cMix: Anonymization by High-Performance Scalable Mixing

David Chaum
Voting Systems Institute, USA
david@chaum.com

Debajyoti Das, Aniket Kate
Purdue University, USA
{das48,aniket}@purdue.edu

Farid Javani, Alan T. Sherman
Cyber Defense Lab, UMBC, USA
{javani1,sherman}@umbc.edu

Anna Krasnova
Radboud University, NL
anna@mechanical-mind.org

Joeri de Ruiter
University of Birmingham, UK
j.deruiter@cs.bham.ac.uk

Abstract

cMix is a suite of cryptographic protocols that can replace today's dominant chat systems, offering superior confidentiality and anonymity, while providing comparable performance to users. cMix permutes batches of uniform-length messages through a fixed cascade of nodes and moves all expensive public-key operations into precomputations that can be carried out using separate dedicated hardware at each node.

cMix provides payload secrecy, sender-recipient unlinkability, sender anonymity, and sender authentication for recipients, unless all cMix nodes are compromised. For each batch, the adversary may know all senders and all recipients of traffic in the underlying packet-switched network, yet the adversary cannot link any sender to recipient.

cMix provides fast delivery of messages, in both the forward and reverse directions, by having each node perform only a small number of symmetric-key and simple group operations (no modular exponentiations) in real time. Performance benefits include moderately low latency (despite large batch sizes) and efficient utilization of node machines. Senders (e.g., smartphones) perform their part of the cMix real-time protocols with similarly modest amounts of computation, resulting in negligible additional delay, battery, or bandwidth usage. The performance of cMix scales linearly in terms of the number of nodes, users, and messages,

Our presentation includes a detailed specification of cMix, simulation-based security proofs, and anonymity analysis. We have implemented cMix on clients on the Android platform, and we give performance analysis, both modelled and measured, of two working prototypes currently running in the cloud.

1 Introduction

Untraceable (anonymous) and unlinkable communication are fundamental to freedom of inquiry, freedom of expression, and increasingly to online privacy, including person-to-person communication. Employing anonymous communication networks has become increasingly popular across the world over the last fifteen years. This popularity is exemplified by use of the *onion routing* network Tor [50].

The Tor network, however, is susceptible to a variety of traffic-analysis attacks [24, 29, 38, 49], based in part on Tor's non-uniform message size and timing. Recent anonymity analyses [4, 31] raise doubts on the quality of anonymity possible using so-called onion routing. By contrast, *mixing networks* (also called mixnets) [13,20,21,27,30,48,51] are inherently less susceptible to these traffic-correlation and network-level attacks. Existing mixnet designs, however, introduce a significant performance overhead to users and mix nodes.

In this paper, we present, implement, and analyze *cMix*, a new suite of fast cryptographic protocols for a variety of anonymity services, including chat. Using precomputation of a group-homomorphic encryption function, cMix avoids all expensive real-time public-key operations (including modular exponentiation) of senders, nodes, and receivers. By performing precomputations on dedicated separate hardware attached to each node, cMix simultaneously permits moderately low message latency, large batch sizes, and high processor utilization of the nodes. cMix's fast performance and key management make it highly scalable for deployment with large anonymity sets and large numbers of nodes.

The main novel and significant contributions of cMix are three: precomputation, key management wherein each sender (and optionally each receiver) establishes a separate shared key with each node, and the integration of these elements and other well-known building blocks to produce a practical and useful anonymity system with strong anonymity properties. In the rest of this section we highlight some of the design features that go into cMix.

By processing messages in large batches, and by requiring all messages of a batch to have the same length, cMix avoids flow-analysis attacks that plague many existing anonymity systems. As for most batched mixnets, cMix can support a variety of ways to manage batches. One option is "threshold and timing" [46], where the system "fires" (sends along all messages in the buffer) every t seconds, provided there are at least β' messages in the buffer. These choices have implications on anonymity and latency but are independent of the cMix system.

In cMix, each sender choosing to participate in a particular round sends an input to the cMix system, which after passing through a fixed cascade of mix nodes, arrives in an output buffer. Unless all nodes collude, the outputs are unlinkable to the inputs, even if the adversary knows for each batch the set of senders and the set of receivers. By contrast, Tor cannot handle this powerful adversarial model.

We envision each mix node to be a powerful highly-reliable computing system, preferably located in a separate region of the world. With traditional mixnets, to achieve low latency of messages, node hardware must be idle for much of the time. Attempts to increase machine utilization by pipelining require smaller batch sizes. By contrast, cMix's use of precomputation facilitates higher machine utilization by having the expensive public-key operations performed on separate dedicated hardware hardware far in advance. Furthermore, these precomputations are highly parallelizable. We assume enough computational resources are provided for the precomputations that they do not become a real-time bottleneck.

The exact format of an input depends on the application. For example, for some applications, each input might be an ordered pair (B_i, M_i), where B_i is the recipient and M_i is the payload. The sender encrypts the entire input using message keys shared by the sender and each mix node. Each sender establishes a long-term shared key separately with each cMix node. Each such shared key seeds a cryptographic pseudorandom number generator to produce a sequence of message keys, the next in the sequence being selected when the sender chooses to participate in a particular round. Each sender encrypts its input with modular multiplication by the next message key for each cMix node.

During the real-time mixing of an input batch, each cMix node replaces each of its message keys with a precomputed random value. Then, using another random value and a random precomputation also determined in the precomputation, each node includes the new random value and applies its permutation to the buffer of messages in the batch. Finally, using a value that was precomputed in a multiparty-secure manner from all the random values and permutations, a single group operation cancels all random numbers, leaving the permuted output batch. By freeing mix nodes from performing expensive public-key operations in real time, real-time mixing is much faster than in previous mix networks. For users, the amount of computation on their smart phones (and thus the corresponding power usage) is also reduced.

cMix can be integrated in a variety of ways into a variety of mechanisms for providing anonymity services. Typically, each sender will send its input to a simple *untrusted* "network handler," who will arrange the arriving inputs into batches. As is typically true for mix networks, receivers do not necessarily contact the nodes, when the network handler (on behalf of nodes) can make the outputs from the output buffer available to the final receivers. The unlinkability of inputs to outputs does not depend on the correct operation of the network handler.

Because each input is encrypted, the network handler cannot read the payload of any input when she receives the input; hence, in particular, the network handler cannot read recipient information if present in the payload when she receives the input. Recipient information, if present, is readable for any message in the output buffer, but no one (not even the network handler) can link the recipient to the sender.

In addition to guaranteeing unlinkability of inputs and outputs, cMix permits end-to-end confidential communication between the sender and the receiver as well as a confidential sender authentication to the receiver, without requiring any external public-key infrastructure for the users. This unique feature of combining message confidentiality and sender authentication with anonymity, which was not possible in any previous anonymity system, emerges naturally from the key management and communication flows of the cMix protocol.

Our contributions include:

1. A suite of new fast scalable cryptographic anonymity protocols, cMix, based on precomputation and on permuting uniform-length messages in batches through a fixed cascade of nodes.
2. Simulation-based security proofs, and anonymity analysis for cMix.
3. An implementation of cMix running in the cloud (Amazon Web Services) and on clients on the Android platform, and performance analysis of the cMix protocol based on modelling and on benchmarks from our cloud implementations.
4. A cryptographic commitment-based defense against active tagging attacks, in which attacks the adversary modifies messages at two different hops to extract information about the receivers.

2 Background and Related Work

Prior practical anonymity systems are based primarily on mixnets or onion routing.

2.1 Mix Networks

In 1981, Chaum [13] introduced the concept of mixing networks (or mixnets) and gave the basic cryptographic protocols whereby messages from a set of users are relayed by a sequence of trusted intermediaries, called *mix nodes* or *mixes*. A mix node 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. This process makes the task of tracing an individual message through the network difficult. Chaum's paper describes batched and unbatched versions of mixing. In more than three decades of research on mixnets, many mix network designs have been proposed including [20,21,27,30,48,51], and a few have been implemented [19,37].

Anonymizing communication through a mix network comes with computation and communication overheads: user messages are batched to create an anonymity set (and therefore delayed), and they are padded or truncated to a standard length to prevent traffic analysis. Furthermore, in current mix networks, multiple *public-key* encryption layers are used to encapsulate the routing information necessary to relay the message through a sequence of mixes. Our novel mixnet architecture, cMix, reduces the computation overhead by replacing real-time public-key operations with symmetric-key operations.

Some early mixing protocols [13, 21] were based on heuristic security arguments, and weaknesses have been discovered with them [43, 48]. By contrast, most of the recent mixing formats [11, 20, 36, 43] are designed with provable security. We also achieve provable security for cMix: we define a simple ideal functionality for cMix and prove simulation-based security for the protocol.

A key distinction of cMix is its shifting of all public key operations to the precomputation phase. Moreover, these public key operations are performed only by the nodes, and no user needs to be involved. In the literature, to the best of our knowledge, only Adida and Wikström [1] have considered an offline/online approach to mixing earlier; however, their scheme still requires several public-key operations in the online phase.

Another notable difference between cMix and most previous mixnets is that each mix node knows all senders. This difference does not weaken the adversarial model because the adversary is expected to know all participants of the mixing round, and in cMix the unlinkability between a sender and a receiver is still ensured, by even any one uncorrupted mix node. On the other hand, this can empower cMix nodes to perform other tasks such as end-to-end secure messaging without introducing a public-key infrastructure of the participants.

2.2 Onion Routing

Higher latency of traditional mix networks can be unsatisfactory for several communication scenarios such as web search or instant messaging. Over the years, a significant number of *low-latency* anonymity networks have been proposed [2,5,11,14,25,32,33,40], and some have been extensively employed in practice [23,50].

Common to many of them is *onion routing* [26, 41], 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 small path 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 [24, 29, 38, 49], such as Tor [50], 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.

In practice, Tor fails to provide ironclad anonymity. A recent blog [16] reports that criminal users of Tor have been deanonymized, and that researchers at Carnegie Mellon University were paid at least $1 million to assist the FBI in this task.

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 users—a major advantage. Each of our users establishes a separate shared secret with each mix node, but this key establishment is performed infrequently, and in contrast with onion routing, users do not perform anonymous key agreement [5, 23, 25, 33] using a telescoping approach or layered public-key encryption. 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 Overview of cMix

cMix is a new mixnet protocol that provides anonymous communications among users. As shown in Figure 1, the core of the system comprises n mix nodes, which process discrete batches of messages. A simple network handler[1] arranges the inputs into batches. The main goal is to ensure unlinkability between messages entering and leaving the system, though it is known which users are communicating in any batch. cMix precomputes all slow public-key encryption, enabling all real-time computations to be carried using only fast multiplications.

