cMix: Anonymization by High-Performance Scalable Mixing

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

Farid Javani
Cyber Defense Lab, UMBC, USA
javani1@umbc.edu

Aniket Kate
Purdue University, USA
aniket@purdue.edu

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

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

Alan T. Sherman
Cyber Defense Lab, UMBC, USA
sherman@umbc.edu

***Abstract*—cMix is a cryptographic protocol for mix networks that uses precomputations of a group-homomorphic encryption function to avoid *all* real-time public-key operations by the senders, mix nodes, and receivers. Like other mix network protocols, cMix can enable an anonymity service that accepts inputs from senders and delivers them to an output buffer, in a way that the outputs are unlinkable to the inputs. cMix's high-performance scalable architecture, which results from its unique pre-computation approach, makes it suitable for smartphone-to-smartphone use while maintaining full anonymity sets independently per round.**

Each sender establishes a shared key separately with each of the mix nodes, which is used as a seed to a cryptographic pseudorandom number generator to generate a sequence of message keys. Each sender encrypts its input to cMix with modular multiplication by message keys. cMix works by replacing the message keys, which are not known in the pre-computation, in real time with a precomputed random value.

Our presentation includes a detailed specification of cMix and simulation-based security arguments. We also give performance analysis, both modelled and measured, of our working prototype currently running in the cloud.

cMix is the core technology underlying our larger PrivaTegrity system that allows smart devices to carry out a variety of applications anonymously (including sending and receiving chat messages), with little extra bandwidth or battery usage. This paper focuses on cMix.

Keywords. Anonymous communications, mix networks, cMix, group-homomorphic encryption, PrivaTegrity.

I. Introduction

Untraceable (anonymous) and unlinkable communication is 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 Tor [39] *onion routing* network.

The Tor network, however, is susceptible to a variety of traffic-analysis attacks [16], [21], [30], [38], based in part on Tor's non-uniform message size and timing. Recent anonymity analyses [4], [23] raise doubts on the quality of anonymity possible using so-called onion routing. By contrast, *mixing networks* (also called mixnets) [9], [13], [14], [19], [22], [37], [41] 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 fast cryptographic protocol for mix networks. cMix is an enabling technology that can support a variety of anonymity services, including sending and receiving anonymous chat messages. Using precomputation, cMix avoids all expensive real-time public-key operations by senders, mix nodes, and receivers. cMix's fast performance and key management makes it highly scalable for deployment with large anonymity sets and large numbers of mix nodes.

In cMix, each sender choosing to participate in a particular round sends an input to the cMix system, which after passing through a cascade of mix nodes, arrives in an output buffer. Unless all mix nodes collude, the outputs are unlinkable to the inputs.

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 is used as a seed into a cryptographic pseudorandom number generator to produce a sequence of message keys, the next key in the sequence being selected when the sender chooses to participate in a particular round. Each sender encrypts its input with modular multiplications by a 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. By freeing mix nodes from performing expensive public-key operations in real time, real-time mixing is much faster than in previous mix networks, and the nodes can achieve a higher utilization of their hardware per message batch. 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 "network handler," who will arrange the arriving inputs into batches. As is typically true for mix networks, receivers do not necessarily establish shared keys with the mix nodes, when the network handler can send the outputs from the output buffer to the final receivers. The unlinkability of inputs to outputs does not depend on the correct operation of the network handler, who does not know any message key.

This paper focuses on the cMix protocol, which is the core enabling technology of our larger *PrivaTegrity* system. PrivaTegrity is an overlay system that allows smart phones, laptops, and other devices to carry out a variety of applications anonymously, with little extra bandwidth or battery usage. Use cases include chat, photo/video sharing, feed following, searching, posting, payments, each with various types of potentially pseudonymous authentication. 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. It includes what aims to be a comprehensive range of lightweight services efficiently supporting the aforementioned use cases, bringing them into the same anonymity sets as those for chat messages.

PrivaTegrity achieves anonymity among all messages sent globally within each one-second time interval. To learn anything about which inputs correspond to which outputs within any such batch of messages, the entire cascade of ten mix servers, each preferably operating independently in a different country, would have to be compromised. By integrating these use cases into the mixing with standardized formats and uniform payload sizes, PrivaTegrity enjoys strong anonymity properties and mitigates recent attacks on mix networks.

Our contributions include:

1) A new fast scalable cryptographic mixing protocol, cMix, based on precomputation.
2) Simulation-based security arguments for cMix.
3) Performance analysis of the cMix protocol based on modelling and on benchmarks from our implementation running in the cloud.
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 their receivers.

II. Background and Related Work

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

A. Mix Networks

In 1981, Chaum [9] 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 individual message through the network difficult. Chaum's paper described both batched and unbatched versions of mixing. In more than three decades of research on mixnets, many mix network designs have been proposed including [13], [14], [19], [22], [37], [41], and a few have implemented [12], [29].

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. In this work, we introduce a novel mixnet architecture, cMix, that allows us to reduce the computation overhead by replacing real-time public-key operations with symmetric-key operations.

Some early mixing protocols [9], [14] were based on heuristic security arguments, and weaknesses have been discovered with them [35], [37]. By contrast, most of the recent mixing formats [8], [13], [28], [35] 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.

B. 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], [8], [10], [17], [24], [25], [31], and some have been extensively employed in practice [15], [39].

Common to many of them is *onion routing* [18], [32], 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 [16], [21], [30], [38], such as Tor [39], 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 [11] 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 unlike in onion routing, users do not perform anonymous key agreement [5], [15], [17], [25] 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.

III. 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 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 operations, enabling all real-time computations to be carried using only fast multiplications.

A. Communication Model

Let m be the number of users of the cMix system, which includes of a sequence of n mix nodes $N_1, N_2, \ldots, N_n$. Each of the nodes 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 VI). 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

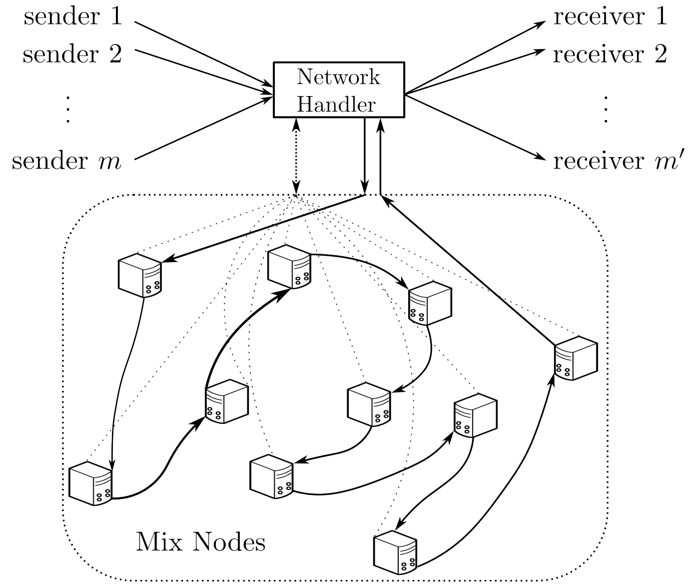

Fig. 1: The cMix communication model.

message of the request delivery sent by the last mix node. All the messages transmitted during one sub-round have the same length and are processed simultaneously.

