cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations

David Chaum[1], Debajyoti Das[2], Farid Javani[3], Aniket Kate[2], Anna Krasnova[4], Joeri De Ruiter[4], and Alan T. Sherman[3]

[1] Voting Systems Institute, USA
[2] Purdue University, USA
[3] Cyber Defense Lab, UMBC, USA
[4] Radboud University, Netherlands

Abstract. We introduce cMix, a new approach to anonymous communications. Through a precomputation, the core cMix protocol eliminates all expensive real-time public-key operations—at the senders, recipients and mixnodes—thereby decreasing real-time cryptographic latency and lowering computational costs for clients. The core real-time phase performs only a few fast modular multiplications.

In these times of surveillance and extensive profiling there is a great need for an anonymous communication system that resists global attackers. One widely recognized solution to the challenge of traffic analysis is a mixnet, which anonymizes a batch of messages by sending the batch through a fixed cascade of mixnodes. Mixnets can offer excellent privacy guarantees, including unlinkability of sender and receiver, and resistance to many traffic-analysis attacks that undermine many other approaches including onion routing. Existing mixnet designs, however, suffer from high latency in part because of the need for real-time public-key operations. Precomputation greatly improves the real-time performance of cMix, while its fixed cascade of mixnodes yields the strong anonymity guarantees of mixnets. cMix is unique in not requiring any real-time public-key operations by users. Consequently, cMix is the first mixing suitable for low latency chat for lightweight devices.

Our presentation includes a specification of cMix, security arguments, anonymity analysis, and a performance comparison with selected other approaches. We also give benchmarks from our prototype.

1 Introduction

Digital messaging has become a significant form of human communication, yet currently such systems do not provide basic protections of untraceability and unlinkability of messages. These protections are fundamental to freedom of inquiry, to freedom of expression, and increasingly to online privacy. Grave threats to privacy exist from global adversaries who construct traffic-analysis graphs detailing who communicates with whom.

We introduce cMix, a new approach to anonymous communications. cMix is a new variant of fixed-cascade mixing networks (mixnets). cMix uses a precomputation phase to avoid computationally intensive public-key cryptographic operations in its core real-time protocol. Senders/clients participate only in the real-time phase. Thus, senders never perform any public-key operations. cMix has drastically lower real-time cryptographic latency than do all other mixnets. Through its use of precomputation, and through its novel key management, cMix is markedly different from all previous mixnets. Due to its lack of public-key operations in its core real-time phase, it is very well suited for applications running on light-weight clients, including chat messaging systems running on smartphones, and for applications on low-power devices .

To provide anonymity online, an alternative approach to mixnets is onion routing, such as implemented in the widely used system TOR [49]. Onion-routing systems, however, have limitations on the level of anonymity achievable: most significantly, because they route different sessions of messages along different paths and they do not perform random permutations of messages, they are vulnerable to a variety of traffic-analysis attacks—for example [13, 45], as well as intersection attacks [7, 14, 15].

By contrast, mixnets hold fundamentally greater promise to achieve higher levels of anonymity than can onion-routing systems because mixnets are resilient to traffic-analysis attacks. Specifically, since all mixnet messages travel through the same fixed cascade of mixnodes, observing the communication paths of messages within a mixnet is not useful to the adversary. Also, mixnets can process larger batches of messages than can onion-routing systems, which is important because the batch size is the size of the anonymity set. Using a fixed cascade achieves resilence against intersection attacks [7]. The main disadvantage of current mixnets is their performance, which is throttled by their use of real-time public-key cryptographic operations, which are much slower than symmetric-key operations.

cMix is a practical solution to the cryptographic latency problem. It also provides resistance to traffic analysis and intersection attacks, as do other fixed-route mixnet designs. cMix scales linearly in number of users. Our prototype implementation on Android clients demonstrates the practicality of cMix.

The main novel and significant contribution of cMix is that, by using precomputation, cMix eliminates all expensive real-time public-key operations by sender, receiver, and mixnodes in its core protocol. In cMix, the clients never perform any public-key operations when sending messages. By splitting the computational load over a computationally intensive precomputation phase and a fast real-time phase, cMix's approach can also increase throughput, when demand is non-uniform. No other mixnet has achieved these performance characteristics. Thus, cMix greatly improves real-time performance (especially latency) over all existing traditional mixnets and all re-encryption mixnets, while enjoying their strong anonymity properties.

Our main contributions are the design, preliminary analysis, and prototype implementation of cMix, a new mixnet variant that, through precomputation, achieves lower real-time computational latency than do all existing mixnets (traditional and re-encryption), while still benefiting from the strong anonymity properties of mixnets over onion-routing systems.

In the rest of this paper we review related work, provide an overview of cMix, describe the core cMix protocol, explain some protocol enhancements, provide security arguments, compare cMix's performance with that of other mixnets, present benchmarks from our prototype implementation, discuss several issues raised by cMix, and present our conclusions.

2 Related Work

We briefly review selected background and related work on mixnets, onion routing, and precomputation for mixnets.

Mix Networks. In 1981, Chaum [11] introduced the concept of mixnets (often referred to as *decryption mixnets*) and gave basic cryptographic protocols whereby messages from a set of users are relayed by a sequence of trusted intermediaries, called *mixnodes* or *mixes*. A mixnode is simply a message relay (or proxy) that accepts a batch of encrypted messages, decrypts and randomly permutes them, and sends them on their way forward. The sender in a decryption mixnet must perform a number of public-key operations equal to the number of mixnodes. The length of the encrypted message is proportional to this number, and the length of the plaintext message is restricted for performance reasons.

Hybrid mixnets allow plaintext messages to have arbitrary length, by combining asymmetric and symmetric cryptography. First proposed in 1985 by Pfitzmann and Waidner [38], hybrid mixnets share a session key in the first message, and then use a stream cipher to encrypt further messages. Various proposals based on block ciphers followed [27,34]. The recent system called Riffle [32] provides both sender and receiver anonymity by using verifiable shuffles and private information retrieval. Periodocially, a client and all servers perform verifiable shuffles to exchange session keys, which are then used for several rounds in a manner similar to that performed by other hybrid networks.

In 1994, Park et al. [37] introduced *re-encryption mixnets*. Taking advantage of homomorphic properties of El-Gamal encryption, each mixnode re-encrypts the incoming message instead of decrypting it as in the original mixnet. Doing so reduces and fixes the size of the ciphertext message traveling through the mixnet. *Universal re-encryption mixnets* [24] do not require mixnodes to know public keys for re-encryption. Because the sender encrypts each message using the public key of the receiver, only the receiver is able to read the plaintext. Consequently, unlike other mixnets, universal re-encryption mixnets provide sender anonymity only against external observers, and not against message recipients.

Precomputation mixnets introduce a precomputation phase to decrease latency during message delivery. Jakobsson [26] introduced precompution to reduce the cost of node operations in re-encryption mixnets, though client costs remain the same. Adida and Wikström [1] considered an offline/online approach to mixing. Their protocol still requires several public-key operations in the online phase, and senders have to perform public-key operations. A notable distinction of the cMix precomputation mixnet is its shifting of *all* public-key operations in its core protocol to the precomputation phase. Moreover, only the mixnodes perform these public-key operations; no sender is involved.

Onion Routing. Higher latency of traditional mixnets can be unsatisfactory for several communication scenarios, such as web search or instant messaging. Over the past several years, a significant number of *low-latency* anonymity networks have been proposed [2, 3, 10, 12, 21, 29, 30, 35, 43], and some have been extensively employed in practice [16,49].

Common to many of them is *onion routing* [22,36], a technique whereby a message is wrapped in multiple layers of encryption, forming an *onion*. A common realization of an onion-routing system is to arrange a collection of onion routers (abbreviated ORs, also called hops or nodes) that relay traffic for users of the system. Users then randomly choose a path with few edges through the network of ORs and construct a circuit—a sequence of nodes that will route traffic. After the OR circuit is constructed, each of the nodes in the circuit shares a symmetric key with the anonymous user, which key is used to encrypt the layers of future onions. Upon receiving an onion, each node decrypts one of the layers, and forwards the message to the next node. Onion routing as it typically exists can be seen as a form of three-node mixing.

Low-latency anonymous communication networks based on onion routing [19, 28, 33, 47], such as TOR [49], are susceptible to a variety of traffic-analysis attacks. By contrast, mixnet methodology ensures that the anonymity set of a user remains the same through the communication route and makes our protocol resistant to these network-level attacks.

There are similarities between our precomputation phase, which uses public-key operations, and the circuit-construction phase of onion routing. Similarly, there are similarities between our real-time phase, which uses symmetric-key operations, and the onion wrapping and unwrapping phases.

