Easy-to-Implement Two-Server based Anonymous Communication with Simulation Security

Adam Bowers
University of Missouri
Columbia, USA
adamconaldbowers@gmail.com

Jize Du
University of Missouri
Columbia, USA
jdwrc@mail.missouri.edu

Dan Lin
University of Missouri
Columbia, USA
lindan@missouri.edu

Wei Jiang
University of Missouri
Columbia, USA
wjiang@missouri.edu

ABSTRACT

Anonymous communication, that is secure end-to-end and unlinkable, plays a critical role in protecting user privacy by preventing service providers from using message metadata to discover communication links between any two users. Techniques, such as Mix-net, DC-net, time delay, cover traffic, Secure Multiparty Computation (SMC) and Private Information Retrieval, can be used to achieve anonymous communication. SMC-based approach generally offers stronger simulation based security guarantee. In this paper, we propose a simple and novel SMC approach to establishing anonymous communication, easily implementable with two non-colluding servers which have only communication and storage related capabilities. Our approach offers stronger security guarantee against malicious adversaries without incurring a great deal of extra computation. To show its practicality, we implemented our solutions using Chameleon Cloud to simulate the interactions among a million users, and extensive simulations were conducted to show message latency with various group sizes. Our approach is efficient for smaller group sizes and sub-group communication while preserving message integrity. Also, it does not have the message collision problem.

CCS CONCEPTS

• **Security and privacy → Pseudonymity, anonymity and untraceability**.

KEYWORDS

anonymous communication, secure multiparty computation

ACM Reference Format:
Adam Bowers, Jize Du, Dan Lin, and Wei Jiang. 2022. Easy-to-Implement Two-Server based Anonymous Communication with Simulation Security. In *Proceedings of the 2022 ACM Asia Conference on Computer and Communications Security (ASIA CCS '22), May 30-June 3, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3488932.3523264

1 INTRODUCTION

Secure end-to-end communication is an essential tool to protect data confidentiality and personal privacy. Although this mechanism prevents network providers from accessing user data, they still know who is talking to whom from message metadata which can disclose a great deal of information regarding an individual [13, 35]. Thus, only hiding the content of a conversation is not enough. To prevent metadata from linking users and revealing their private information, we can utilize anonymous communication tools. In this paper, anonymous communication means the channel is:

(1) *Secure end-to-end*: The communication channel between any two users, e.g., Alice and Bob, is secured. That is, only Alice and Bob can access the information shared between them.
(2) *Anonymous/unlinkable*: Only Alice and Bob know they are communicating with each other. A service provider knows data was sent or received among a group of people, but it does not know which two users are actually communicating. The degree of anonymity depends on the size of the group.

Several methods based on Mix-net [9] have been proposed to create anonymous communication, such as Loopix [33] and Karaoke [29]. To prevent traffic analysis [21], these solutions also incorporate either time delay or covering traffic to hide communication patterns among the users. Like Vuvuzela [38], their security guarantee is related to Differential Privacy (DP) [5, 17] which offers a weaker anonymity protection due to its small information leakage. Pung [3] and MCMix [2] offer stronger anonymity guarantee by adopting Private Information Retrieval (PIR) [12, 26] and Secure Multiparty Computation (SMC) [19, 40] techniques respectively. Pung is computationally secure. By "computational", we mean the protocol does not leak any information regarding the original messages assuming probabilistic polynomially-bounded adversaries. MCMix can theoretically achieves information theoretic security that assumes the adversary has unlimited computing power. Both security guarantees are stronger than DP, but information theoretic security is considered to be the strongest among the three models.

The current implementation of MCMix is not secure against malicious adversaries who can arbitrarily diverge from the prescribed execution path of a protocol. More importantly, message loss is inevitable under MCMix when multiple users simultaneously send message to the same user. This can cause a situation where honest users unknowing jam the system to deny communications among other users. (Details are provided in Section 2.)

1.1 Our Main Contribution

The main goal of this paper is to propose simple and novel solutions to implement an anonymous communication framework with stronger security guarantees against malicious adversaries. Specifically, we design novel broadcasting and secret sharing based two-server protocols for achieving anonymous communication, which possess several desirable properties:

- *Stronger security*: Our proposed protocols are information theoretically secure when one server follows the protocol or the two servers do not collude.
- *Design simplicity*: (1) Most operations are sending/receiving messages which can be easily implemented with servers only providing storage related functionalities. (2) The computations on the server side are minimal, and no communications are required between the servers. (3) Only two non-colluding servers are needed to implement the proposed protocols.
- *Asynchronous dial protocol*: The existing solutions require the users being online and participating in the dial protocol simultaneously. In our design, users do not need to be online all the time, and no synchronization is needed among the parties. Dial protocol allows two parties to establish a "handshake" before starting the actual communication.
- *Built-in support for subgroup communication*: Suppose the subgroup size is κ. The existing solutions need to go through κ rounds and incur κ times of the baseline complexity. However, if $\kappa \leq \left\lfloor \frac{n}{2} \right\rfloor$, our protocols can handle subgroup communication in one round without incurring much additional complexity, where n denotes the group size.