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 network 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.

B. 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.

C. 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 $MK_{i,j}$ 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).

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 every node N_i. Specifically, the next message key $ka_{i,j}$ is the next output of a pseudorandom number generator with seed $MK_{i,j}$. To encrypt a message, the user first computes a composite key using the derived message keys: $Ka_j = \Pi_{i=1}^{n} ka_{i,j}$. Then she can encrypt her first message M_1 as $M_1 \times Ka_j^{-1}$.

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. The system achieves unlinkability of messages if at least one node carries out its permutation honestly.

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.

D. 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. Related Figure 3 fills in many of the details for each step.

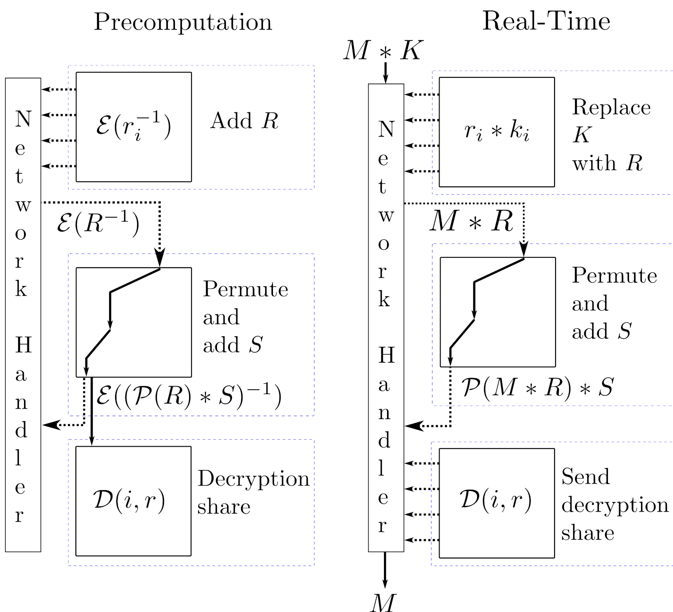

Fig. 2: Overview of the cMix protocol (forward paths).

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 ${r_i}^{-1}$ of the inverses of these values and sends the resulting vector $\mathcal{E}({r_i}^{-1})$ to the network handler. The network handler, exploiting the group homomorphic property of $\mathcal{E}$, computes the (component-wise) direct product $\mathcal{E}(R^{-1})$ 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 p_i. It also multiples in another vector of random values ${s_i}^{-1}$. The result at the exit of the last node is $\mathcal{E}((P(R) \times S)^{-1})$, where P is the vector of component-wise compositions of the p_i's, and S is the direct product of the s_i's. This result is sent both to the network handler and to the first mix node.

In the third step of the precomputation, each node N_i computes its decryption share $D(i, r)$ 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 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 ka_i with its vector of random values 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 $M \times Ka^{-1}$ to $M \times R$. Here, Ka is the direct product of the ka_i's, and R is the direct product of the r_i's.

In the second step of the real-time phase, each node i in order permutes its message buffer with its permutation p_i and multiples in its vector of random values s_i. The result at the exit of the last mix node is $P(M \times R) \times S$. 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 $D(i, r)$, 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.

IV. The cMix Protocol

A. Preliminaries

Nodes in cMix are denoted as N_i with $i = 1, 2, \ldots, n$. 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 when sending a message to the system.

All computation 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 $\mathbf{G}$.

A multi-party group-homomorphic cryptographic scheme based on ElGamal is used, which is also described in [6]. Every node N_i in the scheme holds a share $SK_i \in \mathbb{Z}_p^*$ of the secret key. The public key PK of the scheme can be computed using the secret shares: $PK = \Pi_i g^{SK_i}$. Encryption of message m using this scheme is done using regular ElGamal: $(g^x, m \times PK^x)$, for $x \in_r \mathbb{Z}_p^*$. We call g^x the *random component* and $m \times PK^x$ the *message component* of the ciphertext. To decrypt a ciphertext $(g^x, m \times PK^x)$, all parties need to cooperate. Every node N_i computes a so-called *decryption share* from the decryption component of the ciphertext: $(g^x)^{-SK_i}$. The original message is then retrieved by multiplying all the decryption shares with the message component: $m = m \times PK^x \times \Pi_{i=1}^n (g^x)^{-SK_i}$.

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

- SK_i: the share of node N_i of the secret key SK.
- PK: the public key of the system is based on the nodes' shares of the secret key.
- $\mathcal{E}(m)$: ElGamal encryption under the system's public key. When applying encryption on a vector of values, this means every value in the vector is encrypted individually and the result is a vector of ciphertexts.
- $\mathcal{D}(i, x)$: the decryption share for node N_i using its share of the secret key. x is the random component of a ciphertext. As with encryption, applying this function on a vector of random values results in a vector of corresponding decryption shares.
- p_i: a random permutation of the β slots used by node N_i. The inverse of the permutation is denoted by p_i^{-1}.
- $\mathcal{P}_i(a)$: the permutation performed by the mixnet up to node N_i, i.e., all individual permutations combined:

$$\mathcal{P}_i(a) = \begin{cases} p_1(a) & i = 1 \\ p_i(\mathcal{P}_{i-1}(a)) & 1 < i \leq n. \end{cases}$$

- $\mathcal{P}'_i(a)$: the inverse permutation of slots performed by the mixnet for the return path up to node N_i:

$$\mathcal{P}'_i(a) = \begin{cases} p_n^{-1}(a) & i = n \\ p_i^{-1}(\mathcal{P}'_{i-1}(a)) & 1 \leq i < n. \end{cases}$$

- $ka_{i,j}, ka'_{i,j} \in \mathbf{G}^*$: 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.
- $Ka_j, Ka'_j \in \mathbf{G}^*$: the product of all shared keys for the sending user of slot j: $Ka_j = \Pi_{i=1}^n ka_{i,j}$ and $Ka'_j = \Pi_{i=1}^n ka'_{i,j}$. The user for this slot stores the inverse of these keys, Ka_j^{-1} and $Ka_j'^{-1}$, to blind and unblind messages and responses respectively.
- $M_j, M'_j \in G^*$: the message and the response sent by user U_j in the forward and return phase respectively. Like other values in the system this is a group element, but can easily be converted from, for example, an ASCII encoded string. The size of the group determines the length of the individual messages that can be sent.

For the forward path we have the following values additional:

- $r_{i,a}, s_{i,a} \in \mathbf{G}^*$: random values of node N_i for slots a. Thus, $r_i = (r_{i,1}, r_{i,2}, \ldots, r_{i,\beta})$ is a vector of random values for the β slots in the message map at node N_i. Similarly, s_i is also a vector of random values for node N_i.
- R: the (direct) product of all local random values, i.e., $R_i = \Pi_{j=1}^i r_j$.
- S: the product and permutation of all local random s values:

$$S_i = \begin{cases} s_i & i = 1 \\ p_i(S_{i-1}) \times s_i & 1 < i \leq n. \end{cases}$$

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.
- S': the product and permutation of all local random s' values:

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

We introduce an additional entity called the *network handler* that performs non-sensitive computations, such as computing the product of values output by nodes. In practice this network handler role can also be performed by any of the nodes. The computations of the network handler can also be replaced by a pass through the mixnet, where every node multiplies its local value with the value it received from the previous node. This would balance the computational load over the nodes, but might increase the latency as the values need to be forwarded after every local computation, whereas with the network handler it can start computing when it receives the first values and keep processing while the values from the other nodes come in.