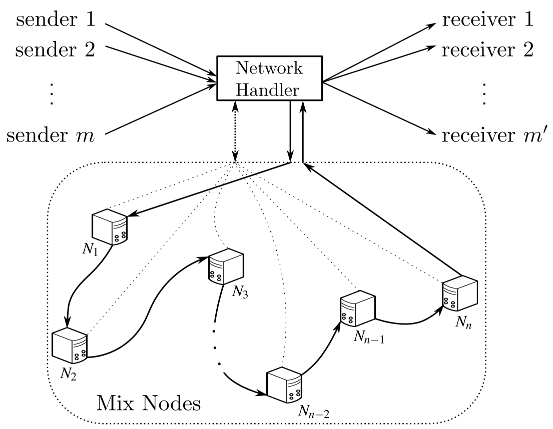


Figure 1: The cMix communication model.

3.1 Communication Model

Let m be the number of users of the cMix system, which includes a sequence of n mix nodes $N_1, N_2, \ldots, N_n$. Each node can process β messages at a time, where $\beta \leq m$. During a precomputation phase, mix nodes fix a permutation of future incoming β messages. In the real-time communication, the nodes permute the messages using this permutation.

We split the real-time phase into rounds, where each round applies one permutation used by the mix nodes to one batch of messages. Each round can be divided into sub-rounds, which can differ by application. Let us consider anonymous web-browsing as an application of cMix (for more about cMix applications, see Section 9.2). In that case a single round is divided into two sub-rounds, one for the delivery of the forward message (browsing request), one for the confirmation message of the request delivery sent by the last mix node. All messages transmitted during one sub-round have the same length and are processed simultaneously.

[1]We introduce the network handler abstraction primarily to improve readability. As for traditional mix networks, all of its functionalities can be realized by the mix nodes.

At the beginning of a round the first mix node accepts up to β messages that require a similar sub-round structure to be executed. For each round, the handler arranges β messages into the input buffer of the first mix node, sorting the messages by lexicographical order. All other messages are not accepted and are sent in a subsequent round.

cMix follows the threshold and timed mixing strategy from [46], where the handler starts a new round every t seconds only if it has at least β' messages in the buffer, for some parameter $\beta' < \beta$, where we expect at least β' users to be using the system at any given time. When a smaller number of users is active, this strategy can lead to increased latency or even disruption. One design choice, at the cost of increased engery consumption, is to inject dummy messages when needed to ensure enough traffic to have β messages every t seconds. The details depend on the application and are orthogonal to mixnet design.

3.2 Adversarial Model

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

The goal of the adversary is to compromise the anonymity of the communication initiator, or to link inputs and outputs of the system. We consider applications where initiators are users of the cMix system. We do not consider adversaries who aim to launch denial-of-service (DOS) attacks.

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

3.3 Solution Overview

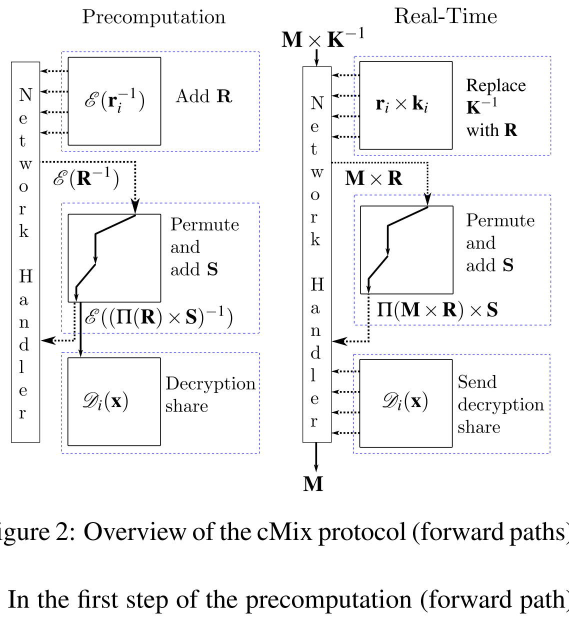
Before using the system, each sender must establish a shared symmetric key separately with each of the mix nodes. For each mix node N_i and each user U_j, let $\mathcal{K}_{i,j}$ denote their shared key. This key establishment can be carried out, for instance, using the Diffie-Hellman (DH) key agreement protocol, with forward secrecy (compromise of a shared key does not compromise any past shared key) and at least one-way authentication (the sender is convinced she is communicating with the true mix node). During this process, each user can also be assigned one or more unique pseudonyms as her identities in the cMix system; doing so better protects the user's identity and her interactions with the system.

When communicating with the mix network, user U_j will encrypt or decrypt each of her messages using message keys derived from her keys shared with each node N_i. Specifically, the next message key $k_{i,j}$ is the next output of a forward-secure pseudorandom number generator with seed $\mathscr{K}_{i,j}$. To encrypt a message, the user first computes a composite key using the derived message keys: $K_j = \prod_{i=1}^{n} k_{i,j}$. Then she can encrypt her first message M_1 as $M_1 \times K_j^{-1}$.

cMix processes each batch of messages in two phases: precomputation and real-time. During each of these phases, cMix performs a forward and reverse path of computations, each organized in steps. Each mix node organizes the messages of the current batch in a buffer (also called a map). During one step of each path, each node permutes the messages within the buffer.

Each node associates each shared key with a slot in its message buffer. During the forward path of the real-time phase, each node replaces its shared key for each slot with a randomly-generated value from the precomputation. During the reverse path, each node multiplies back in the shared keys. In doing so, the real-time phase avoids any expensive public-key operations.

The Protocol Steps. Figure 2 summarizes the precomputation and real-time phases of the forward paths in cMix. Each step is denoted by a solid box. Section 4.1 defines our notations, and Figure 7 (in Appendix) fills in many of the details for each step.



Precomputation
Real-Time
$\mathbf{M} \times \mathbf{K}^{-1}$
Network Handler
$\mathscr{E}(\mathbf{r}_i^{-1})$
Add $\mathbf{R}$
$\mathscr{E}(\mathbf{R}^{-1})$
Permute and add $\mathbf{S}$
$\mathscr{E}((\Pi(\mathbf{R}) \times \mathbf{S})^{-1})$
$\mathscr{D}_i(\mathbf{x})$
Decryption share
$\mathbf{r}_i \times \mathbf{k}_i$
Replace $\mathbf{K}^{-1}$ with $\mathbf{R}$
$\mathbf{M} \times \mathbf{R}$
Permute and add $\mathbf{S}$
$\Pi(\mathbf{M} \times \mathbf{R}) \times \mathbf{S}$
$\mathscr{D}_i(\mathbf{x})$
Send decryption share
$\mathbf{M}$

Figure 2: Overview of the cMix protocol (forward paths).

In the first step of the precomputation (forward path), each node N_i generates a random value $r_{i,j}$ for each slot j in its message buffer. Each node encrypts its vector $\mathbf{r}_i{}^{-1}$ of the inverses of these values and sends the resulting vector $\mathscr{E}(\mathbf{r}_i{}^{-1})$ to the network handler. The handler, exploiting the group homomorphic property of $\mathscr{E}$, computes the (component-wise) direct product $\mathscr{E}(\mathbf{R}^{-1})$ of the encrypted vectors and sends the result to the first mix node.

In the second step of the precomputation, each node N_i in order permutes the message buffer with its random permutation π_i. It also multiplies in another vector of random values $\mathbf{s}_i{}^{-1}$. The result at the exit of the last node is $\mathscr{E}((\mathbf{\Pi}(\mathbf{R}) \times \mathbf{S})^{-1})$, where $\mathbf{\Pi}$ is the component-wise compositions of the π_i's, and $\mathbf{S}$ is the direct product of the permuted $\mathbf{s}_i$'s. This result is sent to the mix nodes for decryption.

In the third step of the precomputation, each node N_i computes its decryption share $\mathscr{D}_i(c)$ of the result from Step 2. Only with knowledge of all of these shares can one perform the decryption. In the final step of the real-time phase, each node will send these shares to the network handler, who will decrypt the permuted messages. The purpose of this subtle third step is to prevent certain "tagging" attacks, in which a corrupt node marks an output. Each node sends a signed commitment of its share to the other nodes, and each node verifies all of these commitments. Alternatively, the correctness of the shares can be established by a zero-knowledge proof.

We now explain the real-time phase. In the first step of the real-time computation, each mix node i sends the product of its vector of shared keys $\mathbf{k}_i$ with its vector of random values $\mathbf{r}_i$ to the network handler. The network handler then multiplies all these values together with the received messages. This action, which uses only multiplications, transforms the encrypted input $\mathbf{M} \times \mathbf{K}^{-1}$ to $\mathbf{M} \times \mathbf{R}$. Here, $\mathbf{K}$ is the direct product of the $\mathbf{k}_i$'s, and $\mathbf{R}$ is the direct product of the $\mathbf{r}_i$'s.

In the second step of the real-time phase, each node i in order permutes its message buffer with its permutation π_i and, to hide π_i, multiples in its vector of random values $\mathbf{s}_i$. The result at the exit of the last mix node is $\mathbf{\Pi}(\mathbf{M} \times \mathbf{R}) \times \mathbf{S}$. The exit node sends this result to the network handler.

In the final step of the real-time phase, each node sends to the network handler its decryption share $\mathscr{D}_i(c)$, which it computed from the last step of the precomputation. With all of these shares, the network handler decrypts the message. The network handler sees the unencrypted payloads but cannot link them to the inputs.

4 The Core cMix Protocol

After explaining our notation, we describe the cMix protocol in detail, including how it detects tagging attacks.

4.1 Preliminaries

For simplicity we assume here that the system already knows which user will use which slot. When implementing the system this 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 $\mathbf{G}$. We use a multi-party group-homomorphic cryptographic scheme based on ElGamal, described by Benaloh [7]. Every node N_i in the scheme holds a share $e_i \in \mathbb{Z}_p^*$ of the secret key. The public key d of the scheme can be computed using the secret shares: $d = \prod_i g^{e_i}$. Using this scheme, a value r is encrypted as follows: $(g^x, r \times d^x)$, for $x \in_r \mathbb{Z}_p^*$. We call g^x the *random component* and $r \times d^x$ the *message component* of the ciphertext. To decrypt a ciphertext $(g^x, r \times d^x)$, all parties need to cooperate. Every node N_i computes a so-called *decryption share* from the random component of the ciphertext: $\mathscr{D}_i(g^x) = (g^x)^{-e_i}$. The original message is then retrieved by multiplying all the decryption shares with the message component: $r \times d^x \times \prod_{i=1}^n (g^x)^{-e_i} = r$.

Within cMix we use the following notation for various functions and variables:

– e_i: the share of node N_i of the secret key e.

– d: the public key of the system, based on the node shares of the secret key.

– $\mathscr{E}(\cdot)$: ElGamal encryption under the system's public key. When applying encryption on a vector of values, each value in the vector is encrypted individually and the result is a vector of ciphertexts.

– $\mathscr{D}_i(\cdot)$: the decryption share of node N_i based on the random component of a ciphertext and the node'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.

– π_i: a random permutation of the β slots used by N_i. The inverse of the permutation is denoted by π_i^{-1}.