Unlike onion routing, however, our precomputation phase requires no participation from the senders. During enrollment, each of our senders establishes a separate shared secret with each mixnode, but this key establishment is performed infrequently. Furthermore, in contrast with onion routing, our senders do not perform anonymous key agreement [3,16,21,30] using a telescoping approach or layered public-key encryption; they can establish these keys using a Diffie-Hellman key exchange. These differences result in a significant reduction in the computation that the users need to perform and make our system more attractive to energy-constrained devices such as smartphones.

3 System Overview

Before defining cMix's core protocol, we first explain our architecture and communication model, adversarial model, and security goals.

3.1 Architecture and Communication Model

cMix is a new mixnet protocol that provides anonymous communication for its users (senders and receivers). The main goal is to ensure *unlinkability* of messages entering and leaving the system, though it is known which users are active at any given moment.

cMix has n mixnodes that compose a *fixed cascade*: all nodes are organized in a fixed order from the first node to the last. Within the cMix system this order can be systematically changed and rotated, without affecting users in any way. Any message sent by a user is forwarded through all n servers. As with any mixnet, cMix collects a certain number of messages in a batch before processing them. Section 10 discusses our strategy for assembling batches, though details may depend on the application.

To become a cMix user, one must first establish for each mixnode a shared key. Section 4.2 provides more details on how these keys can be established. *Round keys* derived from the shared keys are used in each *round* of communication. A round begins with the start of batch processing.

For each round, β messages are collected and randomly ordered. Each message must have the same length, and all messages in a batch are processed simultaneously. The other messages are not accepted and are sent in a subsequent round. To process messages quickly in real time, cMix performs precomputations that do not involve any user. The precomputations are performed in a separate phase during which cMix executes all public-key encryptions, enabling the real-time computations to be carried using only fast multiplications.

3.2 Adversarial Model

We assume authenticated communication channels between all mixnodes. Thus, an adversary can eavesdrop, forward, and delete messages between mixnodes, but not modify, replay, or inject new ones, without detection. For any communication not among mixnodes, we assume the adversary can eavesdrop, modify, or inject messages at any point of the network.

An adversary can also compromise users; however, we assume that at least two users are honest. Mixnodes can also be compromised, but at least one of them needs to be honest for the system to be secure. We assume compromised mixnodes to be malicious but cautious: they aim not to get caught violating the protocol.

The goal of the adversary is to compromise the anonymity of the communication initiator, or to link inputs and outputs of the system. As with other mixnets [1, 11, 24, 26, 32, 37, 38], we do not consider adversaries whose sole purpose is to launch denial-of-service (DoS) attacks.

We envision a deployment model in which there are dedicated trusted data centers serving as the mixnodes (perhaps competitively awarded). As such, they are incentivized not to be exspelled. By contrast, some mixnets allow the mixnodes to enter and leave with low cost.

An implication of our deployment model is that it is sufficient to be able to detect a cheating node with sufficient probability at some point. By contrast, in more flexible deployment models, the nodes should prove more stringently that they have computed correctly before the output is opened. cMix does require that the exit node commit to its output prior to being able to read the system output, which protects against certain adaptive attacks.

3.3 Security Goals

cMix aims to satisfy each of the following two security properties:

– **Anonymity:** A protocol provides *anonymity* if the adversary cannot map any input message to the corresponding output message, with a probability significantly better than that of random guessing, even if the adversary compromises all but two users and all but one mixnode.

- **Integrity:** A protocol provides *integrity* if at the end of every run involving β honest users:
  1. either, the β messages from the honest users are delivered unaltered to the intended recipients,
  2. or, a malicious mixnode is detected with a non-negligible probability and no honest party is proven malicious.

4 The Core Protocol

We now present the core cMix protocol, beginning with some preliminary notations and concepts, followed by a detailed specification.

4.1 Preliminaries

We introduce the primitives and notations used to describe the protocol. There are n mixnodes that process β messages per batch. For simplicity we assume here that the system already knows for each sender what slot to use out of the β slots. When implementing the system this assignment can, for example, be achieved by including the sender's identity (possibly a pseudonym) when sending a message to the system.

All computations are performed in a prime-order cyclic group $\mathbf{G}$ satisfying the decision Diffie-Hellman (DDH) assumption. The order of the group is p, and g is a generator for this group. Let $\mathbf{G}^*$ be the set of non-identity elements of $\mathbf{G}$.

cMix uses a multi-party group-homomorphic cryptographic system. We make use of a system based on ElGamal, described by Benaloh [4], though any such system could be used. This system works as follows:

- $d_i \in \mathbb{Z}_p^*$: the secret share for mixnode i of the secret key d.
- e: the public key of the system, based on the mixnode shares of the secret key: $e = \prod_i g^{d_i}$.
- $\mathcal{E}_e(m) = (g^x, m \cdot e^x)$, $x \in_R \mathbb{Z}_p^*$: encryption of message m under the system's public key e. We call g^x the *random component* and $m \cdot e^x$ the *message component* of the ciphertext. When applying encryption on a vector of values, each value in the vector is encrypted individually—each with a fresh random value—and the result is a vector of ciphertexts.
- $\mathcal{D}_{d_i}(g^x) = (g^x)^{-d_i}$: the decryption share for mixnode i computed from the random component of a ciphertext using the mixnode's share of the secret key. As with encryption, applying this function on a vector of random values results in a vector of corresponding decryption shares.

To decrypt a ciphertext $(g^x, m \cdot e^x)$, all parties need to cooperate because the decryption shares for all mixnodes are required to retrieve the original message:

$$m \cdot e^x \cdot \prod_{i=1}^n \mathcal{D}_{d_i}(g^x) = m \cdot (\prod_{i=1}^n g^{d_i})^x \cdot \prod_{i=1}^n (g^x)^{-d_i} = m \; .$$

The cMix protocol uses the following values:

- $r_{i,a}, s_{i,a} \in \mathbf{G}^*$: random values (freshly generated for each round) of mixnode i for slot a. Thus, $\mathbf{r}_i = (r_{i,1}, r_{i,2}, \ldots, r_{i,\beta})$ is a vector of random values for the β slots in the message map at mixnode i. Similarly, $\mathbf{s}_i$ is also a vector of random values for mixnode i.
- π_i: a random permutation of the β slots used by mixnode i. The inverse of the permutation is denoted by π_i^{-1}.
- $k_{i,j} \in \mathbf{G}^*$: a group element shared between mixnode i and the sending user for slot j. These values are used as keys to blind messages.
- $M_j \in \mathbf{G}^*$: the message sent by user j. Like other values in the system, these values are group elements. They can be easily converted from, for example, an ASCII-encoded string. The group size determines the length of an individual message that can be sent.

For readability we introduce the following shorthand notations:

- $\mathbf{R}_i$: the product of all local random $\mathbf{r}$ values through mixnode i; i.e., $\mathbf{R}_i = \prod_{j=1}^{i} \mathbf{r}_j$.
- $\mathbf{S}_i$: the product and permutation of all local random s values:
$$\mathbf{S}_i = \begin{cases} \mathbf{s}_1 & i = 1 \\ \pi_i(\mathbf{S}_{i-1}) \times \mathbf{s}_i & 1 < i \leq n. \end{cases}$$
- $\boldsymbol{\Pi}_i(a)$: the permutation performed by cMix through mixnode i, i.e., the composition of all individual permutations:
$$\boldsymbol{\Pi}_i(a) = \begin{cases} \pi_1(a) & i = 1 \\ \pi_i(\boldsymbol{\Pi}_{i-1}(a)) & 1 < i \leq n. \end{cases}$$
- $\mathbf{k}_i$ and $\mathbf{k}_i^{-1}$: the vector of keys shared between mixnode i and the users for all β slots and their inverses, respectively; $\mathbf{k}_i = (k_{i,1}, k_{i,2}, \ldots, k_{i,\beta})$ and $\mathbf{k}_i^{-1} = (k_{i,1}^{-1}, k_{i,2}^{-1}, \ldots, k_{i,\beta}^{-1})$.
- K_j: the product of all shared keys of the sending user for slot j: $K_j = \prod_{i=1}^{n} k_{i,j}$.
- $\mathbf{K}$ is a vector of products of shared keys for the β slots; $\mathbf{K} = (K_1, K_2, \ldots, K_\beta)$.

4.2 Protocol Description

We now present the core protocol. In this explanation we focus on simplicity and clarity; see Section 5 for a discussion of possible security issues and enhancements. We separately discuss each of the three protocol phases: setup, precomputation, and real time.

Setup In the setup phase, the mixnodes establish their secret shares d_i and the shared public key e, which are used for the multi-party homomorphic encryption system.