B. At a Glance

cMix comprises two phases: precomputation and real-time. Only in the precomputation phase are public-key cryptographic operations performed. For the real-time phase, only multiplications need to be computed. Below we discuss the protocol at a high level. In Section IV-C, we discuss the protocol in more detail.

In the precomputation phase the nodes and the network handler compute values together that are used in the real-time phase for the forward and return paths. The values used in the forward path are computed in three steps:

1) *Preprocessing*: Each node generates random values r for each slot and encrypts their inverse using the group-homomorphic encryption function $\mathcal{E}$. The nodes send their encrypted r^{-1} values to the network handler, who computes the product of these encryptions. This computation yields the vector of encrypted values $\mathcal{E}(R_n^{-1})$.
2) *Mixing*: The result from the previous step is permuted by each node, which also multiplies in the inverse of its random s values. This computation yields $\mathcal{E}(P_n(R_n^{-1}) \times S_n^{-1})$.
3) *Postprocessing*: In the last step, each node computes their respective decryption shares of the result from the previous step.

For the return path, the values are computed in the reverse order and without the first step: first $\mathcal{E}(S_1'^{-1})$ is computed using

the inverse permutations, after which the nodes compute their decryption shares.

For the real-time phase, the forward path is again performed in three steps:

1) *Preprocessing*: The network handler receives messages M blinded by the inverse of the users' keys: $M \times Ka^{-1}$. In the first step, each node N_i sends the values $ka_i \times r_i$ to the network handler. The network handler uses these values take out Ka^{-1} from the blinded messages and replace it by R_n.
2) *Mixing*: Each node permutes the result of the previous step and multiplies in its s value, computing $P_n(M \times R_n) \times S_n$.
3) *Postprocessing*: Finally, the nodes send their precomputed decryption shares to the network handler who uses these to retrieve the permuted original messages.

As with the precomputation, for the return path we follow the steps in reverse and without the first step. This results in the unpermuted responses, blinded with the users' receiving keys.

C. Detailed Description

Below we discuss the different phases of the protocol in detail, including messages exchanged by the nodes.

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 $\mathcal{E}((\mathcal{P}_n(R_n)\times S_n)^{-1})$ and $\mathcal{E}(S_1'^{-1})$ 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 $\mathcal{E}(R_n^{-1})$ by sending the encryption of their local r_i to the network handler. The network handler computes the product of all the encryptions of the individual values to get the output of this step: $\mathcal{E}(R_n^{-1}) = \Pi_{i=1}^{n}\mathcal{E}(r_i^{-1})$. This output is then sent to the first node as input for the second step.

Forward - Step 2 (mixing). In this step, the nodes exchange the following messages:

$node\ N_i \longrightarrow node\ N_{i+1}:$

$$\mathcal{E}(\mathcal{P}_i(R_n^{-1}) \times S_i^{-1}) = \begin{cases} p_1(\mathcal{E}(R_n^{-1})) \times \mathcal{E}(s_1^{-1}) & i = 1 \\ p_i(\mathcal{E}(\mathcal{P}_{i-1}(R_n^{-1}) \times S_{i-1}^{-1})) \times \mathcal{E}(s_i^{-1}) & 1 < i < n. \end{cases}$$

The last node computes $\mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1}) = p_n(\mathcal{E}(\mathcal{P}_{n-1}(R_n^{-1}) \times S_{n-1}^{-1})) \times \mathcal{E}(s_n^{-1})$. It sends the vector of random components from the ciphertexts of $\mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1})$ to the nodes and stores the message components of $\mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1})$ locally for use in the real-time phase.

Forward - Step 3 (postprocessing). Using the random components received, all the nodes compute their individual decryption shares for $\mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1})$ and store them locally for use in the real-time phase. They publish commitments to their decryption shares, which is necessary to prevent a tagging attack in the real-time phase as will be discussed in Section IV-D.

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

Return - Step 1 (mixing). The nodes compute $\mathcal{E}(S_1'^{-1})$ by sending the following messages:

$node\ N_i \longrightarrow node\ N_{i-1}:$

$$\mathcal{E}(S_i'^{-1}) = \begin{cases} \mathcal{E}(s_n'^{-1}) & i = n \\ p_i^{-1}(\mathcal{E}(S_{i+1}'^{-1})) \times \mathcal{E}(s_i'^{-1}) & 1 < i < n. \end{cases}$$

The first node now computes $\mathcal{E}(S_1'^{-1}) = p_1^{-1}(\mathcal{E}(S_2'^{-1})) \times \mathcal{E}(s_1'^{-1})$. The random components are sent to the other nodes and the message components stored locally for use in the real-time phase.

Return - Step 2 (postprocessing). Like for the forward path all the nodes compute their individual decryption shares for $\mathcal{E}(S_1'^{-1})$ and store them locally for use in the real-time phase.

2) Real-time Phase: For the real-time phase the users construct their message for slot j by taking their message M_j and multiplying it with the inverse of their combined shared key Ka_j to compute the blinded message $M_j \times Ka_j^{-1}$, which they send to the network handler. The network handler collects all the individual messages and combines these to get the vector $M \times Ka$.

Forward - Step 1 (preprocessing). In the first step of the forward path, the Ka^{-1} values are replaced by the r values of each node. Every node N_i sends the values $ka_i \times r_i$ to the network handler. The network handler uses the these values to compute $M \times R_n = M \times Ka^{-1} \times \Pi_{i=1}^{n} ka_i \times r_i$. The result is sent to the first node as input to the next step.

Forward - Step 2 (mixing). In the next step the messages are mixed:

$node\ N_i \longrightarrow node\ N_{i+1}:$

$$\mathcal{P}_i(M \times R_n) \times S_i = \begin{cases} p_1(M \times R_n) \times s_1 & i = 1 \\ p_i(\mathcal{P}_{i-1}(M \times R_n) \times S_{i-1}) \times s_i & 1 < i < n \end{cases}$$

The last node computes $\mathcal{P}_n(M \times R_n) \times S_n = p_n(\mathcal{P}_{n-1}(M \times R_n) \times S_{n-1}) \times s_n$ and computes a commitment to this value. This commitment is then sent to all the other nodes.