– $\mathbf{\Pi}_i(a)$: the permutation performed by cMix through N_i, i.e., the composition of all individual permutations:

$$\mathbf{\Pi}_i(a) = \begin{cases} \pi_1(a) & i = 1 \\ \pi_i(\mathbf{\Pi}_{i-1}(a)) & 1 < i \leq n. \end{cases}$$

– $\mathbf{\Pi}'_i(a)$: the inverse permutation of slots performed by the mixnet for the return path through node N_i:

$$\mathbf{\Pi}'_i(a) = \begin{cases} \pi_n^{-1}(a) & i = n \\ \pi_i^{-1}(\mathbf{\Pi}'_{i-1}(a)) & 1 \leq i < n. \end{cases}$$

– $k_{i,j}, k'_{i,j} \in \mathbf{G}^*$: secret key shared between node N_i and the sending user of slot j used to blind messages and responses, respectively. These keys are group elements.

– $\mathbf{k}_i$: vector of keys shared between node N_i and users for β slots; $\mathbf{k}_i = (k_{i,1}, k_{i,2}, \ldots, k_{i,\beta})$.

– $K_j, K'_j \in \mathbf{G}^*$: the product of all shared keys for the sending user of slot j: $K_j = \prod_{i=1}^n k_{i,j}$ and $K'_j = \prod_{i=1}^n k'_{i,j}$. The user for this slot stores the inverse of these keys, K_j^{-1} and $K_j'^{-1}$, to blind and unblind messages and responses, respectively.

– $\mathbf{K}$, $\mathbf{K}^{-1}$: vectors of products of shared keys for β slots and their inverses, respectively; $\mathbf{K} = (K_1, K_2, \ldots, K_\beta)$ and $\mathbf{K}^{-1} = (K_1^{-1}, K_2^{-1}, \ldots, K_\beta^{-1})$.

– $M_j, M'_j \in G^*$: the message and the response sent by user U_j in the forward and return phase, respectively. 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 the forward path we have the following values:

– $r_{i,a}, s_{i,a} \in \mathbf{G}^*$: random values of node N_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 node N_i. Similarly, $\mathbf{s}_i$ is also a vector of random values for node N_i.

– $\mathbf{R}$: the direct product of all local random values; i.e., $\mathbf{R}_i = \prod_{j=1}^i \mathbf{r}_j$.

– $\mathbf{S}$: the product and permutation of all local random s values:

$$\mathbf{S}_i = \begin{cases} \mathbf{s}_i & i = 1 \\ \pi_i(\mathbf{S}_{i-1}) \times \mathbf{s}_i & 1 < i \leq n. \end{cases}$$

For the return path we use the following corresponding messages:

– $s'_{i,a} \in \mathbf{G}^*$: random value of node N_i for slot a.

– $\mathbf{S}'$: the product and permutation of all local random s' values:

$$\mathbf{S}'_i = \begin{cases} \mathbf{s}'_i & i = n \\ \pi_i^{-1}(\mathbf{S}'_{i+1}) \times \mathbf{s}'_i & 1 \leq i < n. \end{cases}$$

We introduce an additional entity called the *network handler* that performs non-sensitive computations, such as computing the product of values output by nodes. Alternatively, the handler role could be performed by any of the nodes. Or, the computations of the handler could be replaced by a pass through the mixnet, during which every node multiplies its local value with the value it received from the previous node. This strategy would balance the computational load over the nodes but might increase latency, because the values need to be forwarded after every local computation, whereas the network handler can start computing when it receives the first values and continue processing while the remaining values arrive.

4.2 Detailed Description

4.2.1 Precomputation Phase

Here we discuss the precomputation phase to compute the values that are necessary for one real-time phase. The final goal of this phase is for the nodes together to compute the values $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ and $\mathscr{E}(\mathbf{S}_1'^{-1})$ that are used in the real-time phase for the forward and return path, respectively. Below we discuss how these values are computed by the system.

Forward - Step 1 (preprocessing). The nodes start by computing $\mathscr{E}(\mathbf{R}_n^{-1})$ by sending the encryption of their local $\mathbf{r}_i^{-1}$ to the network handler. The network handler computes the product of all the encryptions of the individual values to produce the output of this step: $\mathscr{E}(\mathbf{R}_n^{-1}) = \prod_{i=1}^{n} \mathscr{E}(\mathbf{r}_i^{-1})$. The handler then sends this value to the first node as input for the second step.

Forward - Step 2 (mixing). In this step, the nodes exchange the following messages:
node $N_i \longrightarrow$ *node* N_{i+1} :

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

The last node computes $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1}) = \pi_n(\mathscr{E}(\mathbf{\Pi}_{n-1}(\mathbf{R}_n^{-1}) \times \mathbf{S}_{n-1}^{-1})) \times \mathscr{E}(\mathbf{s}_n^{-1})$. It sends the vector of random components from the ciphertexts of $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ to the nodes and stores the message components of $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ locally for use in the real-time phase.

Forward - Step 3 (postprocessing). Using the random components received, each node computes its individual decryption share for $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ and stores it locally for use in the real-time phase. Each node publishes a commitment to its decryption shares, which is necessary to prevent a tagging attack in the real-time phase (see Section 4.3).

The value needed for the return path is computed in a similar way, though in the opposite direction and without the *r* values.

Return - Step 1 (mixing). The nodes compute $\mathscr{E}(\mathbf{S}_1'^{-1})$ by sending the following messages:
node $N_i \longrightarrow$ *node* N_{i-1} :

$$\mathscr{E}(\mathbf{S}_i'^{-1}) = \begin{cases} \mathscr{E}(\mathbf{s}_n'^{-1}) & i = n \\ \pi_i^{-1}(\mathscr{E}(\mathbf{S}_{i+1}'^{-1})) \times \mathscr{E}(\mathbf{s}_i'^{-1}) & 1 < i < n. \end{cases}$$

The first node now computes $\mathscr{E}(\mathbf{S}_1'^{-1}) = \pi_1^{-1}(\mathscr{E}(\mathbf{S}_2'^{-1})) \times \mathscr{E}(\mathbf{s}_1'^{-1})$. The random components are sent to the other nodes and the message components are stored locally for use in the real-time phase.

Return - Step 2 (postprocessing). As for the forward path, each node computes its decryption share for $\mathscr{E}(\mathbf{S}_1'^{-1})$ and stores it locally for use in the real-time phase.

4.2.2 Real-time Phase

For the real-time phase each user constructs its message for slot j by taking its message M_j and multiplying it with the inverse of the combined shared key K_j to compute the blinded message $M_j \times K_j^{-1}$, which it sends to the network handler. The network handler collects the messages and combines them to yield the vector $\mathbf{M} \times \mathbf{K}^{-1}$.

Forward - Step 1 (preprocessing). In the first step of the forward path, the $\mathbf{K}^{-1}$ values are replaced by the $\mathbf{r}$ values of each node. Every node N_i sends the values $\mathbf{k}_i \times \mathbf{r}_i$ to the network handler. The network handler uses these values 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 the first node as input to the next step.

Forward - Step 2 (mixing). The next step mixes the messages: *node* $N_i \longrightarrow$ *node* N_{i+1} :

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

The last node computes $\mathbf{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n = \pi_n(\mathbf{\Pi}_{n-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{n-1}) \times \mathbf{s}_n$ and a commitment to this value. This commitment is then sent to all other nodes.

Forward - Step 3 (postprocessing). Upon receiving the commitment from the last node, every other node N_i sends its precomputed decryption shares for $(\mathbf{x}, \mathbf{c}) = \mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ to the network handler. The last node N_n sends the multiplication of the result from the previous step with its decryption share and the message component from the precomputation phase: $\mathbf{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \mathscr{D}_n(\mathbf{x}) \times \mathbf{c}$. The handler uses the decryption shares to decrypt the precomputed $\mathscr{E}((\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$ and retrieves the permuted messages:

$$\begin{aligned}&\mathbf{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times \prod_{i=1}^{n} \mathscr{D}_i(\mathbf{x}) \times \mathbf{c}\\ &= \mathbf{\Pi}_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n \times (\mathbf{\Pi}_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1}\\ &= \mathbf{\Pi}_n(\mathbf{M}).\end{aligned}$$

The messages are published or delivered and the responses to these messages are collected in $\mathbf{M}'$.

Return - Step 1 (mixing). For the return path we start with the reversed permutations: *node* $N_i \longrightarrow$ *node* N_{i-1} :

$$\mathbf{\Pi}_i'(\mathbf{M}') \times \mathbf{S}_i' = \begin{cases} \pi_n^{-1}(\mathbf{M}') \times \mathbf{s}_n' & i = n \\ \pi_i^{-1}(\mathbf{\Pi}_{i+1}'(\mathbf{M}') \times \mathbf{S}_{i-1}') \times \mathbf{s}_i' & 1 < i < n. \end{cases}$$

The first node computes $\mathbf{\Pi}'_1(\mathbf{M}') \times \mathbf{S}'_1 = \pi_1^{-1}(\mathbf{\Pi}'_2(\mathbf{M}') \times \mathbf{S}'_2) \times \mathbf{s}'_1$ and commits to this value as before. Again, the value is sent to the network handler.

Return - Step 2 (postprocessing). In the last step, every node N_i retrieves its precomputed decryption share $\mathscr{D}_i(\mathbf{x}')$ for $(\mathbf{x}', \mathbf{c}') = \mathscr{E}(\mathbf{S}_1'^{-1})$ and uses it to compute $\mathscr{D}_i(\mathbf{x}') \times \mathbf{k}'_i$. The first node sends its decryption share multiplied with the message component from the precomputation phase to the network handler: $\mathscr{D}_1(\mathbf{x}') \times \mathbf{c}'$. After receiving the commitment from the first node, the other nodes send their decryption shares to the network handler. Finally, the handler uses the decryption shares to retrieve the permuted messages blinded with $\mathbf{K}'$:

$$\begin{aligned}
&\mathbf{\Pi}'_1(\mathbf{M}') \times \mathbf{S}'_1 \times \prod_{i=1}^{n}(\mathscr{D}_i(\mathbf{x}') \times \mathbf{c}' \times \mathbf{k}'_i) \\
&= \mathbf{\Pi}'_1(\mathbf{M}') \times \mathbf{S}'_1 \times \mathbf{S}_1'^{-1} \times \mathbf{K}' \\
&= \mathbf{\Pi}'_1(\mathbf{M}') \times \mathbf{K}'.
\end{aligned}$$

The messages are published or delivered, and then each user of slot j can unblind its response by multiplying it with its shared key $K_j'^{-1}$.

4.3 Detecting Tagging Attacks

Tagging attacks are a potential threat to all mixnets. If it is possible to determine whether an output message is valid, for example because it is an English message, a tagging attack can determine the output slot corresponding a specific input slot, with cooperation of the network handler and any of the nodes. In the preprocessing step of the real-time phase, the compromised node N_i replaces one value in the vector it sends to the network handler. In the slot j for which it wants to learn the recipient, the compromised node inserts $k_{i,j} \times r_{i,j} \times t$, where t can be a random group element. In the postprocessing step, the handler sends decryption shares of all other nodes and the output of the mixing step to the compromised node. Using its decryption shares, the compromised node retrieves the messages. It can then determine for which slot the message is invalid but becomes valid when divided by t. For this slot it multiplies its decryption share with t^{-1} and sends the decryption shares to the network handler as usual. The output of the system remains the permuted original messages.