The main drawback of our protocols is adopting broadcasting as a building block. For Bob to send a message to Alice within a group of n users, the network will incur about $n + 2$ messages in our approach. Comparing to some existing solutions, our approach incurs higher message complexity when the group size becomes large but possesses stronger security guarantees and is much easier to implement. Detailed comparison is given in Section 2.

The rest of the paper is organized as follows: Section 2 presents the existing anonymous communication platforms and discusses their limitations. Section 3 provides an overview of the proposed solution and the threat model. Section 4 presents the protocol details. Section 5 provides security and complexity analyses, and extensions for subgroup communication. Section 6 presents the empirical results to show the scalability of the proposed solutions. Section 7 concludes the paper with lessons learned and future work.

2 RELATED WORK

There has been countless research related to anonymous communication. A good starting place is the survey on secure messaging [37]. Here we group them into several categories based on their underlining design principles and security guarantees.

2.1 Mix-Net and Differential Privacy

Mix-net [9] based solutions, Tor [16], Hornet [10], and information slicing [22], aim to provide highly efficient anonymity from all but a global adversary. Anonymity is not guaranteed if all links in the network can be observed [21], but in exchange, they can scale to millions of users and terabytes of traffic as shown by Tor.

The second set of works Vuvuzela [38], Alpenhorn [30], Stadium [36], and Karaoke [29] were published in series. They achieve Differential Privacy [5, 17] for encrypted conversation metadata and the latest work, Karaoke, can scale to millions of users. This is done by shuffling traffic between a subset of hundreds of servers and verifying protocol execution. The users need to be online every round, and a party is required to coordinate the start of rounds. Alpenhorn uses an additional set of servers to derive a shared secret between two users using identity-based encryption. XRD [28] instead proposes a novel hybrid shuffle that relies on several separate Mixnets to scale and ensures at least one hop in common between all chains. Loopix [33] adopts time delay to hide traffic patterns and a number of servers to improve system response time. cMix [8] also employs mix-net design but can shuffle messages faster by not using public key operations. All these solutions assume that at least one server is honest and they offer a weaker security because Differential Privacy, by definition, leaks a small amount of information [2]. These solutions can protect message integrity through authenticated channels, but they cannot achieve this for the initial messages that are used to set up the channels. Furthermore, a general framework is proposed in [4] that analyzes the security of anonymous communication protocols based on Differential Privacy.

2.2 SMC and PIR

Secure Multiparty Computation (SMC) and Private Information Retrieval (PIR) can also be used to achieve anonymous communication with stronger security guarantees compared to Differential Privacy. Pung [3], a PIR based technique, offers an anonymity of $\frac{1}{n}$ where n is the group size, but it assumes the existence of a dial protocol to connect users and cannot verify message integrity. Talek [11] is also based on PIR, but it assumes that a group of users shares a common secret and it cannot detect message modifications by malicious servers. Riffle [27] is an interesting middle ground in that any user can access any message and download all messages, or just download a particular message of interest using PIR. It can verify if the servers behaved correctly during the message shuffling phase, but it does not guarantee the users will receive the correct messages. Riposte [14] and Express [18] use distributed point functions to implement anonymous communication. Riposte requires three servers to prevent malicious behaviors. On the other hand, Express only requires two servers and adopts symmetric key encryption and message re-randomization to improve efficiency. To our knowledge, both systems do not provide a mechanism to detect if a message is modified by a malicious server. In addition, the mailbox owner need out-of-band channels to distribute the address of a mailbox to other communicating peers. AsynchroMix [31] and Spectrum [32] both implement secure broadcasting and do not support end-to-end anonymous communication.

MCMix [2] is the other anonymous communication protocol based on SMC, but it is only secure under the semi-honest adversary model where the servers follow the protocol. Thus, its security guarantee is weaker than ours. MCMix utilizes an oblivious sorting protocol [20] to pair up users during dial phase and swap their messages during conversation phase. The protocol requires at least three servers, and as long as two servers are honest and do not collude, anonymity is guaranteed and bounded by the group size n. However, message integrity cannot be verified or ensured which our proposed protocol intends to solve. The message complexity in the entire system is bounded by $\Omega(ln \log n)$. These complexities are for sending one or "more" messages. However, how many more messages can be delivered each round is not clear due to the fact that starvation could occur when multiple senders want to communicate with the same recipient during each round.

In addition, MCMix has much higher communication round complexity comparing to 1 round in our case. Furthermore, our protocol allows efficient subgroup communication with amortized communication complexity $O(n)$ which is more efficient than MCMix. In summary, the mix-net design generally achieves differential privacy [5, 17] for encrypted conversation metadata against global adversaries. Differential privacy offers weaker security guarantee against the global adversary comparing to MCMix, Pung, and our approach. Comparing to MCMix and Pung, our solution provides detectability of malicious behaviors to ensure message integrity.