Forward - Step 3 (postprocessing). Upon receiving the commitment from the last node, every other node N_i sends its precomputed decryption share for $(x, c) = \mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1})$ to the network handler. The last node n sends the multiplication of the result from the previous step with its decryption share and the message component from the precomputation phase: $\mathcal{P}_n(M \times R_n) \times S_n \times \mathcal{D}(n, x) \times c$. The network

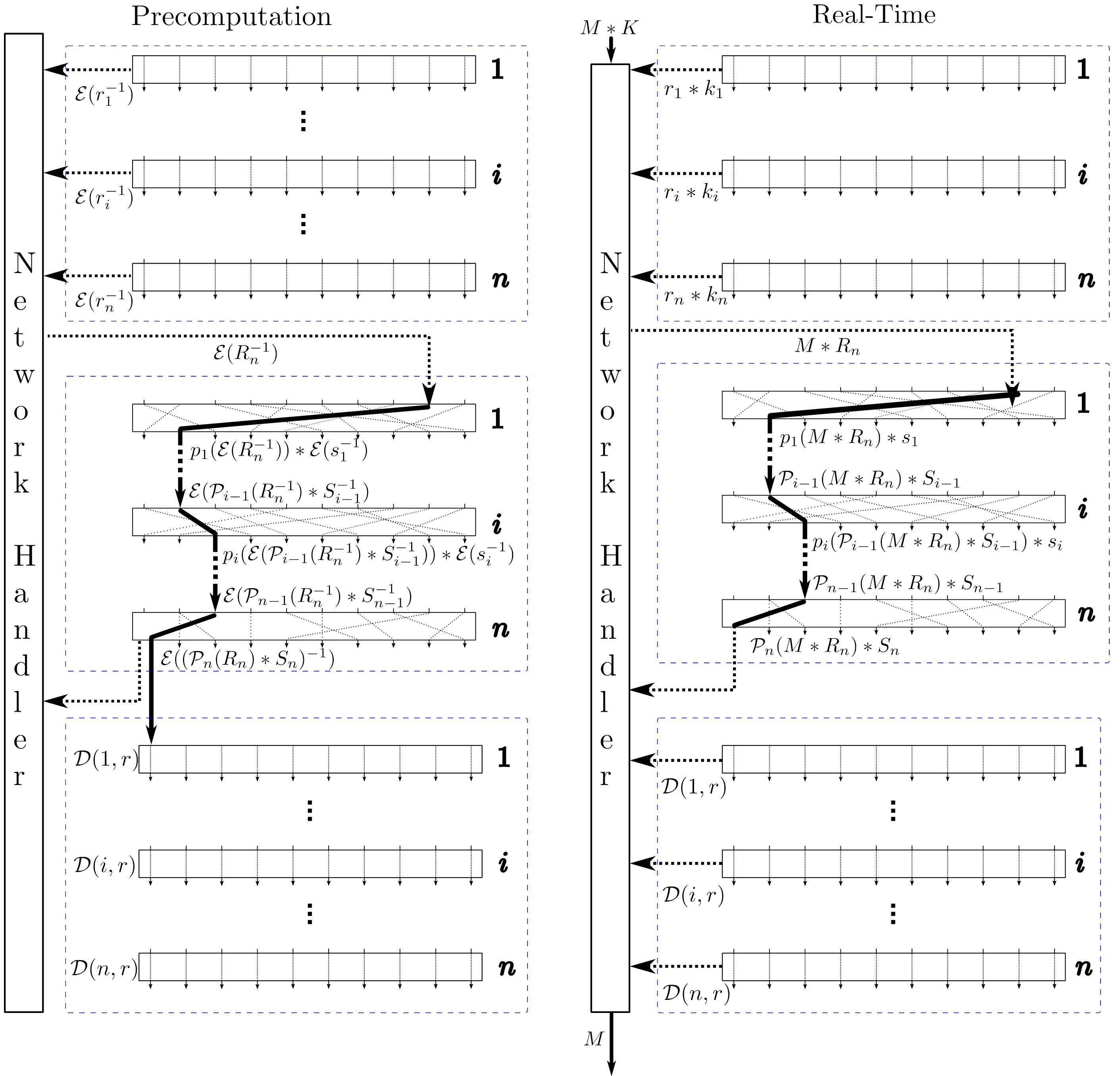

Fig. 3: The cMix protocol: precomputation and real-time computation (forward paths).

handler uses the decryption shares to decrypt the precomputed $\mathcal{E}((\mathcal{P}_n(R_n) \times S_n)^{-1})$ and retrieves the permuted messages:

$$\begin{aligned} &\mathcal{P}_n(M \times R_n) \times S_n \times \Pi_{i=1}^{n} \mathcal{D}(i, x) \times c \\ &= \mathcal{P}_n(M \times R_n) \times S_n \times (\mathcal{P}_n(R_n) \times S_n)^{-1} \\ &= \mathcal{P}_n(M) \end{aligned}$$

The messages are published or delivered and the responses to these messages are collected in M'.

Return - Step 1 (mixing). For the return path we start with the reversed permutations:

$node\ N_i \longrightarrow node\ N_{i-1}:$

$$\mathcal{P}'_i(M') \times S'_i = \begin{cases} p_n^{-1}(M') \times s'_n & i = n \\ p_i^{-1}(\mathcal{P}'_{i+1}(M') \times S'_{i-1}) \times s'_i & 1 < i < n. \end{cases}$$

The first node $\mathcal{P}'_1(M') \times S'_1 = p_1^{-1}(\mathcal{P}'_2(M') \times S'_2) \times s'_1$ and commits to this value the same way 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 $\mathcal{D}(i, x)$

for $(x, c) = \mathcal{E}(S_1'^{-1})$ and uses it to compute $\mathcal{D}(i, x) \times ka_i'$. The first node 1 sends its decryption share multiplied with the message component from the precomputation phase to the network handler: $\mathcal{D}(1, x) \times c$. The other nodes send their decryption shares to the network handler after receiving the commitment from the first node. Finally the network handler uses the decryption shares to retrieve the permuted messages blinded with Ka':

$$\begin{aligned}&\mathcal{P}_1'(M') \times S_1' \times \Pi_{i=1}^n(\mathcal{D}(i, x) \times c \times ka_i') \\ &= \mathcal{P}_1'(M') \times S_1' \times S_1'^{-1} \times Ka' \\ &= \mathcal{P}_1'(M') \times Ka'\end{aligned}$$

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

D. Detecting Tagging Attacks

If it is possible to determine whether an output message is valid, for example because it is an English message, a tagging attack can be performed by any of the nodes to determine the output slot that corresponds with a specific input slot. 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. The value for the slot j for which it wants to learn the recipient is replaced by $ka_{i,j} \times r_{i,j} \times t$, where t can be a random group element. In the postprocessing step, the node waits for the other nodes to reveal their decryption shares and the output of the mixing step. Using its own decryption shares the compromised node retrieves the messages. It can then determine for which slot the message looks odd, but it is correct 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 is now still the permuted original messages.

To prevent this attack, commitments on the values used in the last step are included in the protocol. The nodes have to commit to their decryption shares in order to prevent them to change it during the real-time phase to cancel out any tag added. The last node will have to commit to the output of the mixing step in order to prevent cancelling out any tag using these values. To detect whether any tagging took place, all the values are compared to their commitments. This can be done online, but also after the real-time phase as an audit. A later audit could be possible if the nodes have enough incentive to not be detected of acting malicious.