The users also establish their keys $k_{i,j}$, which they share with all mixnodes. This can be done using any (offline) key distribution method. One way to derive these keys is using a Diffie-Hellman key exchange. The resulting key can be used as a seed to derive unique values for $k_{i,j}$ for every session. Depending on the chosen key distribution protocol, this would be the only time a user is possibly required to perform an asymmetric cryptographic operation. The key exchange must be performed once for each user, and this exchange can be carried during the user's enrollment into the system.

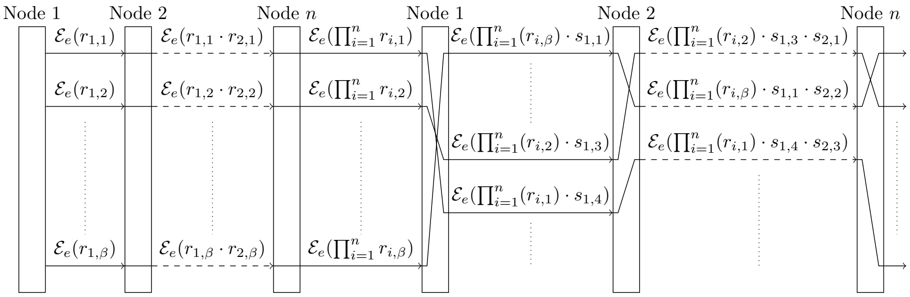
Node 1
Node 2
Node n
Node 1
Node 2
Node n
$\mathcal{E}_e(r_{1,1})$
$\mathcal{E}_e(r_{1,1} \cdot r_{2,1})$
$\mathcal{E}_e(\prod_{i=1}^n r_{i,1})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,\beta}) \cdot s_{1,1})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,2}) \cdot s_{1,3} \cdot s_{2,1})$
$\mathcal{E}_e(r_{1,2})$
$\mathcal{E}_e(r_{1,2} \cdot r_{2,2})$
$\mathcal{E}_e(\prod_{i=1}^n r_{i,2})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,\beta}) \cdot s_{1,1} \cdot s_{2,2})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,2}) \cdot s_{1,3})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,1}) \cdot s_{1,4} \cdot s_{2,3})$
$\mathcal{E}_e(\prod_{i=1}^n (r_{i,1}) \cdot s_{1,4})$
$\mathcal{E}_e(r_{1,\beta})$
$\mathcal{E}_e(r_{1,\beta} \cdot r_{2,\beta})$
$\mathcal{E}_e(\prod_{i=1}^n r_{i,\beta})$

Fig. 1. A schematic example of the first two steps of the precomputation phase, which result in the values $\mathcal{E}_e(\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)$.

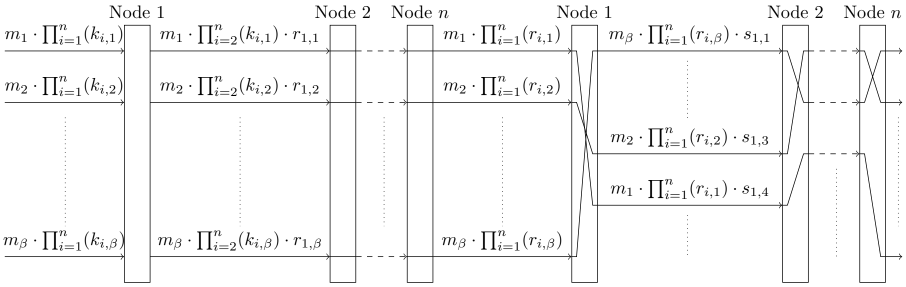
Node 1
Node 2
Node n
Node 1
Node 2
Node n
$m_1 \cdot \prod_{i=1}^n (k_{i,1})$
$m_1 \cdot \prod_{i=2}^n (k_{i,1}) \cdot r_{1,1}$
$m_1 \cdot \prod_{i=1}^n (r_{i,1})$
$m_\beta \cdot \prod_{i=1}^n (r_{i,\beta}) \cdot s_{1,1}$
$m_2 \cdot \prod_{i=1}^n (k_{i,2})$
$m_2 \cdot \prod_{i=2}^n (k_{i,2}) \cdot r_{1,2}$
$m_2 \cdot \prod_{i=1}^n (r_{i,2})$
$m_2 \cdot \prod_{i=1}^n (r_{i,2}) \cdot s_{1,3}$
$m_1 \cdot \prod_{i=1}^n (r_{i,1}) \cdot s_{1,4}$
$m_\beta \cdot \prod_{i=1}^n (k_{i,\beta})$
$m_\beta \cdot \prod_{i=2}^n (k_{i,\beta}) \cdot r_{1,\beta}$
$m_\beta \cdot \prod_{i=1}^n (r_{i,\beta})$

Fig. 2. A schematic example of the first two steps of the real-time phase, which result in the values $\boldsymbol{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n$.

Precomputation The precomputation phase is performed only by the mixnodes, without any involvement from the users. It is performed once for each real-time phase. The mixnodes establish shared values to circumvent the need for public-key operations during the real-time phase. The precomputation phase comprises three different steps given below. The goal of the precomputation phase is to compute the values $\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n$, which are used in the real-time phase. Figure 1 shows a schematic example of the first two steps of the precomputation phase.

Step 1 - Preprocessing. The mixnodes start by generating fresh $\mathbf{r}$, $\mathbf{s}$, $\boldsymbol{\pi}$ values. Then they collectively compute the product of all of their individual $\mathbf{r}$ values under encryption using the public key e of the system, which was computed during the setup phase. This computation takes place by each mixnode i sending the following message to the next mixnode:

$$\mathcal{E}_e(\mathbf{R}_i) = \begin{cases} \mathcal{E}_e(\mathbf{r}_1) & i = 1 \\ \mathcal{E}_e(\mathbf{R}_{i-1}) \times \mathcal{E}_e(\mathbf{r}_i) & 1 < i \leq n. \end{cases}$$

Each mixnode encrypts its own $\mathbf{r}$ values and uses the homomorphic property of the encryption system to compute the multiplication of this ciphertext with the input it receives from the previous mixnode. Eventually, the last mixnode sends the final values $\mathcal{E}_e(\mathbf{R}_n)$ to the first mixnode as input for the next step.

Step 2 - Mixing. In the second step, the mixnodes together mix the values and compute the results $\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n$, under encryption. The mixnodes perform this mixing by having each mixnode i send the following message to the next mixnode:

$$\mathcal{E}_e(\boldsymbol{\Pi}_i(\mathbf{R}_n) \times \mathbf{S}_i) = \begin{cases} \pi_1(\mathcal{E}_e(\mathbf{R}_n)) \times \mathcal{E}_e(\mathbf{s}_1) & i = 1 \\ \pi_i(\mathcal{E}_e(\boldsymbol{\Pi}_{i-1}(\mathbf{R}_n) \times \mathbf{S}_{i-1})) \times \mathcal{E}_e(\mathbf{s}_i) & 1 < i \leq n. \end{cases}$$

As with the first step, the last mixnode sends the final encrypted values $\mathcal{E}_e(\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)$ to the first mixnode. These final values now must be decrypted together by all mixnodes, which happens in the last step of the precomputation.

Step 3 - Postprocessing. To complete the precomputation, the mixnodes decrypt the precomputed values. Each mixnode i computes its decryption shares $\mathcal{D}_{d_i}(\mathbf{x})$, where $(\mathbf{x}, \mathbf{c}) = \mathcal{E}_e(\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n))$. The message parts **c** are multiplied with all the decryption shares to retrieve the plaintext values $\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n$. This computation can be carried out either using another pass through the network (in which every mixnode multiplies in its own decryption share), or by having all mixnodes send their encryption shares to the last mixnode, which can then perform the multiplication. The last mixnode to be used in the real-time phase stores the decrypted precomputed values.

Real Time For the real-time phase, each user constructs its input by taking its message m and multiplying it with its combined shared key k to compute the blinded message $m \times k$. This blinded message is then sent to the mixnet. One option would be to send the blinded messages to the first mixnode. Once the first mixnode receives enough blinded messages, it combines those messages to yield the vector $\mathbf{M} \times \mathbf{K}$. As in the precomputation phase, the real-time phase can again be split into three steps. Figure 2 gives a schematic example of the first two steps.

Step 1 - Preprocessing. During the preprocessing step, the mixnodes take out the keys **k** they share with the users and add in their **r** values to blind the original messages. This computation is performed by each mixnode i sending the following to the next mixnode:

$$\mathbf{M} \times \mathbf{K} \times (\textstyle\prod_{j=1}^{i} \mathbf{k}_j^{-1} \times \mathbf{r}_j) = \mathbf{M} \times \mathbf{K} \times (\prod_{j=1}^{i-1} \mathbf{k}_j^{-1} \times \mathbf{r}_j) \times \mathbf{k}_i^{-1} \times \mathbf{r}_i \ .$$

The last mixnode sends the final values $\mathbf{M} \times \mathbf{R}_n = \mathbf{M} \times \mathbf{K} \times \prod_{j=1}^{n} \left(\mathbf{k}_i^{-1} \times \mathbf{r}_i\right)$, which are the blinded versions of the original messages, to the first mixnode as input for the next step. Now the user-specific keys **k** are taken out and replaced by the user-independent values **r**.