2.3 DC-Net and other Approaches

The DC-Net [7] inspired approaches, DiceMix [34], Verdict [15], and Dissent-AT [39], enable a user to anonymously send a message to a group of users, much like the Dining Cryptographers Problem. These approaches offer strong anonymity properties, but they are not designed for point-to-point communication since all users see every message. To make these solutions work in our problem domain, each pair of users have to share a secret key, and a user would need to decrypt every message. If one of the messages can be decrypted successfully, then the user know he or she is the recipient. This approach is inefficient since all keys and messages pairs have to be examined to retrieve the actual messages.

Like our approach, Cloudtransport [6] uses major storage services like Dropbox or Amazon. They argue that an authority would not want to ban a service like Amazon that plays a significant role in running the country's infrastructure. Therefore, if citizens use that service, the authority will not shut down the service without harming themselves. Users create a rendezvous account within the service. Then some bridge (a third party) uses the account to upload a file on the user's behalf. Finally, another user uses the bridge to retrieve the file on the user's behalf. The issue with this work is that implementation details are not fully specified. For two users to coordinate file exchange, they either need a third party to facilitate exchanging information or need some kind of out-of-band communication. Moreover, it is unclear how to keep user's information secure and anonymous from these third parties.

3 DESIGN OVERVIEW

The core idea of the proposed solution is depicted in Figure 1. As shown, Bob's message m is represented with two shares, denoted by m_1, a random bit string of equal length to m, and $m_2 = m \oplus m_1$. The two shares are stored separately at Server 1 and Server 2. Since each share when viewed individually is random (or pseudo-random in practice depending on the randomness of the generator), each alone does not leak any information under the information-theoretical model (or computational model in practice) regarding m aside from the length of the message. The length can be hidden by padding each share up to a maximum allowable message size. Then the servers can deliver their own shares to users connected to Bob in the network. By combining these shares locally, Bob's friends will be able to retrieve the actual message m. This setting can be extended with more than two servers.

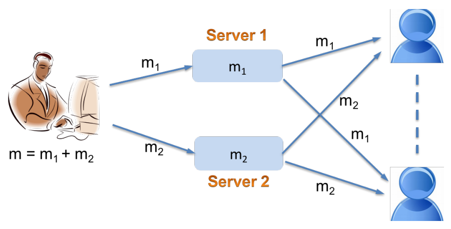

Figure 1: Two-Server Framework for Secure Messaging

The approach given in Figure 1 achieves data confidentiality against the service providers as long as one of the servers follows the protocol or the two servers do not collude. However, the approach is not sufficient to achieve end-to-end secure and anonymous communication since everyone connected to Bob will receive his message. At the other extreme, Bob could send a unique message to each user in one round of communication. This achieves anonymity but increases the amount of messages Bob needs to send. Ideally, if Bob wants to communicate with one person in the group, he should only need to send a constant number of messages. Thus, the key challenge is how to efficiently use secret sharing and the two-server framework to build an anonymous communication network between any two users in a group.

As the existing solutions, our protocol operates in round, and each round includes sending and receiving messages. However, unlike the existing solutions, during each round, our protocol allows a user to send and receive messages to and from multiple other users. In addition, the dial protocols (e.g., obliviously connecting a pair of users before conversing) of the existing work require participation of all users at the same time. On the other hand, our dial protocol is asynchronous and independent from the messaging protocol which provides extra flexibility in practice.

3.1 An Alternative Design based on Broadcasting and PKI

It is possible to design a simple protocol utilizing broadcasting and public key infrastructure (PKI), like BAR [25]. In this solution, all users' public keys are shared, and there is a single server. To send a message m to user i, the sender encrypts m with the public key pk_i of user i and sends the ciphertext $E_{pk_i}(m)$ to the server. The server collects and forwards these ciphertexts to all users. Thus, each recipient receives $\Omega(n)$ messages in each time step if there are total n senders. Each user then tries to decrypt all of the messages it received from the server in every round. This approach can be improved in the following ways:

- PKI key management at user level is both expensive and challenging. For instance, to make sure the legitimacy of the public keys, the users have to send their keys through different channels directly to the other users.
- A malicious server can refuse to broadcast the messages to users without being detected. Adding another server can solve this problem assuming one of the servers is honest. Our proposed solution also adopts the two-server setting; however, we eliminate the need of PKI among the users.
- Our approach, combining secret sharing and the two-server setting, is theoretically more secure than the PKI-based solution, and it only assumes that the communication channels between a server and a user is private. Although private channels are commonly achieved using PKI in practice, more secure solutions are possible, such as secure communication based on quantum cryptography (SECOQC).
- PKI can leak information on user identifies. If users want to remain anonymous in addition to their communication patterns, PKI may not be applicable.

3.2 Threat Model and Assumptions

The anonymous communication application generally classifies the participating parties into three categories:

- Users: the entities who communicate with other users in the same anonymity group or network.
- Servers: the entities who provide anonymous communication services to the users.
- Global adversary: who can observe the entire network traffics and attempt to identify the end-to-end communication links between any two users.