To prevent this attack, in the last step each node commits to its decryption share, to prevent it from changing during the real-time phase to cancel out any added tag. Similarly, the last node commits to the output of the mixing step. To detect whether any tagging took place, all values are compared to their commitments. This comparison can be performed, online or after the real-time phase, by the nodes and/or by independent auditors. Such an audit can prove that no node acted maliciously.

5 Performance Analysis

We analyze the performance of cMix for the forward path. We will express running times in terms of the time $\tau_{\mathscr{E}}$ to perform one public-key encryption, the time $\tau_{\mathscr{D}}$ to compute one public-key decryption share, and the time $\tau_{\mathscr{M}}$ to perform one group multiplication. For our encryption function, $\tau_{\mathscr{D}} = \tau_{\mathscr{E}}/2$. Recall that β is the number of messages processed per batch, and n is the number of mix nodes. We do not consider any parallel computations.

For each precomputation phase for the forward path, cMix computes $2n\beta$ encryptions and $n\beta$ decryption shares, (two encryptions and one decryption share per slot). Thus, this phase takes $(5/2)n\beta\tau_{\mathscr{E}}$ time for the public-key operations.

In the real-time phase for the forward path, cMix performs $\beta(4n+1)$ group multiplications. For every slot and node, one multiplication removes the shared keys, and two add the r and s values. In addition, the network handler performs for each slot n multiplications combining the decryption shares, and one removing the precomputed value from the result. Thus, the real-time phase takes $\beta(4n+1)\tau_{\mathscr{M}}$ time for all multiplications.

It follows that the computations in the real-time phase are approximately $(5/8)(\tau_{\mathscr{E}}/\tau_{\mathscr{M}})$ faster than the precomputation phase when performed in a single thread. Therefore, if the real-time phase processes, for example, α messages per second, the system needs to be able to perform $(5/8)(\tau_{\mathscr{E}}/\tau_{\mathscr{M}})\alpha$ precomputations per second. It could do so, for example, with dedicated machines.

The messages that pass through the network either contain 2β group elements for the encrypted values or β group elements for the other messages. Assuming a group element can be represented in λ bits, the messages have average sizes of $2\beta\lambda$ and $\beta\lambda$, respectively.

6 Security and Anonymity Analyses

In this section, we overview the ideal/real world paradigm-based simulation security analysis of the core cMix protocol. We present an informal description of our ideal functionality F_{cMix}, the simulation security definition, and our main theorem. Then we analyze anonymity properties of cMix in the AnoA framework [3] and against well-known attacks on mixnets. For a complete F_{cMix} description and security proof sketch, see Appendix A .

6.1 Simulation Security Analysis

Ideal Functionality. In the ideal world we assume the existence of a trusted third party (TTP). Each mix-node in the network is connected to every other mix-node as well as to the TTP via private authenticated channels. In our ideal functionality $F_{\text{c}Mix}$, we use the message-based state transitions and consider submachines for all n mix nodes. To communicate with each other through messages and data structures, these submachines share a memory space in the functionality.

An adversary can observe, delay, or stop the messages going from one node to another, but she cannot read the message contents. If any node is compromised, however, it forwards to A every message that it receives, and waits for the adversary's instruction for every outgoing message. Before sending any message, each uncompromised node sends a notification to the (network) adversary, waits for a reply, and proceeds only after getting an approval from the adversary.

$F_{\text{c}Mix}$ starts by processing incoming messages from users, where the messages for compromised users are chosen by the adversary A. $F_{\text{c}Mix}$ accepts only one message per user per batch. When the batch reaches β messages, precomputation starts.

$F_{\text{c}Mix}$ triggers start of precomputation of the forward direction by sending a message to the entry node. The entry node creates a random permutation and stores it in a database. It forwards a command to the next node to start precomputation, which node repeats the same procedure. When the last node finishes, it notifies the other nodes that the precomputation is complete. The last node also notifies the network handler and $F_{\text{c}Mix}$. $F_{\text{c}Mix}$ then initiates precomputation for the return direction, which executes in a similar fashion with two differences: first, each node stores the inverse of the precomputation generated earlier, and second, the first node notifies $F_{\text{c}Mix}$ when the step is complete.

Next, $F_{\text{c}Mix}$ informs the entry node to begin the real-time phase. Starting from the entry node, each node retrieves its previously stored precomputation (the only permutation in $F_{\text{c}Mix}$), applies it to the current batch, and triggers the next node to perform the same operation. Once the last node finishes, it informs the network handler and all nodes to start postpocessing. Each node confirms (pertaining to decryption in cMix) to the network handler whether it has finished the precomputation phase correctly. Once the handler receives confirmation from all nodes, it sends notification to $F_{\text{c}Mix}$ that the real-time phase is complete. $F_{\text{c}Mix}$ then forwards user messages to the handler, who in turn forwards them to A. Once A returns the set of reply messages, the handler notifies $F_{\text{c}Mix}$, which in turn notifies the last node to start the real-time phase in the return direction. The return direction operates in a similar fashion. For more details, see Appendix A.

We now define the concept of simulation security, which captures intuitively the conditions under which a cryptographic protocol constitutes a secure realization of the ideal world defined above. PPT refers to probabilistic polynomial time.

Definition 1 (Simulation Security). *A cryptographic protocol is* simulation secure *if, for all PPT adversaries A in the real world who actively corrupt any arbitrary subset of users and mix-nodes in the anonymous communication network, there exists a simulator S in the ideal world execution, which corrupts the same set of parties and produces an output computationally indistinguishable to the output of A in the real world.*

In Appendix A we show that, with a CPA-secure threshold group-homomorphic encryption scheme and a perfectly hiding commitment scheme, the core cMix protocol securely realizes the ideal world presented in the previous subsection. More formally,

Theorem 1 (Simulation Security). *If $\mathcal{E}$ is a secure threshold group-homomorphic encryption scheme and* $(\mathsf{Commit}, \mathsf{Open})$ *is a non-Interactive Commitment Scheme, and assuming that each pair of user and mix-node has agreed upon a long-term master key, then the cMix protocol is simulation secure as defined in Definition 1 in the random oracle model.*

6.2 Anonymity Analysis

The cMix protocol ensures sender anonymity. *Sender anonymity* holds if all senders of a single round form an anonymity set within which they are indistinguishable from all other potential senders. This holds for both forward and return messages: cMix ensures that the user who initiated communication will remain anonymous. The notion of sender anonymity was initially formulated in [42] and formalized in [3].

We use the following framework to define sender anonymity. Let the Challenger $Ch(b)$ receive inputs from an adversary specified by the function α_{SA} (see Fig. 3). The message that Ch receives from the adversary is forwarded to $F_{\text{c}Mix}$ in place of the user who is selected by the challenge bit b. Another user, selected by the adversary for Ch, sends a random message. Senders and recipients are simulated by the environment, which lets them pick communication partners and messages at random. Let the event that an adversary compromised n nodes be E_α. The goal of this section is to demonstrate that an adversary can break sender anonymity of at least two honest users only if she compromised all nodes in cMix.

Definition 2. *F_{cMix} provides (σ)-sender anonymity if for the function α_{SA} as defined in Fig. 3 for any adversary A with $0 \leq \sigma \leq 1$, $Pr[0 = \langle A|Ch(0)\rangle] \geq Pr[0 = \langle A|Ch(1)\rangle] + \sigma$, where $\sigma = Pr[E_\alpha]$*

```
function $\alpha_{SA}$(s, (Sender_0, Recip, M), (Sender_1, _, _), b)
    if $s \neq$ fresh challenge then
        output $\perp$
    else
        output (Sender_b, Recip, M, challenge over)
```

Figure 3: Sender anonymity function [3].

Assume E_α did not happen, but an adversary compromised a maximum of $n-1$ nodes. Let us consider the forward round. From a message M sent to F_{cMix}, A learns only the sender identity as defined in C_I and the position of the message. From messages sent between any of the submachines in the IF, A learns both the sender and recipient. By invoking Receiving(*corrupt*, N_i), A compromises nodes. From any compromised node, A learns the permutation he applies to the incoming messages, but not the messages themselves. When corrupted nodes perform a precomputation or real-time phase, invoked with the message (*precomp*, *flag*, *r*), they forward all the messages they receive to A. However, the content of messages sent by users is never forwarded to A and is accessed by nodes using shared memory. For any message sent from the IF to the recipient, A learns the recipient identity, as well as the content of the message.

A can see that both users he selected for Ch are sending. He can also see that $Recipient_0$ is receiving the message M_0. Since A compromised $n-1$ nodes, he learns all but one of the permutations applied on the messages. A can calculate which output slot of the honest node contains the message M_0. He can also calculate which input slots of the honest node contain the (unknown) messages of $Sender_0$ and $Sender_1$.

Since the permutation is random, A has probability of $1/2$ to chose one the two senders correctly regardless of the value of b. Thus, $Pr[0 = \langle A|Ch(0)\rangle|\neg Pr[E_\alpha]] = Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]]$.

Using similar arguments for the return round, one can show that the same equation holds for both directions of communication. Furthermore, it can be shown that the equation $Pr[0 = \langle A|Ch(0)\rangle] \geq Pr[0 = \langle A|Ch(1)\rangle] + Pr[E_\alpha]$ holds using the same approach as in [3, p.30].

6.2.1 Analyzing Standard Mixnet Attacks

Because the **K**, **R**, and **S** values are never reused, cMix protects against *reply attacks* [9] and the attacker cannot follow duplicate messages. Section 4.3 gives a defense against *message-tagging* attacks.

Intersection attacks and *statistical disclosure attacks* [9, 18, 22] make use of mix network 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 mix node can be distinguished since they come from different mixes, have different route lengths, etc. 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 mixes [9], 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 permits an adversary to distinguish these paths based on measures such as counting packets [47] and timing communications [17]. These attacks are hard to apply on cMix because cMix permutes message in batches using a fixed cascade of nodes. *Contextual attacks* [44], sometimes referred as *traffic-confirmation attacks* [45] and intersection attacks [8], evaluate the time when particular senders and receivers participate in the protocol, their communication patterns, and how many messages they send and receive. Only unobservable [42] systems protect against this type of attack. cMix aims to reduce the risk of this attack by introducing dummy traffic (see Section 3.1).

7 Special Features and Extensions

This section explains three extensions of cMix: including recipient keys, providing sender authentication to the recipient, and improving message integrity using randomized partial checking. To the best of our knowledge, cMix is the first anonymity system that facilitates end-to-end confidentiality and user authentication without requiring pre-shared keys or PKI among the users.