Step 2 - Mixing. The second step performs the mixing to hide the association between sender and receiver. The **s** values are added in to hide which input message corresponds to which output message. Each mixnode i (except the last mixnode) sends the following message to the next mixnode:

$$\boldsymbol{\Pi}_i(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_i = \begin{cases} \pi_1(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{s}_1 & i = 1 \\ \pi_i(\boldsymbol{\Pi}_{i-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{i-1}) \times \mathbf{s}_i & 1 < i < n. \end{cases}$$

Finally, the last mixnode computes:

$$\boldsymbol{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n = \pi_n(\boldsymbol{\Pi}_{n-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{n-1}) \times \mathbf{s}_n \ .$$

Now every mixnode has performed its mixing, destroying the associations between senders and receivers.

Step 3 - Postprocessing. The last mixnode can perform the final step. This mixnode retrieves the locally stored precomputed values $\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n$. To retrieve the permuted messages it now needs only to perform the following computation, using the result from the previous mixing step:

$$\boldsymbol{\Pi}_n(\mathbf{M}) = \boldsymbol{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times (\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1} \ .$$

How the messages are then delivered to the recipients depends on the application and is independent from cMix. This step concludes the real-time phase, in which no public-key operations are performed.

5 Protocol Integrity

The cryptographic construction presented in Section 4 expects the mixnodes and users to be honest but curious. As for most mixnet protocols [11,27,34,37], cMix requires additional measures to ensure that mixnodes cannot tamper with the messages nor with the flow without detection. In this section, we augment the protocol to protect its anonymity and integrity against malicious attacks by users and by compromised mixnodes, which we shall call "adversarial mixnodes." The overall strategy relies on the assumption that compromised mixnodes are malicious but cautious.

5.1 Integrity of Values and Messages

To enable honest mixnodes to verifiably detect any malicious mixnodes that employ incorrect values or permutations in the precomputation or real-time phase, cMix augments communications with proofs. To this end, all messages exchanged between mixnodes are signed using a digital signature system with existential unforgeability under an adaptive chosen-message attack [23]. In addition, during precomputation, each mixnode commits to the permutation π_j it applies to the incoming slot j using a perfectly hiding commitment (Commit) scheme [25] and signs that commitment. Each node broadcasts its signed commitment using a reliable broadcast (Broadcast) protocol [18] [46]. Doing so makes it possible to reconstruct and verify all individual values $r_{i,j}$, $s_{i,j}$ and $\pi_{i,j}$ that mixnode j applies to slot i.

In the real-time phase, the users become involved, making the process more complicated because they do not perform any public-key operations during the real-time phase. The values that we need to verify that depend on the users are the $k_{i,j}$ values (shared between a mixnode i and the user in slot j), and the $m_j \cdot K_j$ values (the blinded message that a user sends in slot j).

Because the k values are shared between a mixnode and a user, we need them to agree on the commitments to these values. If we detect an anomaly involving the k values, and a mixnode and user disagree on the value used, the commitment needs to provide proof of who misbehaved.

For this task, the following procedure can be followed in case the k values are derived from a seed that is agreed upon by a mixnode and user during the setup phase. After establishing this seed, the mixnode will compute a commitment of the seed and provide this commitment to the user. The mixnode generates the commitment in a way that enables the user to reveal the commitment and prove to other parties that the mixnode generated the commitment. The user must verify the commitment during the setup phase. This verification requires an additional public-key operation, though it will be performed only once provided the protocol runs without any disturbance.

Message Integrity at Entry Messages at entry to cMix need to arrive at cMix and pass through the non-permuted part of the protocol without any undetected modification. Providing integrity of messages at this point is a challenge if one wishes to keep clients free from using any asymmetric cryptography during the real-time phase. We propose a construction where message authentication codes (MACs) are generated over the input message to the cMix system. A shared-key MAC is sufficient here because the communicants (sender, mixnode) do not need to convince any third party.

To accomplish this goal, we introduce additional key values $l_{i,j}$, shared between mixnode i and the user for slot j, established and committed to in a similar way as for the k values. When the user sends the blinded message $m_j \cdot K_j$ to the system, the user also sends the following messages:

$$h_j = \text{Hash}(m_j \cdot K_j) \text{ and } (MAC_{l_{1,j}}(h_j), \ldots, MAC_{l_{n,j}}(h_j)) .$$

During the real-time phase, the first mixnode starts by checking its corresponding MAC value in the list. If it is incorrect, the mixnode informs the other mixnodes and does not forward the computed value for this slot. If the MAC value is correct, it forwards the h values and the MAC values for the other mixnodes, together with its basic computed values.

Each subsequent mixnode follows the same procedure. At the end of the first step of the real-time phase, all mixnodes have checked their MAC values on the received h values, or the value is not processed any further.

5.2 Message Tagging Detection

A message-tagging attack is an attack where the adversary can mark a message at some point during the process, such that it is recognizable when it is output, compromising unlinkability between the inputs and outputs [41]. To perform a tagging attack unnoticed, the tag should also be removed before the messages are output. Tagging attacks are a threat to all mixnets that use some form of malleable encryption, such as homomorphic encryption or group multiplications, where valid messages can be recognized when output by the mixnet. For example, Pfitzmann [40] presents such an attack on re-encryption mixnets.

A simple example of a tagging attack is the following: The last mixnode multiplies the blinded message in one of the slots j with an additional factor t in the first step of the real-time phase. Now the blinded message in slot j will be $m_j \cdot \prod_{i=1}^{n} r_{i,j} \cdot t$ at the end of the first step, whereas the values in the other slots stay the same. When the last mixnode performs Step 3, it will see the final messages before it outputs them. If the messages are recognizable as valid outputs, it will observe that one of the messages does not seem to be valid. If this invalid message becomes valid when multiplied with t^{-1}, this message is likely the tagged one. The mixnode can now link the message, and possibly the recipient, to the sender. The mixnode can remove the tag and output the original messages, making it unobservable to the users and other mixnodes that a tagging attack took place.

Detection. To protect against these kinds of attacks and make them detectable, only small changes are needed to the protocol:

- **Precomputation Phase - Step 3:** The mixnodes no longer send out their decryption shares to retrieve the precomputed values. Instead, they keep their shares secret and publish only a commitment to them. The last mixnode also publishes a commitment to the message component of the ciphertext. The commitments can be computed, for example, using only one signature per mixnode for the decryption shares for all slots simultaneously. The plaintext results of the precomputation phase are thus no longer retrieved in that phase itself.
- **Real-Time Phase - Step 3:** The output of the mixing step $\boldsymbol{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n$ is published by the last mixnode. Afterwards, all mixnodes release their decryption shares $\mathcal{D}_{d_i}(\mathbf{x})$ and the message component of the ciphertext $\mathbf{c}$. The output messages are then computed as follows, where $(\mathbf{x}, \mathbf{c}) = \mathcal{E}_e(\boldsymbol{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)$:
$$\boldsymbol{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \mathbf{c} \times \textstyle\prod_{i=1}^{n} \mathcal{D}_{d_i}(\mathbf{x}) \ .$$

Because the mixnodes committed to all of the values necessary to retrieve the precomputed value, they cannot change these values to take out a possible tag anymore. The output of the mixing step does not reveal anything yet about the messages, because the r and s values are still included. The precomputed value to take out these values is retrieved only after the output of the mixing step is made public. Therefore, the last mixnode would not know from which output slot a possible tag should be removed before the output of the mixing step is made public. Once the output is made public, the tag cannot be removed because all computations for the third step are fixed and can be verified by anyone.

5.3 Sending and Verifying Trap Messages

To ensure the protocol is functioning correctly, we let users send trap messages with a certain probability, and request to open message paths of these traps. Opening of a path includes verification of all messages exchanged between mixnodes, the incoming message from the user, as well as intermediate values and permutations.

Sending Trap Messages and Requesting to Open Them: To send a trap, a user starts by forming a string x with a round ID, user ID, and a statement that this is the dummy message. She calculates a MAC of x with every key $l_{i,j}$ shared between mixnode i and the user for slot j. The trap message is then the string together with its MAC values, i.e., $m = (x, MAC_{l_{1,j}}(x), \ldots, MAC_{l_{n,j}}(x))$. Note that these MAC values are different from the MAC values mentioned in Section 5.1, even though we are using the same keys in both places.