E. Including Recipient Keys

The system described above does not take confidentiality of the messages into account: the messages are output and the responses received in plaintext by the system. This might be sufficient for some applications and when required confidentiality could be added by encrypting messages before sending them to the system. However, it is also possible to extend the system such that the output messages and responses are also blinded. This way the recipient only needs to perform a single multiplication to retrieve the message and no computational more expensive public key cryptography is needed for this. Below we will describe what modifications would need to be done to the system in order to allow for this. An added advantage of this is that it is no longer possible to distinguish correct message in the output, making it impossible to perform the tagging attack described before.

For this new functionality we introduce the following additional notation:

- $kb_{i,j}$ and $kb_{i,j}'$: secret key shared between node N_i and the receiving user of slot j used to blind messages and responses respectively.
- Kb_j and Kb_j': the product of all shared keys for the receiving user of slot j: $Kb_j = \Pi_{i=1}^n kb_{i,j}$ and $Kb_j' = \Pi_{i=1}^n kb_{i,j}'$. The user for this slot stores the inverse of these keys, Kb_j^{-1} and $Kb_j'^{-1}$, to blind and unblind messages and response respectively.

For the forward path, the precomputation phase stays the same. We only need to change Step 3 in the real-time phase: instead of the nodes sending their decryption shares $\mathcal{D}(i, x)$, they send $\mathcal{D}(i, x) \times kb_i$ to the network handler. Similar as in Step 2 of the return path the output of the system would then be $\mathcal{P}_n(M) \times Kb$. Unblinding the message is very efficient as the recipient only needs to perform one multiplication to retrieve the original message.

The return path will change in both the precomputation and real-time phase. It will now 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 use this modification directly in applications where all the messages go to the same destination, for example, when using it for anonymous search. However, in other applications we would need to know which keys to use for the recipient. For this we need to add additional functionality to the forward path. Assuming the recipient also sends the response, no changes are needed for the return path.

One way to add in the recipients is to add a parallel session that uses fresh values for the random variables, but use 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 kb value to use for every slot and they can perform the third step for the actual messages. For more details, see Section VI.

V. Security Analysis

In this section, we analyze our protocol using the ideal/real world paradigm. We describe below the ideal world, which models the intended behavior of the system, in terms of functionality and privacy. We then provide a proof sketch to argue that our cMix protocol from the previous section can be securely abstracted by the ideal world, and informally show that the ideal world does indeed capture the required privacy properties.

A. Ideal World

In the ideal world we assume the existence of a trusted third party (TTP). Each mix-node in the network is connected

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($\mathcal{F}_{cMix}$, *start*)

upon Receiving(*start*) :
  SENDMSG($\mathcal{F}_{cMix}$, *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_i, *real-time*, 1) $\forall i \in [1, n]$
  Wait to receive(*real-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-time*, -1)
  Wait to receive(*real-time_finished*, -1) from $Handler$
  Send replies from B to corresponding U_j

upon Receiving(*precomp*) :
  SENDMSG(N_i, *precomp*, 1) $\forall i \in [1, n]$
  Wait to receive(*output_precomp*, 1) from N_n
  SENDMSG(N_n, *precomp*, -1)
  Wait to receive(*output_precomp*, -1) from N_1
  SENDMSG($\mathcal{F}_{cMix}$, *precomp_finished*)

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

upon Receiving(*corrupt*, N_i,j) from A :
  if $N_i \in \mathcal{C}$ **then**
    Attach a *corrupt* tag to jth 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 $Recipient$

Fig. 4: Ideal Functionality for cMix network $\mathcal{F}_{cMix}$

to every other mix-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 sub-machines for all n mix nodes. To communicate with each other through messages and data structures, these sub-machines 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 it 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.

upon Receiving(*phase*, *dir*) :
▷*phase* is equal to either real-time or precomp
  if $N_i \in \mathcal{C}$ **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($\mathcal{F}_{cMix}$, *output_precomp*, *dir*)

function COMPROMISEDNODE(M)
  Send M to A
  Wait to receive (N_i', M') from A
  Send message M' to N_i'

Fig. 5: Subroutines of $\mathcal{F}_{cMix}$ for node N_i

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($\mathcal{F}_{cMix}$, *real-time_finished*, *dir*)
▷Messages are retrieved and are ready to be delivered to recipients

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

upon Receiving(return, B') from A :
  SENDMSG($\mathcal{F}_{cMix}$, return, B')

Fig. 6: Subroutines of $\mathcal{F}_{cMix}$ for Handler

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 $\mathcal{C}$.

The adversary may corrupt some message, while it is getting processed at the compromised nodes by attaching a corrupt tag to the message; however, he cannot check or remove the tag until the message is output by the network handler. A 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(·,·), using 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. The pseudocode of the general ideal functionality is presented in Figure 4 and the pseudocode for cMix node subroutines is presented in Figure 5. Subroutines for Network Handler are depicted in the Figure 6. Unlike the cMix algorithm, $\mathcal{F}_{cMix}$ does not have any cryptographic operations such as encryption, decryption or commitments; the required security properties are instead insured by the the TTP.

As it was discussed in III-B, we assume secure authenticated channel between cMix nodes. Thus the only influence an attacker has on the messages sent between nodes is to delay or completely drop them, this is reflected in the SENDMSG(·,·) 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.

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

B. 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 1 (Simulation Security)**.** *If $\mathcal{E}$ 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 the cMix protocol is simulation secure as defined in Definition 1 in the random oracle model.*

Proof: 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 simulates 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. Both worlds also match in terms of communication flows, and the simulators are only left 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. Notice that the users are not involved in this step. The main observation here is that all the elements exchanged by the nodes are either commitments, 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 $\mathcal{S}_{pre}$ 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 $\mathcal{S}_{real}$ need 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 get 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 $\mathcal{S}_{real}$ 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 employ open the commitments to the shares such that they match the adversary messages.
- In the backward direction, we achieve this by changing the quotients of S and Ka' values for the honest nodes. As the expected adversary response messages are already known to $\mathcal{S}_{real}$, it can create the respective versions for those to collected by the adversary initiator users by manipulating its quotient values.

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

Therefore, using $\mathcal{S}_{pre}$ and $\mathcal{S}_{real}$, it is possible to simulate the responses expected by the adversary. It is also easy to see that both $\mathcal{S}_{pre}$ and $\mathcal{S}_{real}$ are efficient as 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. ■

C. Anonymity analysis

The cMix protocol ensures sender anonymity of its users. *Sender anonymity* holds if any sender in the cMix network is indistinguishable from all other potential senders. More precisely, all senders of a single round form an anonymity set within which they are indistinguishable. 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 [34] and formalized in [3].

We use this framework to define sender anonymity. Let the Challenger $Ch(b)$ define inputs from an adversary specified by the function α_{SA} (see Fig. 7) as input to the cMix protocol except for the challenge bit b. The message that Ch receives from the adversary is forwarded to $\mathcal{F}_{cmix}$ instead of the user defined 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 denoted as E_α. The goal of this section is to demonstrate that an adversary can only break sender anonymity of at least two honest users if he compromised all nodes in cMix.