7.1 Including Recipient Keys

The core system described in Section 4 does not encrypt output messages or their responses. This transparency might be sufficient for some applications, when confidentiality could be added by the sender with end-to-end encryption before sending a message to cMix. Instead, by extending cMix to encrypt output messages, the recipient needs to perform n multiplication to retrieve the message and no computationally more expensive public-key operation. An additional advantage is mitigation of tagging attacks: with recipient encryption, it is no longer possible to distinguish correct and incorrect output messages.

For this new functionality we introduce the following additional notation:

– $l_{i,j}$ and $l'_{i,j}$: secret keys shared between node N_i and the receiving user of slot j used to blind messages and responses, respectively.

– $\mathbf{l}_i$: vector of keys shared between node N_i and users for β slots; $\mathbf{l}_i = (l_{i,1}, l_{i,2}, \dots, l_{i,\beta})$.

– L_j and L'_j: the product of all shared keys for the receiving user of slot j: $L_j = \prod_{i=1}^{n} l_{i,j}$ and $L'_j = \prod_{i=1}^{n} l'_{i,j}$. The user for this slot stores the inverse of these keys, L_j^{-1} and $L_j'^{-1}$, to blind and unblind messages and responses, respectively.

– $\mathbf{L}$, $\mathbf{L}^{-1}$: vectors of products of shared keys for β slots and their inverses, respectively; $\mathbf{L} = (L_1, L_2, \dots, L_\beta)$ and $\mathbf{L}^{-1} = (L_1^{-1}, L_2^{-1}, \dots, L_\beta^{-1})$.

For the forward path, the precomputation phase remains the same. We need only to change Step 3 in the real-time phase: instead of the nodes sending their decryption shares $\mathscr{D}_i(\mathbf{x})$, they send $\mathscr{D}_i(\mathbf{x}) \times \mathbf{l}_i$ to the network handler. As in Step 2 of the return path, the output of the system would then be $\mathbf{\Pi}_n(\mathbf{M}) \times \mathbf{K}$. Unblinding the message is efficient because the recipient needs only to perform one multiplication to retrieve the message.

The return path will change in both the precomputation and real-time phases. It will be symmetric to the modified forward path: all the random values and keys are fresh as before and the reverse permutation is used.

We can apply this modification directly in applications where all the messages go to the same destination, for example, when using it for anonymous search. In other applications, however, we would need to know which keys to use for the recipient. For this second case we need to add additional functionality to the forward path. Assuming the recipient also sends a response, no changes are needed for the return path.

One way to incorporate recipients is to add a "parallel session" that uses fresh values for the random variables, but still uses the same permutations. The output of this parallel session would be the recipient identities. The first two steps in the real-time phase can be performed concurrently, but the third step needs to be done for the parallel session first to retrieve the recipient identities. After the recipient identities are known, all nodes know which l value to use for every slot, and they can perform the third step for the actual messages.

7.2 End-to-end Sender Authentication

In addition to anonymity and end-to-end confidentiality using recipient keys, we can also empower a sender to authenticate herself to the recipient *without* undermining her anonymity to anyone else. Such a sender-initiated authentication message (say, *SourceAuth*) allows the recipient to confirm the sender's identity (possibly a pseudonym) with a high probability, provided at least one mix node remains honest.

Let $\bar{\mathsf{U}}_j$ be a (possibly pseudonymous) identity for a user U_j. We assume that each identity in the system be represented with ℓ bits. For *SourceAuth* messages, we introduce the following additional notation:

– $\bar{\mathsf{u}}_{j_k}$: the k^{th} bit of identity $\bar{\mathsf{U}}_j$.

– $\bar{b}$: a random one-time-use private bit of user U_j.

– b_i: a random one-time-use private bit of node N_i.

– I_k: k^{th} prime number with $k \in [1, \ell]$. We assign a unique prime to every bit position in the identity string.

– A_j: the product of the primes associated with the one-bits in an identity $\bar{\mathsf{U}}_j$; i.e., $A_j = \Pi_{k=1}^{\ell} (I_k)^{\bar{\mathsf{u}}_{j_k}}$.

The forward path precomputation phase remains the same. During the real-time phase, U_j blinds a message $A_j^{\bar{b}+1}$ using her K_j values. Each node N_i then computes the value ${A_j}^{b_i+1} \times r_{i,j}$ and uses it instead of $r_{i,j}$ for U_j's slot during the preprocessing step. The rest of the forward-path real-time phase remains the same as in Section 7.1. After unblinding the message, the recipient receives a message of the form $A_j^{n'}$, where $n+1 \le n' \le 2(n+1)$. The recipient can easily run a divisibility check for the first ℓ primes on the received message and derive the encoded identity $\bar{\mathsf{U}}_j$. Provided the message $A_j^{n'}$ does not exceed the message space, unique factorization will be possible. If each prime factor does not have the same prime power n' such that $n+1 \le n' \le 2(n+1)$, the recipient rejects the message.

As for the other standard messages, identity $\bar{\mathsf{U}}_j$ remains hidden from the mix nodes during the mixing and postprocessing steps. With high probability, it is also not possible for a dishonest node to change the identity $\bar{\mathsf{U}}_j$ to any other random valid ℓ-bit string; it cannot predict the b_i values of honest nodes nor $\bar{b}$ chosen by U_j. Subsequently, a dishonest node cannot guess the correct power of a prime to be added or removed to obtain a valid *SourceAuth* message. Furthermore, it is not possible for a dishonest sender to impersonate some user U_j with help of the network handler: he cannot predict K_j values of U_j, and with very high probability the message received by the recipient will not decode to a valid identity string.

With 2048-bit encryption, and five mix nodes, we can support up to 32-bit identities. Using two *SourceAuth* messages for a single identity supports 64-bit identities.

7.3 Protocol Integrity

cMix satisfies its integrity property if and only if:

1. each message is forwarded unmodified to its recipient, or

2. all nodes learn that the protocol was not performed successfully.

In this work, we focus on the second condition. We propose to use an existing mechanism to achieve integrity: *Randomized Partial Checking* (RPC), introduced in 2002 by Jakobsson, Juels, and Rivest [28]. This technique probabilistically verifies if the outputs of the mixnet correspond to its permuted inputs. Thus, it verifies both the integrity of messages and that the permutations were applied correctly. Additionally, RPC achieves probabilistic *accountability* [35]: It ensures that if the protocol is performed incorrectly, at least some of the attackers are revealed with sufficiently high probability, while honest parties are never blamed.

In RPC, nodes reveal certain information about a (large) part of their input/output pairs, selected by other nodes or by a random oracle. Revealed pairs are verified against previously made commitments. To maintain privacy of users, adjustment nodes are paired with each node belonging to only one such pair. Nodes in a pair reveal their input such that none of the messages can be followed as an input of one server and an output of a second one. In comparison with the original mix protocol of Chaum [13], RPC achieves a more relaxed level of anonymity under the assumption that at least one pair of adjustment mix nodes behaves according to the protocol, as proven by Küsters et al. [35]. Küsters et al. also demonstrate that a RPC mix-network with over a hundred users has an anonymity level close to the ideal mixnet in the presence of malicious-but-cautious attackers. When implementing RPC, one must perform additional verifications to tackle issues in the original protocol described by Khazaei and Wilkstrm [34].

8 Implementations and Benchmarks

We implemented prototype systems in Python for two scenarios. One system provides the core cMix protocol with a single forward and return path. The other system implements a full messaging system, with support for recipient keys as described in Section 7.1. A single web-based client implements the core protocol, and an Android app implements the client side of the messaging system. Each mix node includes a keyserver (to establish shared keys with the users) and a mixnet server (to carry out the precomputations and real-time computations). Commitments are implemented using SHA-256 and the Ed25519 signature scheme. A parallel process on the nodes precomputes the encryptions and the decryption shares.

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

Table 1: Mean timings in seconds of 100 runs of the precomputation and real-time phases for different batch sizes using five mix nodes for the core protocol.

Batch	**Precomputation**	**Real-time**		
size	**Total**	**Forward**	**Return**	**Total**
10	0.54	0.07	0.03	0.10
50	2.19	0.26	0.12	0.38
100	4.14	0.47	0.25	0.73
200	8.21	0.88	0.45	1.33
300	11.94	1.29	0.62	1.92
400	15.81	1.52	0.84	2.36
500	19.51	1.80	1.01	2.81

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

For our experiments we performed 100 precomputations and real-time phases for different batch sizes up to 500 with five mix nodes. We measured elapsed time on the network handler from the time it instructed the nodes to start until it either received a message from all nodes indicating the precomputation finished successfully or it computed the final responses to be sent to the users in the real-time phase. During the precomputation, the network handler does not receive a message at the end of the forward phase, making it hard to measure exact timings of the forward and return path separately. Because the encryptions are computed in a parallel thread, there is also not a clear distinction between the two paths on the individual nodes.

Tables 1 and 2 give timings for selected batch sizes using five mix nodes for the core protocol and the full messaging implementation, respectively. The means of the different phases can be quite a bit higher than typical due to a few executions with very high timings, probably due to external influences such as background processes running on the instances or traffic delays. Still, these timings show the high performance of the system in the real-time phase. The precomputation can easily be accelerated by performing more computations in parallel, whereas for the real-time phase a network connection with low latency would improve the timings. Additional processors would significantly improve the time it takes to compute all necessary encryptions and decryption shares.

9 Discussion

We discuss applications of cMix, integration of cMix into the larger Privategrity system, and future work.

Table 2: Mean timings in seconds of 100 runs of the precomputation and real-time phases for different batch sizes using five mix nodes for the full messaging implementation.

Batch	Precomputation	Real-time		
size	Total	Forward	Return	Total
10	0.92	0.12	0.03	0.15
50	3.79	0.41	0.10	0.51
100	7.39	0.83	0.20	1.02
200	14.54	1.55	0.40	1.95
300	21.66	2.24	0.59	2.83
400	28.70	2.95	0.79	3.74
500	35.87	3.63	0.97	4.60

9.1 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.

9.2 Integrating cMix into Privategrity

We designed and developed cMix as part of a larger system, called *Privategrity*, which provides a variety of anonymity services. This paper, however, presents and analyses cMix independently from Privategrity. In this section we describe some of the cMix applications planned for Privategrity.

Rather than layering services on top of mixing and allowing widely varying payload sizes, PrivaTegrity's novel approach integrates the services directly into its mixing. PrivaTegrity achieves anonymity among all messages sent globally within each one-second time interval.

cMix enables a range of applications. Examples include private message delivery without use of public key and including confidential authentication of the sender to the recipient. So-called *"untraceable return addresses" (URAs)* can be realized and allow establishing a group to which all members of the group can send. Additional applications include payments, photo sharing, anonymous feed following, voting, anonymous surveys, and general credential mechanisms. We plan to provide more details about these applications in subsequent writings.

To demonstrate that cMix can bring anonymous communication to portable smart devices, we are developing an Android application for instant messaging that uses cMix in its back-end. This app aims to offer a user experience that is familiar and easy to use for non-technical users. Though still in development, the app is functional and will soon be released as a closed alpha.