After the round is over, the trap message appears at the output. Each mixnode verifies the correctness of her associated user ID, round ID, and $MAC_{l_{i,j}}$ values. If any mixnode i detects that any of these values is incorrect, she stops the round. Otherwise, the mixnode sends an authenticated message to user j notifying that the dummy message is received correctly, including an incremented counter of all dummies received from this user.

If a user actually sent a dummy message, but she does not receive a notification message from each mixnode, she initiates the verification procedure. The verification procedure consists of the following steps:

i) User j sends a request, tagged with MAC values, to open the path with her dummy message to all mixnodes.
ii) A mixnode can either agree to participate in the path opening, or can dispute the MAC of the path-opening request.
iii) If mixnode i disputes the MAC intended for her, both the user and mixnode open their local $l_{i,j}$ values to validate the MAC.
iv) If both the user and the mixnode are using the same $l_{i,j}$ value, then the $MAC_{l_{i,j}}()$ values computed by them should be same, and the correctness of the request sent by the user can be easily validated. If the user has sent an invalid request, the request is dropped.
v) If the user and the mixnode dispute over the $l_{i,j}$ value, then the seed of $l_{i,j}$ and the commitment to that seed (made by the mixnode and verified by the user during setup) are opened. The user and mixnode each recomputes $l_{i,j}$ from the seed. Two possibilities follow:
    a) If the recomputed $l_{i,j}$ value is not equal to what the user claims, the path-opening request is dropped.
    b) If the recomputed $l_{i,j}$ value is not equal to what the mixnode claims, she must either agree to participate in the path-opening, or be considered malicious.
vi) Once all the mixnodes agree to participate in the path-opening, they first open the path of the precomputation phase, and then the real-time phase (both are explained below).

Path Opening for the Precomputation Phase: For the input slot j that we want to verify, the mixnodes have to decrypt exchanged messages for this slot and compute the r values in the non-permuted part. For the permuted part, mixnodes subsequently reveal the corresponding permutations and decrypt exchanged messages to obtain the s values. Doing so we can follow the computation through the mixnet and verify whether the precomputed value that was output was correctly computed.

To check, for example, whether mixnode i performed its computation correctly, that mixnode needs to present the signature from the previous mixnode on the values it received. The next mixnode will also have to present the signature it received from mixnode i to obtain proof what values mixnode i output. Once all information about the input, output, and values used in the expected computation are known, due to the signatures, commitments, and threshold decryption, it can be verified whether mixnode i performed the computation as expected or not.

Path Opening for the Real-Time Phase: Malicious mixnodes can also employ incorrect messages, values, and permutations in the real-time phase. We wish to make the mixnodes accountable for their behavior. One challenge is that malicious users may try to victimize some honest mixnodes by providing inconsistent inputs and later deny having done so.

First, the mixing step is verified. This check is performed in a similar fashion as for the precomputation phase. For the preprocessing step, a comparable approach is followed. To verify whether the correct input from the user to the system is used, all the keys l for the MAC values for the suspicious slot are output. The purpose is to detect if a mixnode or user changed the input. Because the values were processed in the mixing step, during the first step of the real-time phase, all mixnodes accepted the MAC values and thus should be able to provide a correct key. The mixnodes also reveal their r and k values for the corresponding slot, and the mixnodes check whether they performed their computations correctly. Although the mixnodes committed to the r values, they have not committed to individual k values. Therefore the sender must be involved in this process. The sender will also release all the k values it used for this message.

If a k value released by the sender does not match the one released by the corresponding mixnode, either the mixnode or the sender is misbehaving. The sender might do this to blame a mixnode of misbehaving to cause it to be removed from the system. This dispute needs to be resolved by having the user reveal the commitment by the mixnode on the shared keys. Doing so might also reveal k values used in previous sessions, but these values are only one of the components of K and therefore do not leak the original messages. There are two possibilities: either the user was malicious, in which case we do not care about his or her previous messages, or the mixnode was malicious, in which case we consider the k values to be compromised already. This procedure will reveal who acted maliciously and modified the output message.

6 Integrity Analysis

We define the integrity property as following:

Definition 1. *A protocol maintains* integrity *if, at the end of each run involving β honest users:*

1. *either, all the β messages are delivered unaltered to the intended recipients.*
2. *or, a malicious mixnode is detected with a non-negligible probability, and no honest party is proven malicious.*

If all the parties involved in the cMix protocol behave correctly, the messages are delivered unaltered to the intended recipients. We shall argue informally that, if any of the messages are altered by any node, (a) no honest party can be proven malicious, and (b) at least one malicious mixnode is detected with non-negligible probability.

No honest party is proven malicious. An honest mixnode can be targeted by some malicious colluding users or mixnodes that provide inconsistent inputs. If the output is corrupted by $n-1$ colluding mixnodes that provide incorrect decryption shares, then validating the commitments will reveal this fact. Because the commitment scheme is secure, malicious mixnodes cannot publish incorrect commitments without detection.

If the previous mixnode has provided inconsistent input, then the current mixnode can prove its correctness by providing the input signed by the previous mixnode, since all messages exchanged between mixnodes are digitally signed. If a mixnode falsely claims that the previous mixnode has provided incorrect input, it will have to provide the input signed by the previous node as proof. In both cases, the adversary will be successful only if the digital signature scheme is not secure.

If a user has provided inconsistent input and claimed falsely that the mixnode is malicious, then the mixnode can prove its innocence using the MAC value received from the mixnode. If the user and the mixnodes show disagreement about the key used for computing the MAC value, the seed for generating the key is revealed. In the setup phase, the mixnode generates a commitment to the seed, and the user verifies the commitment. Thus, the user can successfully claim a different seed for that commitment only when the commitment scheme is not secure.

We need to note here that user j_1 can not modify the message for slot j_2, only a mixnode can do that. So, a user can not modify the message of other users to target an honest mixnode.

At least one malicious mixnode is detected with non-negligible probability. We assume our adversary is malicious but cautious. So, if there is even a slight chance of a malicious node being detected and thrown out of the system, it will refrain from engaging in any such mischief. We now show that if an adversary behaves maliciously, there is a significant probability that it will be caught and will lose one mixnode.

We need to note here that if a message is modified, it is modified by one or more of the mixnodes, a user can not modify messages sent by other users.

Let p denote the probability with which a user will send a trap message. In each round there are a total of β messages. The expected number of trap messages in a batch is $\alpha = p\beta$.

We want to calculate the probability that a mixnode will be detected even if it modifies only one path. When paths are opened, the probability that the corrupted path will be chosen among one of the α trap paths is $\binom{\beta-1}{\alpha-1}/\binom{\beta}{\alpha} = \alpha/\beta = p$.

Thus, in case of malicious behavior, at least one of the adversarial nodes will be caught with probability p. If $p = 1/2$, we can use 2β slots to ensure an anonymity set of β, while having a significant probability to detect a malicious node.

7 Anonymity Analysis

We analyze the anonymity of the cMix protocol. First, we state a precise definition for a desired anonymity property. Second, we sketch a proof that cMix satisfies this definition. Third, we explain why cMix resists well-known attacks on mixnets.

7.1 Formal Anonymity Analysis

Motivated by [9, 48], we define our desired anonymity property in the form of a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, where the adversary $\mathcal{A}$ is a probabilistic polynomial time (PPT) Turing machine. Upon request, $\mathcal{C}$ runs the protocol and returns the outputs to $\mathcal{A}$. For ease of exposition, we fix the number of users to β and assume each user sends a message during every round. Our static adversary $\mathcal{A}$ is allowed to compromise all but two users, as well as all but one mixnode before execution begins. The challenger $\mathcal{C}$ executes the protocol on behalf of honest nodes and users, while $\mathcal{A}$ acts on behalf of the compromised users and nodes. The adversary can eavesdrop on all messages among the mixnodes, but it cannot modify them; thus, $\mathcal{C}$ also sends a copy of every communication between two honest parties along with its source and destination to $\mathcal{A}$. We denote the output of the adversary from this game as $\langle \mathcal{A}|\mathcal{C} \rangle$.

The anonymity game works as follows:

Setup phase: The challenger $\mathcal{C}$ runs the setup for all honest nodes, and provides all public information to the adversary $\mathcal{A}$.

Query phase: As many times as $\mathcal{A}$ requests, $\mathcal{C}$ takes input messages in the form of (sender, message)-pairs for all the β slots from $\mathcal{A}$, and runs the cMix protocol with those inputs.