The standard threat model adopted by the existing solutions assumes that the users are honest, at least one server follows the protocol, and the global adversary does not collude with the servers. Also, the communication channel between a user and a server is private. To our knowledge, the existing solutions do not consider the situations where both servers and users are malicious. Initially, we adopt the same threat model as the existing work, and we later discuss possible secure extensions of our protocol against both malicious servers and users.

Since the servers only send/receive messages from the users and manage the session keys (used for pair-wise communication between two users), malicious behaviors regarding the servers may include: not participating in the protocol, modifying the messages and the session keys. A malicious global adversary can modify and drop messages. A malicious user may eavesdrop on other users' conversation. In summary, our initial solution provides the following security guarantees when either server follows the protocol:

- *Anonymity and confidentiality*: Connection anonymity and message confidentiality are achievable. Connection anonymity refers to the fact that the servers and the global adversary cannot predict the communication link between a pair of users better than a random guess.
- *Detectability or verification*: Malicious behaviors can be detected. In other words, the users are able to verify the integrity of the received messages.

Later, we propose protocol extensions that combat malicious users (while still assuming one of the servers is malicious) to prevent:

- *Eavesdropping on other users*: Our initial design assumes that each user can only have one account at each sever. If a malicious user creates multiple accounts at each server, the user may be able to eavesdrop on other users' communication; that is, message confidentiality cannot be guaranteed if a malicious user opens multiple accounts at each server. To prevent this, we embed Diffie-Hellman key exchange protocol into the two-server framework that allows a user to establish secure pairwise communication in the presence of malicious users.
- *Framing the honest server*: Malicious users may frame honest servers by generating messages that do not follow the required message format. In this case, we need to adopt digital signature schemes to authenticate the origin of the messages. However, we do not need PKI or a central certificate authority, and the two servers and the users work together to authenticate the keys.

Note that preventing server-user collusion is seemingly impossible because a malicious message recipient can simply share the received messages with a malicious server or other entities without being caught. We are not aware of any existing solutions that can prevent this type of collusion. Thus, our proposed solutions assume that the servers and users do not collude and malicious behaviors are limited to the aforementioned ones.

4 THE PROPOSED PROTOCOL

The proposed anonymous communication protocol is based on secret sharing and the two-server framework presented in Figure 1.

4.1 Secret Sharing Scheme

Assuming m is a non-negative integer, to secretly share m between two servers S_1 and S_2, first randomly select a value m_1 from $\mathbb{Z}_p$ where p denotes the domain size for the shares and is sufficient large to accommodate m. Then m_2 can be computed according to $m = m_1 + m_2 \bmod p$. The secret share m_i is stored at S_i for $1 \leq i \leq 2$. If the shares are intended for Alice, S_i will send m_i to her. By combining the two secret shares together, Alice can learn m. Because m_1 is randomly generated, it does not leak any information regarding the actual message m. In addition, due to the "+" and "mod" operations, m_2 is also randomly distributed in $\mathbb{Z}_p$. That is, m_2 alone does not reveal any information regarding m. Thus, m_1 and m_2 are called secret shares of m. The XOR operator $\oplus$ can also be used to generate secret shares in our protocol.

4.2 Illustration of the Main Idea

The proposed protocol consists of three phases: session key generation or dial protocol (one-time only for each communicating pair of users, Section 4.3), message distribution (Section 4.4) and message reconstruction (Section 4.5). Each session key is associated with a specific message recipient, and it is generated by the message sender. Each key can be used for multiple messages between the sender and the recipient. Once the key is established, the conversation can begin. The proposed session key generation protocol has the same purpose as the dial protocol in the literature. Unlike the existing solutions, our protocol can be performed asynchronously without requiring all users to participate at a specific time period. Thus, our protocol is more flexible. To implement this, we could use push or poll notification techniques utilized by E-mail applications. A temporary storage is required at the servers' side; however, the storage can be very small since only one connection request to a user needs to be buffered.

We consider a group of n users registered with two servers, and their IDs can be the same across these servers. User IDs are managed by the servers. If a new user joins the network and a user group, the servers broadcast the user's ID to the group. If a user already knows his or her friend's ID within the group, utilizing the network, the user can send the friend a secure message to reveal his or her real identity. If a user leaves a network or a specific group, the servers will remove the user ID from the groups' user lists.

To enable secure and anonymous pairwise communication using broadcasting, the main idea is to divide or partition this group of users into two randomly generated sub-groups. The message recipient is placed into different sub-groups across the two servers. The other users will be in the same sub-groups. In this way, only the recipient gets two shares of the actual message, and the other users get only one share of m. Thus, the resulting partitions/sub-groups serves as a session key for a pair of users to communicate securely.

Here we provide a concrete example to illustrate the above idea. Suppose there are a group of five users $G = \{u_1, u_2, u_3, u_4, u_5\}$. To achieve anonymous communication between any pair of users in G, two disjoint sub-groups are generated by a random process performed by the sender, e.g., Bob being one of the users:

- Randomly partition G into two random sub-groups G_1 and G_2, such that $G_1 \cap G_2 = \emptyset$ and $G_1 \cup G_2 = G$.