9.3 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 many of the applications described in Section 9.2, and working out strategies for handling dummy messages (see Section 3.1). We would also like to carry out more performance studies.

Second, we plan to explore different 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 begin to work out how to apply key-homomorphic pseudorandom functions [10] and an appropriate additive homomorphic encryption scheme to allow any length message (see Appendix B).

Fourth, we would like to explore possible ways of reusing a precomputation in a secure way.

10 Conclusion

cMix's powerful security model and speedup in real-time computation are very promising. Unlike previous mixnets, cMix enables smartphones to communicate anonymously without slowing computations, draining batteries, and burning up network bandwidth. 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 mix nodes, cMix scales well for deployment with large anonymity sets and large numbers of mix nodes. Even though the adversary may know all senders and receivers in each batch, she cannot link any sender and receiver unless all nodes are compromised. cMix's security model, coupled with its wide range of applications being pursued, holds promise for a new day in anonymous social interaction.

Acknowledgments

We thank the following people for helpful comments: 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., AND WIKSTRÖM, D. Offline/online mixing. In *ICALP 2007* (2007), pp. 484–495.

[2] BACKES, M., GOLDBERG, I., KATE, A., AND MOHAMMADI, E. Provably secure and practical onion routing. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)* (2012).

[3] BACKES, M., KATE, A., MANOHARAN, P., MEISER, S., AND MOHAMMADI, E. AnoA: A framework for analyzing anonymous communication protocols. In *26th Computer Security Foundations Symposium (CSF)* (2013), pp. 163–178. `http://eprint.iacr.org/2014/087`.

[4] BACKES, M., KATE, A., MEISER, S., AND MOHAMMADI, E. (nothing else) MATor(s): Monitoring the anonymity of Tor's path selection. In *Proc. 21th ACM conference on Computer and Communications Security (CCS 2014)* (November 2014).

[5] BACKES, M., KATE, A., AND MOHAMMADI, E. Ace: An efficient key-exchange protocol for onion routing. In *Proc. 11th ACM Workshop on Privacy in the Electronic Society (WPES)* (2012), pp. 55–64.

[6] BANERJEE, A., AND PEIKERT, C. New and improved key-homomorphic pseudorandom functions. *IACR Cryptology ePrint Archive 2014* (2014), 74.

[7] BENALOH, J. Simple verifiable elections. In *Proc. USENIX/Accurate Electronic Voting Technology Workshop (EVT)* (2006), pp. 5–5.

[8] BERTHOLD, O., AND LANGOS, H. *Privacy Enhancing Technologies: Second International Workshop, PET 2002 San Francisco, CA, USA, April 14–15, 2002 Revised Papers*. Springer Berlin Heidelberg, 2003, ch. Dummy Traffic against Long Term Intersection Attacks, pp. 110–128.

[9] BERTHOLD, O., PFITZMANN, A., AND STANDTKE, R. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer Berlin Heidelberg, 2001, ch. The Disadvantages of Free MIX Routes and How to Overcome Them, pp. 30–45.

[10] BONEH, D., LEWI, K., MONTGOMERY, H. W., AND RAGHUNATHAN, A. Key homomorphic PRFs and their applications. In *Advances in Cryptology - CRYPTO 2013* (2013), pp. 410–428.

[11] CAMENISCH, J., AND LYSYANSKAYA, A. A formal treatment of onion routing. In *Advances in Cryptology — CRYPTO* (2005), pp. 169–187.

[12] CASTAGNOS, G., AND LAGUILLAUMIE, F. Linearly homomorphic encryption from $$\mathsf {DDH}$$. In *Topics in Cryptology - CT-RSA 2015* (2015), pp. 487–505.

[13] CHAUM, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 4*, 2 (1981), 84–88.

[14] CHEN, C., ASONI, D. E., BARRERA, D., DANEZIS, G., AND PERRIG, A. HORNET: high-speed onion routing at the network layer. In *Proc. 22nd ACM Conference on Computer and Communications Security* (2015), pp. 1441–1454.

[15] COX, J. Court docs show a university helped FBI bust Silk Road 2, child porn suspects. Motherboard, November 2015. `http://motherboard.vice.com/read/court-docs-show-a-university-helped-fbi-bust-silk-road-2-child-porn-suspects?gbwlbe`.

[16] DANEZIS, G. *Privacy Enhancing Technologies: 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004. Revised Selected Papers*. Springer Berlin Heidelberg, 2005, ch. The Traffic Analysis of Continuous-Time Mixes, pp. 35–50.

[17] DANEZIS, G., DIAZ, C., AND TRONCOSO, C. *Privacy Enhancing Technologies: 7th International Symposium, PET 2007 Ottawa, Canada, June 20-22, 2007 Revised Selected Papers*. Springer Berlin Heidelberg, 2007, ch. Two-Sided Statistical Disclosure Attack, pp. 30–44.

[18] DANEZIS, G., DINGLEDINE, R., AND MATHEWSON, N. Mixminion: Design of a Type III anonymous remailer protocol. In *Proc. 24th IEEE Symposium on Security & Privacy* (2003), pp. 2–15.

[19] DANEZIS, G., AND GOLDBERG, I. Sphinx: A compact and provably secure mix format. In *Proc. 30th IEEE Symposium on Security & Privacy* (2009), pp. 269–282.

[20] DANEZIS, G., AND LAURIE, B. Minx: A simple and efficient anonymous packet format. In *Proc. 3rd ACM Workshop on Privacy in the Electronic Society (WPES)* (2004), pp. 59–65.

[21] DANEZIS, G., AND SERJANTOV, A. *Information Hiding: 6th International Workshop, IH 2004, Toronto, Canada, May 23-25, 2004, Revised Selected Papers*. Springer Berlin Heidelberg, 2005, ch. Statistical Disclosure or Intersection Attacks on Anonymity Systems, pp. 293–308.

[22] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium (USENIX)* (2004), pp. 303–320.

[23] EVANS, N. S., DINGLEDINE, R., AND GROTHOFF, C. A practical congestion attack on tor using long paths. In *Proc. 18th USENIX Security Symposium (USENIX)* (2009), pp. 33–50.

[24] GOLDBERG, I., STEBILA, D., AND USTAOGLU, B. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography* (2012), 1–25.

[25] GOLDSCHLAG, D. M., REED, M. G., AND SYVERSON, P. F. Onion routing. *Commun. ACM 42*, 2 (1999), 39–41.

[26] GULCU, C., AND TSUDIK, G. Mixing email with Babel. In *Proc. of the Network and Distributed System Security Symposium (NDSS '96)* (1996), pp. 2–16.

[27] JAKOBSSON, M., JUELS, A., AND RIVEST, R. L. Making mix nets robust for electronic voting by randomized partial checking. In *Proc. 11th USENIX Security Symposium (USENIX)* (2002), pp. 339–353.

[28] JANSEN, R., TSCHORSCH, F., JOHNSON, A., AND SCHEUERMANN, B. The sniper attack: Anonymously deanonymizing and disabling the Tor network. In *(NDSS'14)* (2014).

[29] JERICHOW, A., MLLER, J., PFITZMANN, A., PFITZMANN, B., AND WAIDNER, M. Real-time mixes: A bandwidth-efficient anonymity protocol. *IEEE Journal on Selected Areas in Communications 16*, 4 (1998), 495–509.

[30] Johnson, A., Wacek, C., Jansen, R., Sherr, M., and Syverson, P. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 20th ACM conference on Computer and Communications Security (CCS 2013)* (November 2013).

[31] Kate, A., and Goldberg, I. Using Sphinx to improve onion routing circuit construction. In *Proc. 14th Conference on Financial Cryptography and Data Security (FC)* (2010), pp. 359–366.

[32] Kate, A., Zaverucha, G. M., and Goldberg, I. Pairing-based onion routing with improved forward secrecy. *ACM Trans. Inf. Syst. Secur. 13*, 4 (2010), 29.

[33] Khazaei, S., and Wikstrm, D. Randomized partial checking revisited. In *Topics in Cryptology: CT-RSA 2013*. 2013, pp. 115–128.

[34] Küsters, R., Truderung, T., and Vogt, A. Formal analysis of chaumian mix nets with randomized partial checking. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy* (2014), SP '14, IEEE Computer Society, pp. 343–358.

[35] Möller, B. Provably secure public-key encryption for length-preserving Chaumian mixes. In *Proc. CT-RSA* (2003), pp. 244–262.

[36] Möller, U., Cottrell, L., Palfrader, P., and Sassaman, L. Mixmaster protocol – Version 2. IETF Internet Draft, 2003.

[37] Murdoch, S. J., and Danezis, G. Low-cost traffic analysis of Tor. In *Proc. 26th IEEE Symposium on Security & Privacy* (2005), pp. 183–195.

[38] Naor, M., Pinkas, B., and Reingold, O. Distributed pseudo-random functions and kdcs. In *Advances in Cryptology–EUROCRYPT'99* (1999), Springer, pp. 327–346.

[39] Øverlier, L., and Syverson, P. Improving efficiency and simplicity of Tor circuit establishment and hidden services. In *Proc. 7th Privacy Enhancing Technologies Symposium (PETS)* (2007), pp. 134–152.

[40] Øverlier, L., and Syverson, P. F. Locating hidden servers. In *Proc. 27th IEEE Symposium on Security & Privacy* (2006), pp. 100–114.

[41] Pfitzmann, A., and Hansen, M. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, Aug. 2010. v0.34.

[42] Pfitzmann, B., and Pfizmann, A. How to break the direct RSA-implementation of mixes. In *Advances in Cryptology — EUROCRYPT '89* (1990), pp. 373–381.

[43] Raymond, J.-F. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*. Springer Berlin Heidelberg, 2001, ch. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems, pp. 10–29.

[44] Reed, M., Syverson, P., and Goldschlag, D. Anonymous connections and onion routing. *Selected Areas in Communications, IEEE Journal on 16*, 4 (May 1998), 482–494.

[45] Serjantov, A., Dingledine, R., and Syverson, P. *Information Hiding: 5th International Workshop, IH 2002 Noordwijkerhout, The Netherlands, October 7-9, 2002 Revised Papers*. Springer Berlin Heidelberg, 2003, ch. From a Trickle to a Flood: Active Attacks on Several Mix Types, pp. 36–52.

[46] Serjantov, A., and Sewell, P. *Computer Security – ESORICS 2003: 8th European Symposium on Research in Computer Security, Gjøvik, Norway, October 13-15, 2003. Proceedings*. Springer Berlin Heidelberg, 2003, ch. Passive Attack Analysis for Connection-Based Anonymity Systems, pp. 116–131.

[47] Shimshock, E., Staats, M., and Hopper, N. Breaking and provably fixing Minx. In *Proc. 8th Privacy Enhancing Technologies Symposium (PETS)* (2008), pp. 99–114.

[48] Sun, Y., Edmundson, A., Vanbever, L., Li, O., Rexford, J., Chiang, M., and Mittal, P. Raptor: Routing attacks on privacy in Tor. In *Proc. 24th USENIX Security Symposium (USENIX))* (2015), pp. 271–286.