Challenge phase: $\mathcal{A}$ chooses two honest users S_0 and S_1 and two distinct messages m'_0 and m'_1. $\mathcal{A}$ also chooses messages for all other honest users as m, where $m \notin \{m'_0, m'_1\}$, and sends this challenge to $\mathcal{C}$. The challenger $\mathcal{C}$ tosses a uniform random coin to obtain a bit b, and assigns $m_0 = m'_b$ and $m_1 = m'_{1-b}$, the messages corresponding to users S_0 and S_1, respectively. The challenger $\mathcal{C}$ then runs the protocol with these input messages, and gives the output message set to the adversary.

Query phase: After the challenge run, as many times as $\mathcal{A}$ requests, $\mathcal{C}$ takes message inputs from $\mathcal{A}$ and runs the cMix protocol again.

Output phase: Finally, the adversary outputs $\bar{b}$ as its guess for b.

The adversary's advantage in the anonymity game is :

$$Pr[0 = \langle A|C \rangle \mid b = 0] - Pr[1 = \langle A|C \rangle \mid b = 0].$$

We now define the anonymity property as follows:

Definition 2. *A protocol maintains* anonymity *if the advantage of the adversary in the anonymity game is negligible.*

Theorem 1 states that the cMix protocol satisfies Definition 2.

Theorem 1. *If $\mathcal{E}$ is a CPA-secure group-homomorphic encryption scheme, and* Commit *is a perfectly-hiding non-interactive commitment scheme, and if the protocol maintains integrity, then cMix offers anonymity as defined in Definition 2 in the random oracle model.*

Proof (Sketch). We prove the security of cMix by reduction from the security of the encryption system $\mathcal{E}$. In our argument, we use an equivalent modified version of the standard encryption game used to define the security of $\mathcal{E}$: on a given challenge message pair (m_0, m_1), the challenger $\mathcal{C}_\mathcal{E}$ of the encryption system returns a ciphertext pair $(\mathcal{E}_{pk}(m_b), \mathcal{E}_{pk}(m_{1-b}))$ instead of just one ciphertext $\mathcal{E}_{pk}(m_b)$.[5]

Without loss of generality, we assume that $\mathcal{A}$ can compromise $\beta - 2$ users and $n - 1$ nodes. Let S_0 and S_1 be any two honest users and let i be the only honest mixnode in the system.

During the setup phase, the challenger $\mathcal{C}$ initiates the CPA game with the encryption challenger $\mathcal{C}_\mathcal{E}$, and ensures that the encyption public key in cMix is the same as the

[5] By defining an appropriate hybrid in the simulation, this modified game can be easily shown equivalent to the standard CPA-security encryption game.

public key in the CPA-security game with $\mathcal{C}_{\mathcal{E}}$. In the query phase, $\mathcal{C}$ can easily simulate the honest nodes, including the decryption shares for those nodes. It can do so because it manages and checks the hashing (random) oracle queries made by the adversary for its commitments to messages as well as permutations, and since it can open the committed decryption shares (obtained using the perfectly-hiding commitment scheme) with its own choice of values.

In the challenge phase, $\mathcal{C}$ assigns the corresponding messages $m_0 = m'_{\hat{b}}$ and $m_1 = m'_{1-\hat{b}}$, as determined by a random bit $\hat{b}$. In addition, it chooses s values (say $s_{i,0}$ and $s_{i,1}$) for the slots of S_0 and S_1 at honest node i such that $m_0/m_1 = s_{i,0}/s_{i,1}$. The challenger $\mathcal{C}$ then runs the challenge phase with the CPA-security challenger $\mathcal{C}_{\mathcal{E}}$ with $s_{i,0}$ and $s_{i,1}$ as two challenge messages. $\mathcal{C}_{\mathcal{E}}$ returns with a ciphertext pair $\mathcal{E}_{pk}(s_{i,b})$ and $\mathcal{E}_{pk}(s_{i,1-b})$. The anonymity-game challenger $\mathcal{C}$ uses this response pair in the precomputation phase for node i and slots S_0 and S_1. This way the challenger $\mathcal{C}$ embeds its decryption challenge from the CPA-security game into an unknown permutation for an honest player in the anonymity game of cMix.

In the real-time phase of cMix, the challenger $\mathcal{C}$ runs the protocol in a standard manner, except at the mixing step for node i. Here, we use $s_{i,1}$ and $s_{i,0}$ for slots S_0 and S_1, resepctively, such that $m_0 s_{i,1} = m_1 s_{i,0}$. Doing so allows $\mathcal{C}$ to open any of two honest real-time messages to any of the m_0 and m_1 values.

Finally, during the postprocessing step of the real-time phase, the challenger $\mathcal{C}$ knows all permutations except for two slots associated with m_0 and m_1 (i.e., except for $m'_{\hat{b}}$ and $m'_{1-\hat{b}}$). As a result, it can open the decryption shares appropriately for all other messages. Since $m_0 s_{i,1} = m_1 s_{i,0}$, the challenger can open m_0 and m_1 in any order.

In the end, if the anonymity game adversary A predicts the bit $\hat{b}$ correctly, the challenger $\mathcal{C}$ knows that its choice of $s_{i,1}$ and $s_{i,0}$ for slots S_0 and S_1 in the real-time phase matched the challenge ciphertexts used in the precomputation phase. In this case, it outputs $b = 1$ to the encryption game challenger $\mathcal{C}_{\mathcal{E}}$. Otherwise, if the adversary guesses $\hat{b}$ incorrectly, the challenger $\mathcal{C}$ outputs 0 to $\mathcal{C}_{\mathcal{E}}$.

As a result, if the cMix protocol does not provide anonymity, we can apply it to break the CPA-security of the underlying group-homomorphic encryption system, a contradiction.

7.2 cMix Resists Standard Mixnet Attacks

We explain how cMix resists standard attacks on mixnets. There are several active attacks against fixed-cascade mixnets. The literature can be confusing because there is no common nomenclature for these attacks, so we define each attack type we discuss.

Many of these attacks are based on adding, deleting, or modifying messages in the mixnet at the entrance. An adversary can either block incoming messages from $\beta - 1$ users, or replace them from the batch with her own, allowing her to trace a target message. Such attacks are called $n - 1$ or *flooding* attacks [11].

Replay attacks [11] [7] work by retransmitting a message from a previous session. These attacks can work only if encryption or re-encryption is deterministic. A generalization of these attacks is called a *blending* attack [17] and happens when the adversary manipulates messages that are in the same batch that includes the target messages. Dingeldine et al. [17] discuss a number of countermeasures. Section 5.2 addresses tagging attacks and our protections against them. Because the **K**, **R**, and **S** values are never reused, cMix protects against *replay attacks* (see Section 5).

Contextual attacks [41], sometimes referred as *traffic-confirmation attacks* [42], *communication pattern attacks* [41], and intersection attacks [6], analyze the time when particular senders and receivers participate in the protocol, their communication patterns, and how many messages they send and receive. Only unobservable [39] systems protect against this type of attack. These attacks are sometimes included in the following category of attacks.

Intersection attacks and *statistical disclosure attacks* [7, 14, 15] make use of mixnet topologies that allow users to choose routes freely for their messages (*free mix routes*). In such systems, sets of messages in a batch of a mixnode can be distinguished, for example, since they come from different mixnodes or have different route lengths. Assuming that users often use the same routes for their messages, these routes can be distinguished by analyzing network flow data. Because cMix uses a fixed cascade of mixnodes [7], cMix is not susceptible to this family of attacks.

Traffic-analysis attacks are targeted at connection-based anonymity systems, as opposed to message-based systems. These connection-based systems often do not batch and permute incoming packets, and they use free mix routes. This approach permits an adversary to distinguish these paths based on measures such as counting packets [45] and timing communications [13]. These attacks do not work on cMix because cMix permutes messages in batches using a fixed cascade of nodes.

Galteland, Mjølsnes, and Olimid [20] propose a tagging attack and an insider attack against the cMix protocol, as described in the preliminary cMix eprint. But security mechanisms specified in this preliminary cMix eprint prevent both attacks, as do alternative integrity mechanisms (e.g., trap messages) specified in the current cMix paper. In particular, as presented in the preliminary cMix eprint, cryptographic commitments enforced by Random Partial Checking (RPC) [31] prevent both attacks. Thus, these purported "Norwegian Attacks" do not work.

8 Comparison with Other Mixnets

We compare cMix with well-known fixed-cascade mixnet approaches based on performace. Specifically, as summarized in Table 1, we compare the performance of the core cMix protocol with that of each of the following three competing approaches: original mixnet and hybrid mixnets, re-encryption mixnet, and re-encryption mixnet with precomputation.

For each approach, we compare the precomputation and real-time costs, further compared by number of single-party public-key operations, multi-party public-key operations, and multiplications (each by client and by mixnodes). Note that, when using ElGamal encryption, the multiplication of two ciphertexts requires two multiplications.