Suppose $u \in G$ is the message recipient. Based on G_1 and G_2, the partitions on S_1 and S_2 are assigned as follows:

- On server S_1: $G_1^{S_1} \leftarrow G_1$ and $G_2^{S_1} \leftarrow G_2$
- On server S_2: without loss of generality, assuming $u \in G_1$. Remove u from G_1 and add it to G_2 The resulting sub-groups are denoted by $\hat{G}_1$ and $\hat{G}_2$. $G_1^{S_2} \leftarrow \hat{G}_1$ and $G_2^{S_2} \leftarrow \hat{G}_2$.

Let's assume $u = u_3$ for this example, and $G_1 = \{u_2, u_3, u_5\}$ and $G_2 = \{u_1, u_4\}$. We have the following partition on each server:

- $G_1^{S_1} = \{u_2, u_3, u_5\}$ and $G_2^{S_1} = \{u_1, u_4\}$
- $G_1^{S_2} = \{u_2, u_5\}$ and $G_2^{S_2} = \{u_1, u_3, u_4\}$

The above group partitions can also be represented as a bit vector assuming the ordering of the user IDs is known to the user. In our example, the bit vector representations of the these partitions are given below:

- $G_1^{S_1} = 01101$ and $G_2^{S_1} = 10010$
- $G_1^{S_2} = 01001$ and $G_2^{S_2} = 10110$

After the partitions are generated, Bob sends either the actual partitions or their vector representations to the servers. Suppose Bob sends an l-bit message m to u_3, then Bob creates random shares with the following notation convention: m_j indicates the shares of the actual message m, and the subscript j is the server index.

- $m_1 \leftarrow_R \{0,1\}^l$ and $m_2 \leftarrow m \oplus m_1$
- $m_1' \leftarrow_R \{0,1\}^l$ and $m_2' \leftarrow_R \{0,1\}^l$

For u_3 to receive m, Bob sends the following messages:

- $m_1 \rightarrow \langle S_1, G_1^{S_1} = \{u_2, u_3, u_5\}\rangle$
- $m_2 \rightarrow \langle S_2, G_2^{S_2} = \{u_1, u_3, u_4\}\rangle$
- $m_1' \rightarrow \langle S_1, G_2^{S_1} = \{u_1, u_4\}\rangle$
- $m_2' \rightarrow \langle S_2, G_1^{S_2} = \{u_2, u_5\}\rangle$

The shares m_1 and m_2 are sent to S_1 and S_2 respectively and distributed to the sub-groups to which u_3 belongs. The random message m_1' is sent to the users in the sub-group indexed by 2 (i.e., $G_2^{S_1}$) at server S_1. The random message m_2' is sent to the users in the sub-group indexed by 1 (i.e., $G_1^{S_2}$) at server S_2. At the end, each user receives a pair of messages:

- u_1: $\langle m_2, m_1'\rangle \rightarrow m_2 \oplus m_1'$ $\qquad$ u_2: $\langle m_1, m_2'\rangle \rightarrow m_1 \oplus m_2'$
- u_3: $\langle m_1, m_2\rangle \rightarrow m_1 \oplus m_2$ $\qquad$ u_4: $\langle m_2, m_1'\rangle \rightarrow m_2 \oplus m_1'$
- u_5: $\langle m_1, m_2'\rangle \rightarrow m_1 \oplus m_2'$

Obviously, only u_3 receives m_1 and m_2. As a result, u_3 is able to reconstruct $m \leftarrow m_1 \oplus m_2$. This example only illustrates the key ideas, and it lacks implementation details such as, how does u_3 know m is the actual message and how do the other users know the messages they received are not the actual messages? These questions will be answered in Section 4.4. The generated disjoint sub-groups/partitions serve as a session key. Our formal group partition algorithm is presented next.

Algorithm 1 Group-Partition$(G, u) \rightarrow \langle G^{S_1}, G^{S_2}\rangle$

```
1:  for $i = 1$ to 2 do
2:      $G_i \leftarrow \emptyset$
3:  end for
4:  for each $u_i \in G$ do
5:      $j \in_R \{1, 2\}$
6:      $G_j \leftarrow G_j \cup \{u_i\}$
7:  end for
8:  $G^{S_1} \leftarrow \langle G_1, G_2\rangle$
9:  if $u \in G_1$ then
10:     $G^{S_2} \leftarrow \langle G_1 - \{u\}, G_2 \cup \{u\}\rangle$
11: else
12:     $G^{S_2} \leftarrow \langle G_1 \cup \{u\}, G_2 - \{u\}\rangle$
13: end if
14: return $\langle G^{S_1}, G^{S_2}\rangle$
```

4.3 Group Partition

Here we generalize the steps given in the previous example assuming $n \geq 2$. The key steps are given in Algorithm 1. The number of sub-groups is the same as that of the servers. Steps 1-3 initialize each sub-group to an empty set. Steps 4-7 distribute elements in G randomly to each of the two sub-groups, and the resulting set of the sub-groups are denoted by G^{S_1}. Steps 9-13 generate G^{S_2} by removing u from one sub-group and adding it to the other sub-group.