Definition 2. *The cMix protocol provides (σ)-sender anonymity if for the function α_{SA} as defined in Fig. 7 for any adversary A with $0 \leq \sigma \leq 1$ if*

$$Pr[0 = \langle A|Ch(0)\rangle] \geq Pr[0 = \langle A|Ch(1)\rangle] + \sigma,$$

where $\sigma = Pr[E_\alpha]$

function $\alpha_{SA}(s, (\mathit{Sender_0}, \mathit{Recipient}, M), (\mathit{Sender_1}, _, _), b)$
    if $s \neq$ *fresh challenge* **then**
        output $\bot$
    else
        output $(\mathit{Sender_b}, \mathit{Recipient}, M,$ *challenge over*$)$

Fig. 7: Sender anonymity function (A simplified form) [3]

Assume E_α did not happen, but an adversary compromised the maximum number of $n-1$ nodes. Let us consider the forward round.

From a message M sent from sender to $\mathcal{F}_{cmix}$ A learns only the sender identity as defined in C_I and the position of the message. From messages send 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 $\mathit{Recipient_0}$ is receiving the message M_0. Since A compromised $n-1$ nodes, he gets to know 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 $\mathit{Sender_0}$ and $\mathit{Sender_1}$.

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,

$$Pr[0 = \langle A|Ch(0)\rangle|\neg Pr[E_\alpha]] = Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]].$$

The argument for the return round is similar. During the return round, $\mathit{Recipient_0}$ is sending a random message M_r back to $\mathit{Sender_b}$. Message M_r is forwarded by $\mathcal{F}_{cmix}$ to A. From messages send between any of the submachines in the IF, A learns the same information as in the forward round. From messages sent by $\mathcal{F}_{cmix}$ to users, A learns only the identities of the users who receive the messages.

Again, since A compromised $n-1$ nodes, he gets to know all but one of the permutations applied on the messages. He can determine the slot of the input message M_r and the two slots containing the output messages sent to $\mathit{Sender_0}$ and $\mathit{Sender_1}$. Since the permutation is random, A has a probability of $1/2$ for choosing one the two senders correctly regardless of the value of b. Thus, $Pr[0 = \langle A|Ch(0)\rangle|\neg Pr[E_\alpha]] = Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]]$. This equation holds for both the forward and return round.

Now 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].

$$\begin{aligned}
&Pr[0 = \langle A|Ch(0)\rangle] \\
&\quad= Pr[0 = \langle A|Ch(0)\rangle|Pr[E_\alpha]] \times Pr[E_\alpha] \\
&\qquad+ Pr[0 = \langle A|Ch(0)\rangle|\neg Pr[E_\alpha]] \times Pr[\neg E_\alpha] \\
&\quad= Pr[0 = \langle A|Ch(0)\rangle|Pr[E_\alpha]] \times Pr[E_\alpha] \\
&\qquad+ Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]] \times Pr[\neg E_\alpha] \\
&\quad\leq Pr[E_\alpha] + Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]] \times Pr[\neg E_\alpha] \\
&\quad\leq Pr[E_\alpha] + Pr[0 = \langle A|Ch(1)\rangle|\neg Pr[E_\alpha]] \times Pr[\neg E_\alpha] \\
&\qquad+ Pr[0 = \langle A|Ch(1)\rangle|Pr[E_\alpha]] \times Pr[E_\alpha] \\
&\quad= Pr[E_\alpha] + Pr[0 = \langle A|Ch(1)\rangle]]
\end{aligned}$$

Although similar to other mix-net designs, cMix provides a good protection against several standard traffic analysis techniques, we still need to study the applicability of the specialized mixing analyses such as [33], [36], [40].

D. Protocol integrity

The integrity property of cMix holds only if one of the two conditions holds:

1) the unmodified message M is forwarded to the recipient;
2) all the nodes learn that the protocol was not performed successfully.

In this work, we focus on the latter condition. We propose to use one of the existing mechanisms to achieve integrity. It is called *Randomized Partial Checking* (RPC) and is introduced in 2002 in [20]. This technique allows to perform a probabilistic verification if the outputs of the mix-net correspond to it's permuted inputs. Thus, it verifies not only the integrity of messages, but also if the permutations were applied correctly. Additionally, RPC allows to achieve probabilistic *accountability* [27]. It ensures that if the protocol run is performed incorrectly, at least some of the attackers are revealed with sufficiently large probability, at the same time honest parties are never blamed.

In RPC, nodes reveal certain information about a (large) part of their input/output pairs selected by either 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 the second one. RPC achieves a more relaxed (compared to the original mix protocol as described in [9]) level of anonymity under assumption of at least one pair of adjustment mix-nodes behaves according to the protocol as proven in [27]. When implementing RPC technique one has perform additional verifications to tackle issues in the original protocol described in [26].

VI. Applications

The cMix protocol serves well as a building block for a range of applications. Examples include private message delivery without use of public key and including confidential authentication of the sender to the recipient. Furthermore, so-called *"untraceable return addresses" (URAs)* can be realized and allow establishing a group to which all members of the group can send. In our work on PrivaTegrity, a number of additional applications are being developed using cMix as a primitive, including payments, photo sharing, anonymous feed following, and general credential mechanisms. Other possible applications include voting and anonymous surveys. These take advantage of the pre-arranged relationship of each user with each node, a new and promising security model.

In the remainder of this section, without going into formal detail or security arguments, we briefly abstract how the basic message delivery features and URAs can be realized using cMix. We plan to provide more details about these applications in subsequent writings.

Consider the delivery of a message sent anonymously through cMix to a user of cMix who is known to the sender, but where the payload is to remain confidential to the two users. In the absence of the additional use of public key or pre-arranged keys, we design a method in which message confidentiality will depend on non-collusion of the mix nodes.

The notion of a *"parallel components"* yields a solution. Two cMix batches can be processed using the same permutations but independent keys. Each user sends two inputs, each assigned the same slot position in its respective batch. In the two batches, because the same permutation is used for each batch, the corresponding inputs are linked, and the corresponding outputs are linked. The pair of user inputs, and the pair of outputs, may be called "coordinated components" and comprise what might be thought of as a "meta-message."

Private delivery of a message can be accomplished using parallel components as follows: The sender includes in the payload of the first of the two components in cleartext the identifier of the recipient within the cMix system along with the delivery address, if that is not implicit in the participant identification. The second parallel component of the meta message, as formed by the sender, has the cleartext message content to be delivered as its payload. In a first phase, the output of the first batch is posted, revealing the respective delivery address for each component of the second batch. In the second phase, the nodes choose the message key values "k-values") for the intended recipients (so that only the intended recipient can decrypt the message payload sent by the sender) and leave them in the respective components of the second output batch and allow that batch to be known to the network handler. In the third phase, the network handler delivers the messages in the second batch using the corresponding addresses posted in the first batch. This action allows the recipient to use the k-values it knows to recover the cleartext payload from the message received.

Along similar lines, a further extension allows for *authentication of the sender of a meta-message.* In this example a third batch of parallel components is used to provide the recipient of each message with the identifier of the sender of that message. To achieve this objective, the input to the third batch is provided entirely by the network handler and not by the sender; its payload is an identifier of the sender within cMix, which may be assumed known in such examples safely on the input side of the mixing. The output of the third batch is treated just as the output of the second batch: each cMix sender identifier is delivered confidentially to the corresponding recipient of the message.