The original mixnet [11] requires each sender to perform n encryptions; each of the n mixnodes in the cascade performs one decryption per message and β decryptions per batch. In total, all mixnodes perform n decryptions per message and βn decryptions per

	Precomputation (ops for all mixnodes)			Real time (ops per client; ops for all mixnodes)		
	PK ops	Multi-party PK ops	Mult	PK ops	Multi-party PK ops	Mult
cMix (core)	$2n\beta$	β	$4n - 2\beta$	0; 0	0; 0	n; $3n\beta$
Original mix	-	-	-	n; $n\beta$	0; 0	0; 0
Re-encryption mix	-	-	-	1; $n\beta$	0; β	0; $2n\beta$
Re-encryption mix (precomputation)	$n\beta$	0	0	1; 0	0; β	0; $2n\beta$

Table 1. Performance comparison of the core cMix protocol with three competing mixnet [11,37] approaches: number of multiplication (Mult) and public-key (PK) operations performed for a batch of β messages processed by an n-node mixnet. One multi-party public-key operation (ops) requires all nodes to participate.

batch. Hybrid mixnets [27,34] require the same amount of asymmetric encryptions, but on a smaller plaintext.

In re-encryption mixnets [37], each client performs one encryption of its message using the mixnet's shared public key. Each node re-randomizes every message, instead of decrypting each one as with original mixnets, resulting in one public-key operation and one multiplication of ciphertexts per message per node. In addition, the nodes need to decrypt the output of the mixnet in a multi-party computation.

Re-encryption mixnets can be improved in a straightforward way using precomputation to perform the public-key operations necessary for the re-randomization, similarly to cMix's strategy. As shown in Table 1, however, with regard to real-time computation, cMix outperforms re-encryption mixnets with precomputation.

Universal re-encryption mixnets perform much more slowly because they require senders to encrypt messages with public keys of their recipients.

9 Proof of Concept

We implemented a proof-of-concept protoype in Python, including tagging detection as discussed in Sections 5.2. Towards reducing the communication latency, we have also introduced a network handler.

Introducing an untrusted *network handler* reduces latency. In Steps 1 and 3 of the precomputation and real-time phases, only products of values known by the individual nodes are computed (see Section 4.2). To compute these products, it is not necessary to make a full pass through the mixnet. Instead, each node can send its values to an untrusted third party, which we call the network handler, who can compute the products and return the results to the mixnet. Doing so reduces latency of the network significantly because each node can send its values simultaneously to the handler, instead of forwarding its local result to the next node sequentially. The network handler does not learn any secret value, and it computes only values that would anyway become public. The network handler, and each of the mixnodes, is a single point of failure. In the event of failure, however, because the handler performs only public operations, it can be easily replaced by another entity—for example, by one of the mixnodes.

Each mixnode includes a keyserver (to establish shared keys with the users) and a mixnet server (to carry out the precomputations and real-time computations). We use Ed25519 [5] signatures to implement authenticated channels between mixnodes. For the precomputation, each mixnode uses parallel processes for the computation of the encryptions and the decryption shares. In the real-time phase, all operations are performed in a single thread.

We performed experiments by running the protoype on Amazon Web Services (AWS) instances, with each node comprising a c3.large with two virtual processors and 3.75 GB of RAM. For all values, we used a prime-order group of 2048 bits.

On the AWS instances, each 2048-bit ElGamal encryption took approximately 10 milliseconds on average, and the computation of a decryption share took approximately 5 milliseconds. Multiplication of group elements took only a fraction of a millisecond.

For our experiments we performed 100 precomputation and real-time phases for selected batch sizes up to 1000 with five mixnodes. Table 2 gives observed timings on the network handler for selected batch sizes using five mixnodes, without any enhanced security mechanisms mentioned in Section 5. We measured elapsed time on the network handler from the time it instructed the nodes to start until either the precomputation finished successfully, or until it computed the final responses to be sent to the users in the real-time phase. Table 3 gives timings for the real-time phase per node and for the network handler, in both CPU and wall clock time. These timings show the low computational load on the nodes during this phase.

These timings demonstrate the high performance of the system in the real-time phase. The precomputation can be easily accelerated by performing more computations in parallel. Additional processors would significantly improve the time it takes

Batch	Precomputation		Real time	
size	Mean	Std. dev.	Mean	Std. dev.
10	0.48	0.07	0.07	0.02
50	1.99	0.04	0.21	0.04
100	4.00	0.30	0.38	0.06
200	7.74	0.09	0.75	0.08
300	11.46	0.13	1.09	0.08
400	15.24	0.11	1.44	0.08
500	19.08	0.23	1.80	0.11
1000	37.94	0.19	3.58	0.12

Table 2. Timings measured on the network handler from the start of the phases until the final values or responses are computed. Timings are in seconds (wall clock) for 100 runs of the precomputation and real-time phases, for various batch sizes using five mixnodes.

Batch	Mixnode		Network handler	
size	CPU	Wall	CPU	Wall
10	0.01	0.04	0.01	0.07
50	0.04	0.16	0.02	0.21
100	0.07	0.30	0.02	0.38
200	0.14	0.60	0.04	0.75
300	0.22	0.87	0.06	1.09
400	0.29	1.15	0.07	1.44
500	0.36	1.45	0.09	1.80
1000	0.73	2.88	0.19	3.58

Table 3. Mean timings in seconds (CPU and wall clock) for 100 runs of the real-time phase of the cMix protocol measured on the mixnodes and network handler, for various batch sizes using five mixnodes. For the mixnodes the mean time is taken over all mixnodes.

to compute all necessary encryptions and decryption shares. For the real-time phase, a network connection with low latency would improve the timings.

10 Extensions, Discussion, and Future Work

Here, we first describe how receivers can send immediate responses. We then briefly discuss how to arrange messages into batches and how to deal with node failures. We also outline some of our future plans.

Return Path. It is easy to extend cMix to enable a receiver to send an immediate response through the mixnet, for example, to acknowledge receiving a message. To accomplish this goal, the nodes generate additional random values $\mathbf{s}'$ and compute the permuted products $\mathbf{S}'$ during the precomputation phase. The nodes and users also generate fresh keys $\mathbf{k}'$, which will be used to encrypt the return message.

For a return path, the mixnodes apply the inverse permutations π^{-1} so that the responses will arrive at the original senders. Unless the recipient who sends a response shares keys with the system, no fresh $\mathbf{r}'$ values are needed because the message would not enter the system blinded and hence Step 1 of the real-time phase could be skipped. In Step 2, the system applies the inverse permutations π^{-1} and fresh $\mathbf{s}'$ values. In Step 3, to encrypt the response to the original sender, instead of multiplying only with its decryption component, each node multiplies with the product of its decryption component and $\mathbf{k}'$ value.

Batch Strategy. cMix follows the “threshold and timed mixing strategy” from Serjantov et al. [44], where a new round is started every t seconds only if there are at least β messages in the buffer. We expect at least β users to be active at any given time. When a smaller number of users is active, this strategy can lead to increased latency or even disruption. At the cost of increased energy consumption, one design choice is to inject dummy messages when needed to ensure enough traffic to have β messages every t seconds. Alternatively, empty slots can be used to verify if the precomputation was performed correctly, by revealing all committed values for these slots, where empty slots to use should be chosen at random.

Node Failure. Because cMix uses a fixed cascade of nodes, it is important to consider what happens if a node fails. First, we consider a node failure to be a highly rare event because we expect each node to be a highly reliable computing service that is capable of seamlessly handling failures. Second, the system will detect node failure and notify the senders and the other nodes; senders will be instructed to resend using a new cascade (e.g., the old cascade without the failed node). Each node can detect failures by listening for periodic “pings” from the other nodes.

To minimize possible disruption caused by a single failure, at the cost of increasing the precomputations, the following option can be deployed: Each node can have a reserve of precomputations ready to use for certain alternative cascades. For example, this reserve can include each of the alternative cascades formed by removing any one node from the current cascade.

Future Steps. Tasks we plan to work on in the future include the following: First, we would like to deploy cMix, including implementing and refining different applications.

We would also like to carry out more performance studies. Second, we plan to explore alternative and even more efficient approaches for enforcing integrity of the nodes, to ensure that they cannot modify any message without detection. Third, currently, message length is restricted by the group modulus. We have begun to work out how to apply key-homomorphic pseudorandom functions [8] and an appropriate additive homomorphic encryption system to allow any length message. Fourth, we would like to explore possible ways of reusing a precomputation in a secure way.