4.4 Generating Authenticated Messages

As illustrated in the previous example, the users in the same partition receive the same message, and the number of messages a user receives is the same as the number of servers, which is two. In addition, the number of partitions is also equal to 2. Therefore, the total number of messages that need to be generated by a sender is four, two of which are the secret shares of the actual message m. The rest of messages are randomly generated. Regarding the security guarantee, two important issues need to be addressed when generating these messages:

- The message recipient knows the message received is the actual message.
- The recipient can detect if the message has been altered by a malicious server.

To solve the above problems, we propose a way to construct an authenticated message. Let m be a message:

- $\hat{m} \equiv 1||m$: if m is real, prepend 1 to it.
- $\hat{m} \equiv 0||m$: if m is random, prepend 0 to it.

Next we define a scheme to authenticate $\hat{m}$ where the main idea is borrowed from [24]. Let q be a prime, such that $\hat{m} \in \mathbb{Z}_q^+$ and $m < \lfloor \frac{q}{2} \rfloor$. The authentication function is given as:

$$C(\hat{m}, r) = \hat{m} * r \bmod q, \text{ where } r \in_R \mathbb{Z}_q^+ \tag{1}$$

Then the shares of the message is produced as follows:

- $\langle m_1, m_2 \rangle \leftarrow$ Gen-Shares($C(\hat{m}, r)||r||\hat{m}, p$)

The Gen-Shares function produces secret shares of $C(\hat{m}, r)||r||\hat{m}$ according to the scheme discussed in Section 4.1 and p or $\mathbb{Z}_p$ denotes the share domain of m_1 and m_2. When m is very large, instead of authenticating m directly, the authentication code can be computed based on a cryptographic hash of m. Adding a hash computation to our scheme is straightforward, and we will ignore this issue for the rest of the paper.

Algorithm 2 Gen-Authenticated-Msg(m, p, q) $\rightarrow \vec{z_1}, \vec{z_2}$

```
Require: $m$ is the actual message, $p$ and $q$ are primes, such that
    $p > q^3$ to accommodate the three components of $C(\hat{m}, r)||r||\hat{m}$
1:  $\hat{m} \leftarrow 1||m$
2:  $r \in_R \mathbb{Z}_q^+$
3:  $\langle m_1, m_2 \rangle \leftarrow$ Gen-Shares($C(\hat{m}, r)||r||\hat{m}, p$)
4:  for $i = 1$ to 2 do
5:      $r'_i \in_R \{1, \ldots, \lfloor \frac{q}{2} \rfloor - 1\}$
6:      $r''_i \in_R \mathbb{Z}_q^+$
7:      $m'_i \leftarrow C(0||r'_i, r''_i)||r''_i||0||r'_i$
8:      $r_i \leftarrow m'_i \oplus m_i$
9:      $\vec{z}_i \leftarrow \langle r_i, m_i \rangle$
10: end for
```

The steps for generating authenticated messages and their shares are provided in Algorithm 2. The algorithm takes the actual message m and two prime numbers such that $p > q^3$, and m is less than $\lfloor \frac{q}{2} \rfloor$. Steps 1-3 produce authenticated actual message and its shares. Steps 4-10 produce authenticated random messages and their shares. The purpose of each main step is discussed below:

- Step 1: prepend flag 1 to m to indicate m is an actual message.
- Step 2: generate a random number r, and it is used to authenticate $\hat{m}$ in the next step.
- Step 3: first, compute the authentication code of $\hat{m}$, denoted by $C(\hat{m}, r)$. Then two additive shares of $C(\hat{m}, r)||r||\hat{m}$ are generated in $\mathbb{Z}_p$, denoted by m_1 and m_2.
- Step 5: r'_i is randomly generated for representing the i^{th} random message.
- Step 6: r''_i is randomly generated for committing or authenticating r'_i.
- Step 7: first, r'_i is prepended with a 0 flag to indicate r'_i is a random message. Then the commitment of $0||r'_i$ is computed. The whole authenticated message is denoted by m'_i.
- Step 8: r_i is generated by $m'_i \oplus m_i$. Thus, r_i and m_i are two shares of m'_i.
- Step 9: $\vec{z}_i$ denotes the shares of m'_i which will be distributed to the users in the same sub-group.

Steps 5-9 are repeated one more time to produce another authenticated random message and its additive shares.

4.4.1 *Distributing Shares.*

Once the partitions and secret shares are produced, the users are ready to communicate. Protocol 3 presents the key steps that instruct the sender to deliver the shares of his or her message to the servers which in turn send the shares to the users in the same group. In this protocol, we need to pay special attention to the two shares of m, each of which is distributed to one partition at a particular server.