[49] The Tor project. `https://www.torproject.org/`, 2003. Accessed Nov 2015.

[50] Wikström, D. A universally composable mix-net. In *Proc. of the 1st Theory of Cryptography Conference (TCC)* (2004), pp. 317–335.

An earlier version of this paper was submitted to the *25th USENIX Security Symposium*. February 23, 2016.

Appendices

A Postponed Security Analysis

In this section, we analyze our protocol using the ideal/real world paradigm. We describe the ideal world, which models the intended behavior of the system, in terms of functionality and privacy. We then argue that our core cMix protocol can be securely abstracted by the ideal world.

A.1 Ideal World

In the ideal world we assume the existence of a trusted third party (TTP). Each mix node in the network is connected to every other node as well as to the TTP via a private authenticated channel. In our ideal functionality, we use the message-based state transitions and consider submachines for all n nodes. To communicate with each other through messages and data structures, these submachines share a memory space in the functionality. Messages are sent using an instruction *Send*. An adversary can observe, delay, or stop the messages going from one node to another, but she cannot read the message contents.

As in the rest of the paper, we denote a user as U_j, $(1 \leq j \leq m)$, cmix nodes as N_i, $(1 \leq i \leq n)$, and M_j denotes a message of the user U_j. An adversary is denoted as A. To obtain a value v stored in a table T under key k, we use the notation $v \leftarrow query(T, key = k)$, while *Update* $T \leftarrow (t)$ describes storing a tuple t in a table T.

Internal data structures.

The ideal functionality maintains the following data structures. A list of incoming messages is stored in B. A list of compromised nodes is maintained in $\mathscr{C}$. The adversary may corrupt some message while it is processed at the compromised nodes by attaching a corrupt tag to the message; however, she cannot check or remove the tag until the message is output by the network handler. A

```
upon An input($setup$) :
    $B = J_u = C = \emptyset$,
    Empty tables $P$ and $T$

upon Receiving ($send$, $U_j$,$M_j$) :
    if $j \notin J_u$ then ▷This user has not yet sent a message
        Set $J_u \leftarrow J_u \cup \{j\}$
        Append $M_j$ to buffer $B$
        if $|B| = \beta$ then ▷the buffer is full
            SENDMSG($\mathscr{F}_{cMix}$, $start$)

upon Receiving($start$) :
    SENDMSG( $\mathscr{F}_{cMix}$, $precomp$)
    Wait to receive($precomp\_finished$)
▷All nodes start (real-time) preprocessing simultaneously
▷For forward direction $dir = 1$, and $dir = -1$ for backword
    SENDMSG($N_1$, $real\text{-}time$, 1)
    Wait to receive($real\text{-}time\_finished$, 1) from $Handler$
    SENDMSG($Handler$, output, $B$)
    Wait to receive($reply$) $B'$ from $Handler$
▷Reply messages have been collected at $Handler$
    Replace the set $B$ with the received set $B'$
    SENDMSG($N_n$, $real\text{-}time$, $-1$)
    Wait to receive($real\text{-}time\_finished$, $-1$) from
$Handler$
    Send replies from $B$ to corresponding $U_j$

upon Receiving($precomp$) :
    SENDMSG($N_1$, $precomp$, 1)
    Wait to receive($output\_precomp$, 1) from $N_n$
    SENDMSG($N_n$, $precomp$, $-1$)
    Wait to receive($output\_precomp$, $-1$) from $N_1$
    SENDMSG( $\mathscr{F}_{cMix}$, $precomp\_finished$)

upon Receiving($compromise$, $N_i/U_j$) from $A$ :
    Set $\mathscr{C} \leftarrow \mathscr{C} \cup \{N_i/U_j\}$

upon Receiving($corrupt$, $N_i$,$j$) from $A$ :
    if $N_i \in \mathscr{C}$ then
        Attach a $corrupt$ tag to $j$th message in $B$ during
the next processing at $N_i$

function SENDMSG($Recipient, header, payload$)
    Send($Sender$, $Recepient$, $header$, $|payload|$) to $A$
    Wait to receive $forward$ from $A$
    Send message $(header, payload)$ to $Recipient$
```

Figure 4: Ideal Functionality for cMix network $\mathscr{F}_{cMix}$.

table of intermediate values stored by nodes and handler: T with tuples $(N_i, phase, direction, party)$, where $party$ indicates who stores the given record. A table P with tuples $(N_i, permutation)$ containing the precomputed permutation of the node with ID N_i.

Ideal functionality. All cMix nodes are a part of the ideal functionality, and thus they have access to appropriate internal data structures of the ideal functionality. Nodes communicate with each other using these data structures and the function SENDMSG$(\cdot,\cdot)$, which triggers the ideal functionality to send messages with the help of communication model. For simplicity the ideal functionality accepts only one input from each user, and encompasses only one round of communication. Figure 4 presents the general ideal functionality in pseudocode; Figure 5 gives pseudocode for cMix node subroutines; and Figure 6 depicts subroutines for the network handler. Unlike the cMix algorithm, $\mathscr{F}_{cMix}$ does not have any cryptographic operations such as encryption, decryption or commitments; the required security properties are instead insured by the the TTP.

As discussed in 3.2, we assume a secure authenticated channel between cMix nodes. Thus, the only influence an attacker has on the messages sent between nodes is to delay or drop them; this is reflected in the SENDMSG$(\cdot,\cdot)$ function. The only information an attacker learns is the sender and recipient of the message, as well as its length. To learn the messages sent and received by nodes, an attacker compromises them. When a node is compromised, it invokes compromised node function that forwards all the messages the node receives to A and waits for instructions from him.

We define below the concept of simulation security, which intuitively captures under which conditions a cryptographic protocol constitutes a secure realization of the ideal world defined above. PPT refers to probabilistic polynomial time.

Definition 3 (Simulation Security)**.** *A cryptographic protocol is* simulation secure *if, for all PPT adversaries A in the real world who actively corrupt any arbitrary subset of users and mix nodes in the anonymous communication network, there exists a simulator S in the ideal world execution, which corrupts the same set of parties and produces an output computationally indistinguishable to the output of A in the real world.*

A.2 Simulation Security

Here, we perform an informal security analysis of the cMix protocol. In particular, we present a proof sketch to demonstrate that the cMix protocol with a CPA-secure threshold group-homomorphic encryption scheme and a perfectly hiding commitment scheme, securely realize the ideal world presented in the previous subsection. More formally,

Theorem 2 (Simulation Security)**.** *If $\mathscr{E}$ is a secure threshold group-homomorphic encryption scheme and* (Commit, Open) *is a non-Interactive Commitment Scheme, and assuming that every pair of user and mix node have agreed upon a longer term master key, then*

```
upon Receiving($phase$, $postproc$, $dir$) :
    Update $T \leftarrow (N_i, phase, dir, Handler)$

upon Receiving($phase\_preproc$, $dir$) :
▷In the preprocess phase all nodes send messages to the handler
    Wait to receive($phase\_preproc$, $dir$) from $N_i$, $\forall i \in [1,n]$
    SendMsg($N_1$, $phase$, $dir$, $mixing$)

upon Receiving($decrypt\_share$, $dir$) :
    Wait to receive($decrypt\_share$, $dir$) from $N_i$, $\forall i \in [1,n]$
    SendMsg($\mathscr{F}_{cMix}$, $real\text{-}time\_finished$, $dir$)
▷Messages are retrieved and are ready to be delivered to recipients

upon Receiving(output, $B$) from $\mathscr{F}_{cMix}$ :
    Forward messages in $B$ to $A$

upon Receiving(return, $B'$) from $A$ :
    SendMsg($\mathscr{F}_{cMix}$, return, $B'$)
```

Figure 6: Subroutines of $\mathscr{F}_{cMix}$ for Handler.

```
upon Receiving($phase$, $dir$) :
▷$phase$ is equal to either real-time or precomp
    if $N_i \in \mathscr{C}$ then
        CompromisedNode($N_i$, $phase$, $dir$)
        return
    if $dir = 1$ then
        SendMsg($Handler$, $phase$, $preproc$, $dir$)
        Wait to receive ($phase$, $dir$, $mixing$)
    if $(dir = 1)$ AND $(phase = precomp)$ then
        Create a random permutation $p_i$
        Update $P \leftarrow (N_i, p_i)$
    else if $phase = real\text{-}time$ then
        $p_i \leftarrow query(P, key = N_i)$
        $B \leftarrow p_i^{dir}(B)$
    if $(i + dir = n + 1)$ OR $(i + dir = 0)$ then ▷If $N_i = N_n$ and $dir = 1$ or $N_i = N_1$ and $dir = -1$
        SendMsg($N_i$, $phase$, $postproc$, $dir$), $\forall i \in [1,n]$
        SendMsg($Handler$, $phase$, $postproc$, $dir$)
    else
        SendMsg($N_{(i+dir)}$, $phase$, $dir$)

upon Receiving($phase$, $postproc$, $dir$) from $N_i$ :
    Update $T \leftarrow (N_i, phase, dir, N_i)$
    if $phase = real\text{-}time$ then
        $v \leftarrow query(T, key = (N_i, precomp, dir, N_i))$
        if $v \neq \perp$ then
            SendMsg($Handler, decrypt\_share, dir$)
    else
        SendMsg($\mathscr{F}_{cMix}$, $output\_precomp$, $dir$)

function CompromisedNode($M$)
    Send $M$ to $A$
    Wait to receive $(N_i', M')$ from $A$
    Send message $M'$ to $N_i'$
```

Figure 5: Subroutines of $\mathscr{F}_{cMix}$ for node N_i.

the cMix protocol is simulation secure as defined in Definition 1 in the random oracle model.

Proof Outline. The general idea of the proof is to provide a set of efficient simulators that run the corrupted instances of the network in the ideal world and to simulate the inputs that those would expect in the real protocol execution.

For every execution, our real as well as ideal worlds are divided in two phases: precomputation phase and real-time phase. These worlds also match in terms of communication flows, and the simulators are left only with the task of correctly realizing the cryptographic messages.

For the precomputation phase, the core step of the proof is to simulate the homomorphic encryption of random R and K, chose random permutations for the corrupted mix node, and then commit the decryption shares. The users are not involved in this step. Importantly, all elements exchanged by the nodes are either commitments or encryptions of random messages. As we require all of these outputs to hide statistically the inputs of the respective protocols, it is easy for a simulator S_{pre} to simulate the correct distribution of the input that the adversary is expecting with random values in the appropriate domain.

Simulating the real-time phase requires a more sophisticated analysis. Here, a simulator S_{real} needs to simulate the protocols for the corrupted users along with corrupted mix nodes. Messages from honest users remain perfectly hidden from the adversary at all parts of the networks, except when they are released to the network handler in the forward direction, and when the responses from the network handler are collected by the exit node.

There are two key challenges. The first challenge is that S_{real} needs to output the adversaries inputs (i.e., receive-message pairs input by the corrupted users) correctly in the end of the forward as well as backward phase. The second challenge is that the adversary may try to tag the simulated messages from the honest users, when they are getting permuted at a corrupted node. In that case, it should not be possible for the adversary to remove the tag at some later stage at another corrupted node.