11 Conclusion

cMix offers a promising new approach to anonymous communications, building on the strong anonimity properties of mixnets, and improving real-time cryptographic performance by eliminating real-time public-key operations in its core protocol. By replacing real-time public-key operations with precomputations, and by avoiding the user's direct involvement with the construction of the path through the mixnodes, cMix scales well for deployment with large anonymity sets and large numbers of mixnodes. Even though the adversary may know all senders and receivers in each batch, she cannot link any sender and receiver unless all mixnodes are compromised. cMix offers the potential for real-time performance improvements over existing mixnets, without losing any of their security guarantees.

Acknowledgments

We thank the anonymous reviewers for their comments. We also thank the following people for helpful suggestions: David Delatte, Russell Fink, Bryan Ford, Moritz Neikes, and Dhananjay Phatak.

Sherman was supported in part by the National Science Foundation under SFS grant 1241576 and a subcontract of INSuRE grant 1344369, and by the Department of Defense under CAE-R grant H98230-15-10294. Krasnova conducted this research within the Privacy and Identity Lab (PI.lab, http://www.pilab.nl) funded by SIDN.nl (http://www.sidn.nl/).

References

1. Adida, B., Wikström, D.: Offline/online mixing. In: ICALP 2007. pp. 484–495 (2007)
2. Backes, M., Goldberg, I., Kate, A., Mohammadi, E.: Provably secure and practical onion routing. In: Proc. 25th IEEE Computer Security Foundations Symposium (CSF). pp. 369–385 (2012)
3. Backes, M., Kate, A., Mohammadi, E.: Ace: An efficient key-exchange protocol for onion routing. In: Proc. 11th ACM Workshop on Privacy in the Electronic Society (WPES). pp. 55–64 (2012)
4. Benaloh, J.: Simple verifiable elections. In: Proc. USENIX/Accurate Electronic Voting Technology Workshop (EVT). pp. 5–5 (2006)
5. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering 2(2), 77–89 (2012)

6. Berthold, O., Langos, H.: Privacy Enhancing Technologies: Second International Workshop, PET 2002 San Francisco, CA, USA, April 14–15, 2002 Revised Papers, chap. Dummy Traffic against Long Term Intersection Attacks, pp. 110–128. Springer Berlin Heidelberg (2003)
7. Berthold, O., Pfitzmann, A., Standtke, R.: The disadvantages of free mix routes and how to overcome them. In: Proc. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. pp. 30–45 (2001)
8. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Advances in Cryptology - CRYPTO 2013. pp. 410–428 (2013)
9. Brickell, J., Shmatikov, V.: Efficient anonymity-preserving data collection. In: Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006. pp. 76–85 (2006)
10. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: Advances in Cryptology — CRYPTO. pp. 169–187 (2005)
11. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 4(2), 84–88 (1981)
12. Chen, C., Asoni, D.E., Barrera, D., Danezis, G., Perrig, A.: HORNET: high-speed onion routing at the network layer. In: Proc. 22nd ACM Conference on Computer and Communications Security. pp. 1441–1454 (2015)
13. Danezis, G.: The traffic analysis of continuous-time mixes. In: 4th International Workshop on Privacy Enhancing Technologies. pp. 35–50 (2005)
14. Danezis, G., Diaz, C., Troncoso, C.: Two-sided statistical disclosure attack. In: 7th Privacy Enhancing Technologies Symposium. pp. 30–44 (2007)
15. Danezis, G., Serjantov, A.: Statistical disclosure or intersection attacks on anonymity systems. In: 6th International Workshop on Information Hiding. pp. 293–308 (2005)
16. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proc. 13th USENIX Security Symposium. pp. 303–320 (2004)
17. Dingledine, R., Shmatikov, V., Syverson, P.: Synchronous Batching: From Cascades to Free Routes, pp. 186–206. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
18. Dolev, D., Reischuk, R., Strong, H.R.: Early stopping in byzantine agreement. J. ACM 37(4), 720–741 (Oct 1990)
19. Evans, N.S., Dingledine, R., Grothoff, C.: A Practical Congestion Attack on Tor Using Long Paths. In: Proc. 18th USENIX Security Symposium. pp. 33–50 (2009)
20. Galteland, H., Mjølsnes, S.F., Olimid, R.F.: Attacks on cMix - Some small overlooked details. Cryptology ePrint Archive, Report 2016/729 (2016), http://eprint.iacr.org/2016/729
21. Goldberg, I., Stebila, D., Ustaoglu, B.: Anonymity and one-way authentication in key exchange protocols. Designs, Codes and Cryptography pp. 1–25 (2012)
22. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Onion routing. Commun. ACM 42(2), 39–41 (1999)
23. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (Apr 1995)
24. Golle, P., Jakobsson, M., Juels, A., Syverson, P.: Universal re-encryption for mixnets. In: Proc. Topics in Cryptology — CT-RSA 2004. pp. 163–178 (2004)
25. Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Advances in Cryptology - CRYPTO '96. pp. 201–215 (1996)
26. Jakobsson, M.: Flash mixing. In: Proc. 18th Annual ACM Symposium on Principles of Distributed Computing (PODC). pp. 83–89 (1999)
27. Jakobsson, M., Juels, A.: An optimally robust hybrid mix network. In: Proc. 20th Annual ACM Symposium on Principles of Distributed Computing. pp. 284–292 (2001)
28. Jansen, R., Tschorsch, F., Johnson, A., Scheuermann, B.: The sniper attack: Anonymously deanonymizing and disabling the Tor network. In: Proc. Network and Distributed System Security Symposium (NDSS'14) (2014)

29. Kate, A., Goldberg, I.: Using Sphinx to improve onion routing circuit construction. In: Proc. 14th Conference on Financial Cryptography and Data Security (FC). pp. 359–366 (2010)
30. Kate, A., Zaverucha, G.M., Goldberg, I.: Pairing-based onion routing with improved forward secrecy. ACM Trans. Inf. Syst. Secur. 13(4), 29:1–29:32 (Dec 2010)
31. Khazaei, S., Wikström, D.: Randomized partial checking revisited. In: Topics in Cryptology: CT-RSA 2013, pp. 115–128 (2013)
32. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle: An efficient communication system with strong anonymity. PoPETs 2016(2), 115–134 (2016)
33. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: Proc. 26th IEEE Symposium on Security & Privacy. pp. 183–195 (2005)
34. Ohkubo, M., Abe, M.: A length-invariant hybrid mix. In: Proc. Advances in Cryptology — ASIACRYPT 2000. pp. 178–191 (2000)
35. Øverlier, L., Syverson, P.: Improving efficiency and simplicity of Tor circuit establishment and hidden services. In: Proc. 7th Privacy Enhancing Technologies Symposium (PETS). pp. 134–152 (2007)
36. Øverlier, L., Syverson, P.F.: Locating hidden servers. In: Proc. 27th IEEE Symposium on Security & Privacy. pp. 100–114 (2006)
37. Park, C., Itoh, K., Kurosawa, K.: Efficient anonymous channel and all/nothing election scheme. In: EUROCRYPT '93. pp. 248–259 (1994)
38. Pfitzmann, A., Waidner, M.: Networks without user observability – design options. In: Proc. Advances in cryptology—EUROCRYPT '85. pp. 245–253 (1986)
39. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management (Aug 2010), v0.34
40. Pfitzmann, B.: Breaking an efficient anonymous channel. In: Proc. Advances in Cryptology — EUROCRYPT'94. pp. 332–340 (1995)
41. Raymond, J.F.: Traffic analysis: Protocols, attacks, design issues, and open problems. In: Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. pp. 10–29 (2001)
42. Reed, M., Syverson, P., Goldschlag, D.: Anonymous connections and onion routing. IEEE J-SAC 16(4), 482–494 (May 1998)
43. Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2P Mixing and Unlinkable Bitcoin Transactions. In: NDSS'17 (2017)
44. Serjantov, A., Dingledine, R., Syverson, P.: From a trickle to a flood: Active attacks on several mix types. In: 5th International Workshop Information Hiding. pp. 36–52 (2003)
45. Serjantov, A., Sewell, P.: Passive attack analysis for connection-based anonymity systems. In: ESORICS 2003. pp. 116–131 (2003)
46. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distributed Computing 2(2), 80–94 (1987)
47. Sun, Y., Edmundson, A., Vanbever, L., Li, O., Rexford, J., Chiang, M., Mittal, P.: Raptor: Routing attacks on privacy in Tor. In: Proc. 24th USENIX Security Symposium. pp. 271–286 (2015)
48. Syta, E., Corrigan-Gibbs, H., Weng, S., Wolinsky, D., Ford, B., Johnson, A.: Security analysis of accountable anonymity in dissent. ACM Trans. Inf. Syst. Secur. 17(1), 4:1–4:35 (2014)
49. The Tor project. https://www.torproject.org/ (2003), accessed Apr 2017