- Step 1(a): The sender permutes the ordering of r_1 and m_1 so that server S_1 does not know which of the two messages is related to the actual message. Without this step, the protocol is still secure, but it makes easier to provide a simulation based proof. See Section 5.1 for more details. The permuted messages are denoted by ζ_1.
- Step 1(b): Based on $\vec{\zeta_1}$ and $\vec{z_2}$, this step produces $\vec{\zeta_2}$ as follows. If the first message or component of $\vec{\zeta_1}$ is m_1, then m_2 needs to be the second component of $\vec{\zeta_2}$. This makes sure that only the targeted user receives both m_1 and m_2.
- Step 1(c): The notation $\langle \vec{\zeta_1}[1], G_1^{S_1} \rangle$ indicates that the message $\vec{\zeta_1}[1]$ is intended for users in $G_1^{S_1}$ whose partition/sub-group is managed by S_1. The notation $\langle \vec{\zeta_2}[1], G_2^{S_1} \rangle$ and Step 1(d) can be interpreted similarly.
- Steps 2 and 3: Each server delivers the message shares to their intended users.

To prevent other users from receiving m, these users shall not receive both shares of m. The way that the partitions are formed and the shares are distributed enforces that only the message recipient obtains both shares of m. This will be proved in Section 5.

4.5 Message Verification

When a user receives messages from the servers, the messages are grouped by the sender's ID and the message time-stamps. Here we assume that m_1 and m_2 have the same timestamps, and this can be easily achieved by the sender's local application. As mentioned earlier, we only adopt additive secret sharing for the two-server setting. In this case, after a user receives both shares m_1 and m_2 of a message, the user can reconstruct $m \leftarrow m_1 \oplus m_2$ where m represents either the actual message or a randomly generated message. Let $c||r||b||m$ be the reconstructed message, the user performs the following verification steps:

- If $C(b||m, r) = c$ and $b = 1$, accept the message.
- If $C(b||m, r) = c$ and $b = 0$, ignore the message.
- If $C(b||m, r) \neq c$, malicious behavior is detected.

Protocol 3 Dist-Msg$\left(\vec{z}_1, \vec{z}_2, G^{S_1}, G^{S_2}\right)$

```
Require: Denote $\vec{z}_1 \equiv \langle r_1, m_1\rangle$ and $\vec{z}_2 \equiv \langle r_2, m_2\rangle$. Let $\vec{z}_i\langle 1\rangle$ and
    $\vec{z}_i\langle 2\rangle$ be the first and second element of $\vec{z}_i$ Denote $G^{S_1} \equiv$
    $\langle G_1^{S_1}, G_2^{S_1}\rangle$ and $G^{S_2} \equiv \langle G_1^{S_2}, G_2^{S_2}\rangle$
1: Sender:
    (a) Randomly permute the ordering of the elements of $\vec{z}_1$,
        denote the resulting vector as $\vec{\zeta}_1$
    (b) If $\vec{\zeta}_1[1] = m_1$:
            • $\vec{\zeta}_2 \leftarrow \langle r_2, m_2\rangle$
        Otherwise:
            • $\vec{\zeta}_2 \leftarrow \langle m_2, r_2\rangle$
    (c) Send $\langle\vec{\zeta}_1[1], G_1^{S_1}\rangle$ and $\langle\vec{\zeta}_2[1], G_2^{S_1}\rangle$ to $S_1$
    (d) Send $\langle\vec{\zeta}_1[2], G_1^{S_2}\rangle$ and $\langle\vec{\zeta}_2[2], G_2^{S_2}\rangle$ to $S_2$
2: Server $S_1$:
    (a) Send $\vec{\zeta}_1[1]$ to users in $G_1^{S_1}$
    (b) Send $\vec{\zeta}_2[1]$ to users in $G_2^{S_1}$
3: Server $S_2$:
    (a) Send $\vec{\zeta}_1[2]$ to users in $G_1^{S_2}$
    (b) Send $\vec{\zeta}_2[2]$ to users in $G_2^{S_2}$
```

5 PROTOCOL ANALYSES AND EXTENSIONS

In this section, we present the detailed security analysis of our protocol according to criteria given in Section 3.2. We will also discuss protocol complexity and how to efficiently handle many-to-one and one-to-many (sub-group) anonymous communications.

5.1 Security Analysis

First, we reiterate that sender anonymity is achieved via round synchronization, the same approach as the existing solutions. That is, the protocol operates in round, and all users participate in each round which effectively disguise the actual senders and the communication patterns. Additionally, the communication channels between users and servers are private. As a result, the global adversary who observes the entire network traffic cannot discover the peer-to-peer communication links among the users and the actual messages sent from users to servers, and vice versa.

The key functionality required at the servers is passing the messages around and no additional computations are needed. As a result, the other security guarantees of the protocol is directly related to the underlying secret sharing scheme and how the shares are distributed between the servers. First, we prove that our protocol achieves pair-wise communication. Then, we show message confidentiality and connection anonymity are guaranteed through the formal simulation based method.