Chaum [9] introduced the notion of *untraceable return address (URA)*. While rarely implemented, it is believed to have significant utility. For instance, providing delivery of replies to advertisements or postings of whatever type, while keeping the identity of the person placing the advertisement or making the posting otherwise anonymous. One way to incorporate URAs into cMix is to allow the user wishing to create a URA to create a "virtual user" with the nodes. This method entails setting up keys with each node for the virtual user. This setup, however, would preferably be done using an anonymous channel, such as possibly cMix itself, to maintain unlinkability between URA and the user who created it.

Group messaging can be achieved using URAs. Each group member publishes, anonymously through cMix, a URA it has created for this purpose along with the shared unique group identifier. When a message is to be sent to the whole group, its payload in the first parallel component is the group identifier. The second parallel component contains the message to be sent to the group as its payload. This second message component is replicated by the network handler and is used to create a separate message for each group member. Each of these messages is sent through a subsequent final cMix. The decryption with the virtual user's k is accomplished on the input to the final cMix by the nodes using the group's k, but then includes the URA creator's k to protect its privacy in transit to the anonymous group member.

Immediate replies from the receiver are handled in the following way. In the considered example of web-browsing, these are the delivery confirmations. For the confirmation messages, the last mix node waits for a notification from the receivers for a fixed amount of time to generate a batch of β responses. After the waiting time expires, the last node generates any sending-failure notifications and adds them to the batch. All messages from that batch are sent within one sub-round, and thus have the same length.

Replies that are not delivered immediately are handled in a separate round, using URAs. Longer replies are split into several messages that are transferred in separate rounds.

VII. Implementation and Benchmarks

We implemented our prototype system in Python, supporting forward and return paths. Fig. 8 shows the system architecture, which includes users, nodes, and a network handler. Each node includes a keyserver (to establish shared keys with the users) and a mixnet server (to carry out the precomputations and real-time computations). Currently the prototype supports anonymous search and the publishing of plaintext messages that are automatically acknowledged. The commitments are at the moment simulated by computing a SHA-256 hash. For the precomputation, the computation of the encryptions and decryption shares is performed by a parallel process on the nodes.

We ran experiments by installing the prototype on Amazon Web Services (AWS) instances, with each node comprising a c3.large with two virtual processors and 3.75GB of RAM. For encryption, including of messages and random values, we used a group with a prime-order of 1024 bits.

On the AWS instances, each 1024-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 1000. 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 about 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.

Table I gives timings for selected batch sizes. The means of the different phases are 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 delays in network traffic. To illustrate typical observed timings, we also include the medians timings. Still, the mean 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. Additional processors would significantly improve the time it takes to compute all necessary encryptions and decryption shares.

VIII. Performance Analysis

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

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

In the real-time phase for the forward path, cMix performs $4\beta(n+1)$ group multiplications. For every slot, three multiplications are used to remove the shared keys, and to add the r and s values. In addition, cMix performs n multiplications per slot to combine the decryption shares and to decrypt the precomputed value, and one to take out the precomputed value from the result. Thus, the real-time phase takes $4\beta(n+1)\tau_{\mathcal{M}}$ time for all multiplications.

It follows that the computations in the real-time phase are approximately $(5/8)(\tau_{\mathcal{E}}/\tau_{\mathcal{M}})$ 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 $(5/8)(\tau_{\mathcal{E}}/\tau_{\mathcal{M}})\alpha$ 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 on average, the messages have average sizes of $2\beta\lambda$ and $\beta\lambda$ respectively.

IX. Discussion

We discuss our major design decisions, PrivaTegrity's authentication and accountability model, and future work.

A. Major Engineering Design Decisions

cMix is not just another mixnet; it is fundamentally different. Our design motivation is to enable large anonymity sets (large batch sizes) and many mix nodes. This design motivation implies the need for a highly scalable architecture with fast real-time computations. To achieve this goal, we must depart from two limiting practices of traditional anonymity systems: real-time public-key operations, and the involvement of the user in establishing routing paths through the mix nodes. By contrast, as a result of these limiting practices, current implementations of anonymity systems use small batch sizes and a small number of mix nodes.

To achieve the aforementioned goals, our first major design decision was to replace all real-time public-key operations with fast symmetric-key operations, by using precomputed group-homomorphic encryptions of random values. Our second major

Batch size	Precomputation - Total (s) Mean	Median	Real-time - Forward (s) Mean	Median	Real-time - Return (s) Mean	Median	Real-time - Total (s) Mean	Median
50	1.56	1.51	0.15	0.10	0.06	0.04	0.20	0.14
100	3.02	2.94	0.24	0.18	0.09	0.08	0.33	0.25
200	5.87	5.84	0.47	0.32	0.18	0.15	0.64	0.47
300	8.85	8.79	0.68	0.48	0.26	0.22	0.70	0.94
500	14.59	14.56	1.06	0.81	0.44	0.37	1.51	1.18
1000	28.87	28.86	2.19	2.04	0.90	0.85	3.09	2.95

TABLE I: Mean and median of the timings in seconds (s) of 100 runs of the precomputation and real-time phases for different batch sizes.

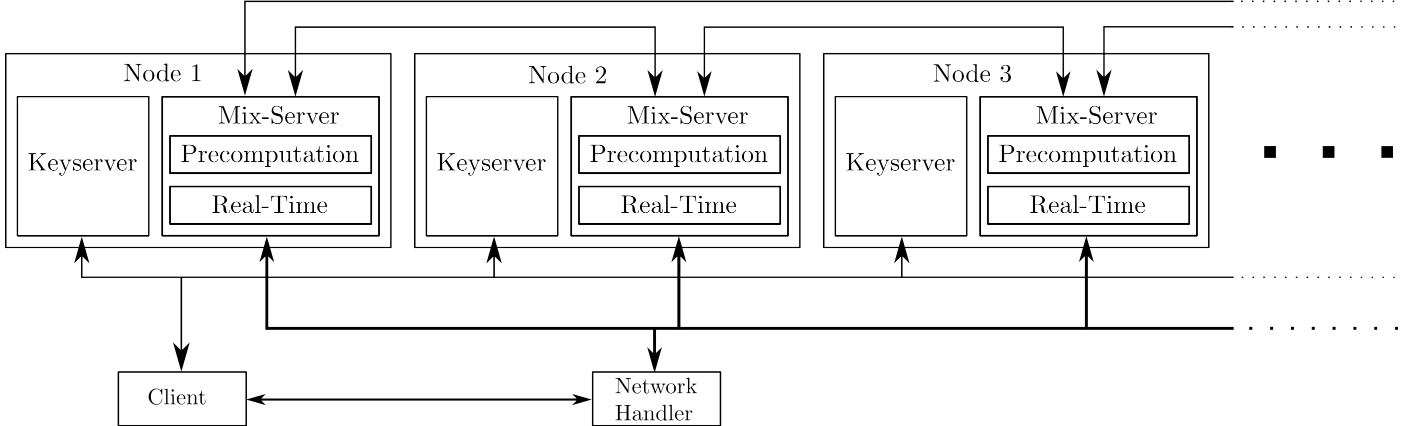

Fig. 8: Architecture for our prototype.

decision was for each sender to establish a shared key separately with each of the mix nodes, thus avoiding the need to involve the sender in the establishment of routing paths. These design choices avoided the two limiting practices mentioned above and achieved our design goals.