We solve the first challenge as follows:

– In the forward direction, we open the commitments to the shares such that they match the adversary messages.

– In the backward direction, we achieve this goal by changing the quotients of the S and Ka' values for all honest nodes. Because the expected adversary response messages are already known to S_{real}, it can create the respective versions for those to be collected by the adversary by manipulating its quotient values.

The second challenge is easy to solve because the messages remain perfectly hidden from the adversary until they are decrypted during the open algorithm of the decryption step.

Therefore, using S_{pre} and S_{real}, it is possible to simulate the responses expected by the adversary. It is also easy to see that both S_{pre} and S_{real} are efficient because they can complete their tasks by simulating decryption with help of commitments in the forward direction and re-randomization (i.e., quotients of S and Ka') in the backward direction. $\square$

B Allowing Larger Messages

Our core cMix protocol only allows messages representable as elements of the multiplicative group $\mathbf{G}$. In this section, we present an exposition of the idea to overcome this limitation.

Our idea necessitates use of a key-homomorphic PRF [6,10,39] along with a group homomorphic encryption scheme.

Key Homomorphic PRF. A PRF $\mathscr{F} : \mathscr{K} \times \mathscr{X} \to \mathscr{Y}$ is called key homomorphic if $\mathscr{F}(k_1,x) \otimes \mathscr{F}(k_2,x) = \mathscr{F}(k_1 \oplus k_2,x)$, where $\otimes$ and $\oplus$ are the group operations defined in domains $\mathscr{Y}$ and $\mathscr{K}$, respectively.

Additively homomorphic encryption. Let $\mathsf{E}_{pk}(m_1)$ and $\mathsf{E}_{pk}(m_2)$ denote encryptions of m_1 and m_2 under some public key pk, respectively. The encryption scheme is homomorphic if $\mathsf{E}_{pk}(m_1) \otimes \mathsf{E}_{pk}(m_2) = \mathsf{E}_{pk}(m_1 \oplus m_2)$ for some operations $\otimes$ and $\oplus$ defined in respective domains (groups). The scheme is additively (multiplicatively) homomorphic if $\oplus$ represents the addition (multiplication) operation.

All known key homomorphic PRF constructions in the literature have been *additive* homomorphic in nature [6,10,39]. Therefore, we have to consider additively homomorphic encryption scheme [12] here, to match with the domain of the key of key-homomorphic PRF.

B.1 Construction

Here we specify the required modifications to cMix to handle larger messages. ElGamal encryption employed in the main body of the paper is multiplicatively homomorphic, and it has to be replaced by an additively homomorphic encryption [12].

The precomputation phase will require the following changes :

- In the preprocessing step, instead of sending $\mathscr{E}(\mathbf{r}_i^{-1})$, each node sends $\mathsf{E}_{pk}(-\mathbf{r}_i)$ to the first node. At the end of the step, the first node receives $\otimes_{i=1}^{n} E_{pk}(-\mathbf{r}_i) = E_{pk}(\oplus_{i=1}^{n}(-\mathbf{r}_i)) = E_{pk}(-\mathbf{R})$ (say), instead of $\mathsf{E}_{pk}(\otimes_{i=1}^{n}(\mathbf{r}_i^{-1}))$ from the network handler.

- In the mixing step, each node multiplies with $\mathsf{E}_{pk}(-\mathbf{s}_i)$, instead of $\mathscr{E}(\mathbf{s}_i^{-1})$. Therefore, at the end the step, the last node have $E_{pk}(\Pi(-\mathbf{R}) \oplus (-\mathbf{S}))$, Π denotes the aggregate permutation function used by cMix nodes. Similar to the original protocol, this is partially decrypted to get shares of $(\Pi(-\mathbf{R}) \oplus (-\mathbf{S})) = \mathbf{D}$(say).

Now if we have a huge message, we can divide the message into multiple segments M_s , where s is the segment id. For each segment, a randomly generated tag value, t, is used (alternatively, it can be the segment id).

In Real-time phase, instead of multiplying with the $\mathbf{r}_i$ and $\mathbf{s}_i$ values each node multiplies with $\mathscr{F}(\mathbf{r}_i,t)$ and $\mathscr{F}(\mathbf{s}_i,t)$ values respectively. So, after Mixing step the last node will have $M_s \otimes \mathscr{F}(\Pi(\mathbf{R}) \oplus \mathbf{S},t)$, where t is the tag corresponding to segment M_s . After the last node has received all the segments, the shares are reconstructed to obtain $\mathbf{D}$, and then function $\mathscr{F}(\mathbf{D},t)$ is computed for each t . The segments are retrieved by computing $M_s \otimes \mathscr{F}(\Pi(\mathbf{R}) \oplus \mathbf{S},t) \otimes \mathscr{F}(\mathbf{D},t)$. The last node combines the segments to reconstruct the whole message, and sends the message to the receiver.

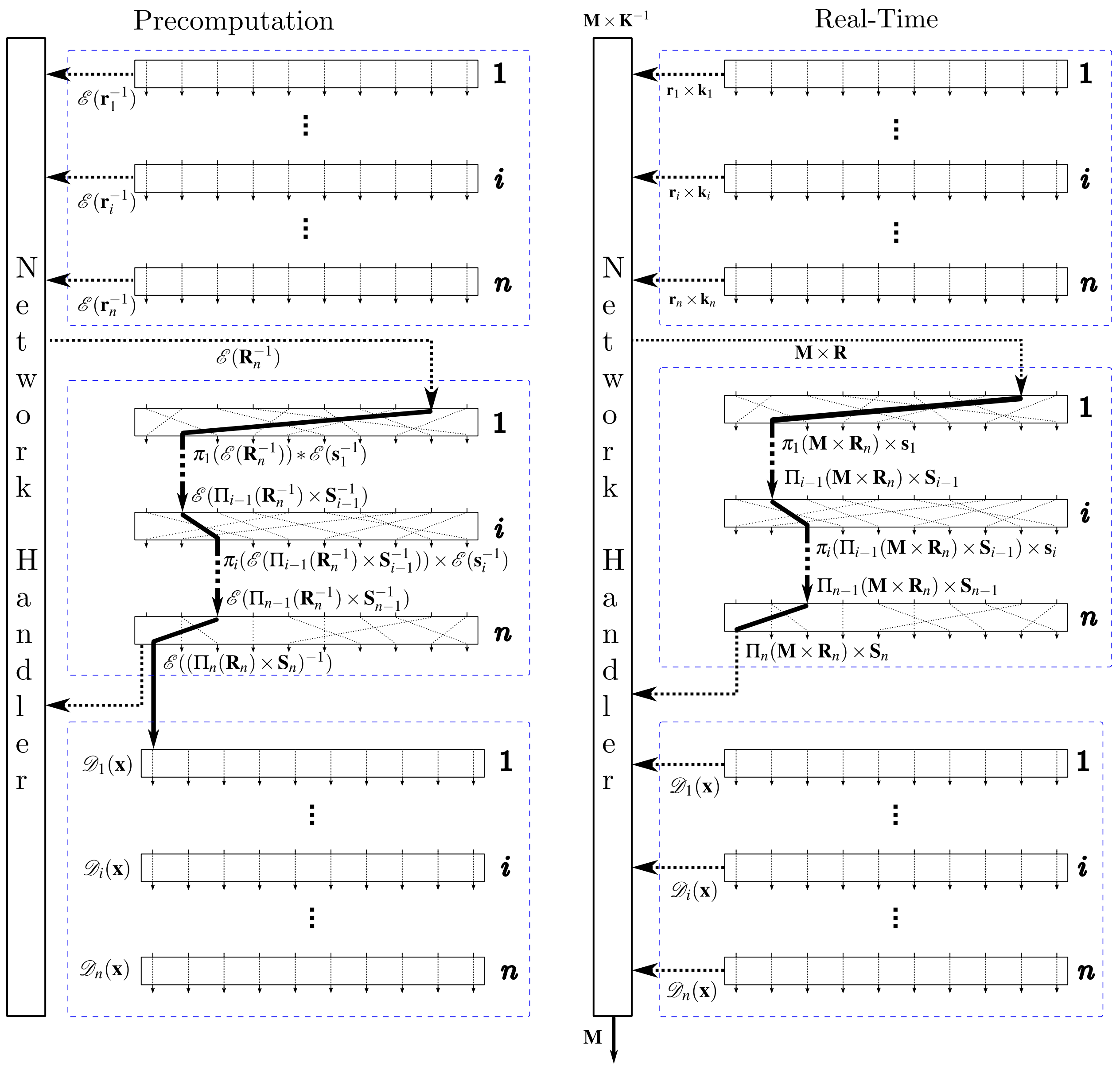

Precomputation
Real-Time
$\mathbf{M} \times \mathbf{K}^{-1}$
Network Handler
Network Handler
$\mathscr{E}(\mathbf{r}_1^{-1})$
$\mathscr{E}(\mathbf{r}_i^{-1})$
$\mathscr{E}(\mathbf{r}_n^{-1})$
$\mathscr{E}(\mathbf{R}_n^{-1})$
$\pi_1(\mathscr{E}(\mathbf{R}_n^{-1})) * \mathscr{E}(\mathbf{s}_1^{-1})$
$\mathscr{E}(\Pi_{i-1}(\mathbf{R}_n^{-1}) \times \mathbf{S}_{i-1}^{-1})$
$\pi_i(\mathscr{E}(\Pi_{i-1}(\mathbf{R}_n^{-1}) \times \mathbf{S}_{i-1}^{-1})) \times \mathscr{E}(\mathbf{s}_i^{-1})$
$\mathscr{E}(\Pi_{n-1}(\mathbf{R}_n^{-1}) \times \mathbf{S}_{n-1}^{-1})$
$\mathscr{E}((\Pi_n(\mathbf{R}_n) \times \mathbf{S}_n)^{-1})$
$\mathscr{D}_1(\mathbf{x})$
$\mathscr{D}_i(\mathbf{x})$
$\mathscr{D}_n(\mathbf{x})$
$\mathbf{r}_1 \times \mathbf{k}_1$
$\mathbf{r}_i \times \mathbf{k}_i$
$\mathbf{r}_n \times \mathbf{k}_n$
$\mathbf{M} \times \mathbf{R}$
$\pi_1(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{s}_1$
$\Pi_{i-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{i-1}$
$\pi_i(\Pi_{i-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{i-1}) \times \mathbf{s}_i$
$\Pi_{n-1}(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_{n-1}$
$\Pi_n(\mathbf{M} \times \mathbf{R}_n) \times \mathbf{S}_n$
$\mathscr{D}_1(\mathbf{x})$
$\mathscr{D}_i(\mathbf{x})$
$\mathscr{D}_n(\mathbf{x})$
$\mathbf{M}$
1
i
n

Figure 7: The cMix protocol: precomputation and real-time computation (forward paths).