13, Berlin, Heidelberg, Springer-Verlag (2013) 115–128
18. Küsters, R., Truderung, T., Vogt, A.: Formal analysis of chaumian mix nets with randomized partial checking. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy. SP '14, Washington, DC, USA, IEEE Computer Society (2014) 343–358