The use and identification of a network handler was a minor design decision. All mixnets must provide some similar functionality, though not necessarily through such a named entity. For example, some of this functionality might be carried out by the first mix node. We find it conceptually helpful to abstract the functions of the network handler in a separate named entity.

B. Anonymity and Accountability in PrivaTegrity

Independent from cMix, PrivaTegrity addresses potential abuse of anonymity services by establishing a trust model that offers a balance of anonymity and accountability. On the one hand, PrivaTegrity aims to provide privacy at a technical level that is not penetrable by nation states. On the other hand, PrivaTegrity aims to provide integrity, both prior restraint and accountability after the fact, that is inescapably tied to individuals. Only if all of the mixing nodes cooperate, can the senders and receivers of messages be linked or identified.

PrivaTegrity implements a new approach to user identification requiring each user to provide a small but different type of identifying information to each mix node. Some nodes may require photos or answers to personal history questions; others may request mobile phone numbers or email addresses. A user reveals comparatively little to any single node, but collectively the nodes possess significant identifying information. Each node can obligate itself contractually to trace and aggregate identifying information only according to a published policy, resulting in accountability and effective identification of users who violate the policy.

C. Future Work

Three tasks we plan to work on in the future include the following: First, we would like to deploy PrivaTegrity, including implementing and refining applications described in Section VI.

Second, we plan to explore different approaches for enforcing integrity of the mix nodes, to ensure that they cannot modify any message without detection.

Third, currently, message length is restricted by the group modulus. We would like to investigate if it is possible to allow any length message, for example, by using key-homomorphic pseudorandom functions [7].

X. Conclusion

The extraordinary speedup in real-time computation offered by cMix is a game changer. 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. PrivaTegrity's unique 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 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. Anna was conducting this research within the Privacy and Identity Lab (PI.lab, `http://www.pilab.nl`) and funded by SIDN.nl (`http://www.sidn.nl/`).

References

[1] B. Adida and D. Wikström, "Offline/online mixing," in *ICALP 2007*, 2007, pp. 484–495.

[2] M. Backes, I. Goldberg, A. Kate, and E. Mohammadi, "Provably secure and practical onion routing," in *Proc. 25th IEEE Computer Security Foundations Symposium (CSF)*, 2012.

[3] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi, "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] M. Backes, A. Kate, S. Meiser, and E. Mohammadi, "(nothing else) MATor(s): Monitoring the anonymity of Tor's path selection," in *Proceedings of the 21th ACM conference on Computer and Communications Security (CCS 2014)*, November 2014.

[5] M. Backes, A. Kate, and E. Mohammadi, "Ace: an efficient key-exchange protocol for onion routing," in *Proc. WPES'12*, 2012, pp. 55–64.

[6] J. Benaloh, "Simple verifiable elections," in *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*. USENIX Association, 2006.

[7] D. Boneh, K. Lewi, H. W. Montgomery, and A. Raghunathan, "Key homomorphic PRFs and their applications," in *Advances in Cryptology - CRYPTO 2013*, 2013, pp. 410–428.

[8] J. Camenisch and A. Lysyanskaya, "A formal treatment of onion routing," in *Advances in Cryptology — CRYPTO*, 2005, pp. 169–187.

[9] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 4, no. 2, pp. 84–88, 1981.

[10] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: high-speed onion routing at the network layer," in *Proc. 22nd ACM Conference on Computer and Communications Security*, 2015, pp. 1441–1454.

[11] J. Cox, "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.

[12] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III anonymous remailer protocol," in *Proc. 24th IEEE Symposium on Security & Privacy*, 2003, pp. 2–15.

[13] G. Danezis and I. Goldberg, "Sphinx: A compact and provably secure mix format," in *Proc. 30th IEEE Symposium on Security & Privacy*, 2009, pp. 269–282.

[14] G. Danezis and B. Laurie, "Minx: A simple and efficient anonymous packet format," in *Proc. 3rd ACM Workshop on Privacy in the Electronic Society (WPES)*, 2004, pp. 59–65.

[15] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. 13th USENIX Security Symposium (USENIX)*, 2004, pp. 303–320.

[16] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths," in *Proc. 18th USENIX Security Symposium*, 2009, pp. 33–50.

[17] I. Goldberg, D. Stebila, and B. Ustaoglu, "Anonymity and one-way authentication in key exchange protocols," *Designs, Codes and Cryptography*, pp. 1–25, 2012.

[18] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Onion routing," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[19] C. Gulcu and G. Tsudik, "Mixing email with Babel," in *Proc. of the Network and Distributed System Security Symposium (NDSS '96)*, 1996, pp. 2–16.

[20] M. Jakobsson, A. Juels, and R. L. Rivest, "Making mix nets robust for electronic voting by randomized partial checking," in *11th USENIX Security Symposium*, 2002, pp. 339–353.

[21] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann, "The sniper attack: Anonymously deanonymizing and disabling the Tor network," in *(NDSS'14)*, 2014.

[22] A. Jerichow, J. Mller, A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Real-time mixes: A bandwidth-efficient anonymity protocol," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 495–509, 1998.

[23] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "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.

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

[25] A. Kate, G. M. Zaverucha, and I. Goldberg, "Pairing-based onion routing with improved forward secrecy," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, p. 29, 2010.

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

[27] R. Kuesters, T. Truderung, and A. Vogt, "Formal analysis of Chaumian mix nets with randomized partial checking," Cryptology ePrint Archive, Report 2014/341, 2014, http://eprint.iacr.org/.

[28] B. Möller, "Provably secure public-key encryption for length-preserving Chaumian mixes," in *Proc. CT-RSA*, 2003, pp. 244–262.

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

[30] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE Symposium on Security and Privacy*, 2005, pp. 183–195.

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

[32] L. Øverlier and P. F. Syverson, "Locating hidden servers," in *IEEE Symposium on Security and Privacy*, 2006, pp. 100–114.

[33] F. Pérez-González and C. Troncoso, "A least squares approach to user profiling in pool mix-based anonymous communication systems," in *IEEE WIFS 2012*, 2012, pp. 115–120.

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

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

[36] D. Rebollo-Monedero, J. Parra-Arnau, J. Forné, and C. Díaz, "Optimizing the design parameters of threshold pool mixes for anonymity and delay," *Computer Networks*, vol. 67, pp. 180–200, 2014.

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

[38] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "Raptor: Routing attacks on privacy in Tor," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 271–286.

[39] "The Tor project," https://www.torproject.org/, 2003, accessed Nov 2015.

[40] C. Troncoso and G. Danezis, "The bayesian traffic analysis of mix networks," in *ACM CCS 2009*, 2009, pp. 369–379.

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