Claim 1. *Suppose the group partitions are generated using Algorithm 1 and Algorithm 2, and the shares of m and other random values are distributed according to Protocol 3. Then only u can receive two shares of m, and the other users receive only one share of m.*

Proof. According to Algorithm 1, the users can be classified into two categories: u and $\{v_i\}$ where u is the targeted message recipient, and $\{v_i\}$ denotes the set of users who remain in the same partition across all servers. It is apparent that only u belongs two different partitions or sub-groups between the two servers. According to Protocol 3, m_1 and m_2 are sent to users in two different partitions. Therefore, it is clear that u receives all shares of m assuming no faults occurred in the system. All users in $\{u_i\}$ are in the same partition across all servers and receive either message pairs denoted by $\vec{\zeta}_1$ or $\vec{\zeta}_2$ which contains one valid share of m. □

The above claim is not valid when a malicious user can have multiple accounts at each server which may allow the user to receive all shares of the actual messages intended for other users. We address this issue in Section 5.3. Next we prove that the servers do not learn anything about the actual message based on the simulation paradigm presented in [19]. In general, when considering malicious adversaries, to prove a protocol is secure in literature of Secure Multiparty Computation (SMC) [19, 40], we need to show whatever an adversary can learn in the real model (executing the real protocol) is indistinguishable from what the adversary can learn from the ideal model (computations are performed by a trusted third party). Since our protocol does not return any values to the servers, our proof can adopt a simplified version of the read-ideal paradigm. In particular, we produce a simulator *Sim* that generates a simulated view of an adversary, identical to the real view. More details are provided below.

Claim 2. *Assume one server follows the protocol and the users do not collude with the servers, Protocol 3 is information theoretically secure against the servers. In other words, the servers learn nothing about the actual message.*

Proof. The servers receive two types of messages related to either the session key (Algorithm 1) or message shares (Algorithm 2 and Protocol 3). We build a simulator *Sim* for simulating both messages or views. Since our protocol is symmetric, *Sim* is the same for either server, and it performs the following with input G (the set of n) users and p (the domain size):

- Initialize G_1^* and G_2^* to be empty set.
- For each $u_i \in G$, *Sim* flips a fair coin: if it is head, u_i is added to G_1^*. Otherwise, u_i is added to G_2^*.
- Randomly generate r_1^* and r_2^* from $\mathbb{Z}_p$.

The simulated view of an adversary consists of:

- $\text{View}_{G,p}^{Sim} \equiv \left\{\left\langle r_1^*, G_1^*\right\rangle, \left\langle r_2^*, G_2^*\right\rangle\right\}$

According to the protocol, the real view of an adversary consists of:

- $\text{View}_{G,p}^{\pi} \equiv \left\{\left\langle\vec{\zeta}_1[i], G_1^{S_i}\right\rangle, \left\langle\vec{\zeta}_2[i], G_2^{S_i}\right\rangle\right\}$

Partly because r_1^*, r_2^*, $\vec{\zeta}_1[i]$ and $\vec{\zeta}_2[i]$ are independently and identically distributed in $\mathbb{Z}_p$ and partly because G_1^*, G_2^*, $G_1^{S_i}$, and $G_1^{S_i}$ are also independently and identically distributed, $\text{View}_{G,p}^{Sim}$ is identical to $\text{View}_{G,p}^{\pi}$. This implies that the adversary or each server does not learn anything about the actual message during protocol execution. Thus, Dist-Msg (Protocol 3) is information theoretically secure. □

Claim 3. *Using the additive secret sharing scheme and one server follows the protocol or the two servers do not collude, the proposed protocol achieves detectability with probability* $1 - \frac{1}{q}$.

Proof. In this proof, without loss of generality, we consider S_1 is the malicious server. There are two cases that guide our analysis:

- The malicious server S_1 wants the recipient u to receive and accept a specific message $\hat{m}$ whose two shares are denoted by $\hat{m}_1$ and $\hat{m}_2$.
- The malicious servers want the recipient u to receive and accept any $\hat{m}$ that is different from m.

In either case, S_1 modifies the first share to replace the actual share m_1 of m it received. As a result, u receives $\hat{m}_1$ and m_2 where m_2 is the non-modified share of m from server S_2. For u to accept a specific $\hat{m}$ as a legitimate message, it must be the case that $\hat{m}_2 = m_2$. Since m_2 is uniformly random, Prob$(\hat{m}_2 = m_2) = \frac{1}{|m_2|} = \frac{1}{q^3}$.

Under the second case, since the only condition for u to accept a message is to verify the commitment, the chance of a message, constructed by randomly generated shares, resulting a consistent commitment is given by $\frac{q^2}{q^3} = \frac{1}{q}$. Consequently, combining the two cases, any modification to the legitimate message can be detected with probability bounded by $1 - \frac{1}{q}$ in the worst case. □

As acknowledged before, when the users are malicious and have multiple accounts at each server, it is possible for these users to eavesdrop on other users and frame an honest server. We propose certain mechanisms to combat malicious users, and the detailed discussion is give in Section 5.3.

5.2 Message Complexity

The local computation at each server is negligible comparing to the communication cost. Thus, we only focus on message complexity including the number of messages and rounds. Suppose each message has a fixed size of s bits. Since the group partition needs to be performed once for a recipient (but not for each message), we separate the complexity into two phases: group partition and sending a message. Assuming each set can be packed into one message, then the sender sends two messages per server. Thus, the total number of messages sent is four or $4s$ bits. On the server side, local computation time is approximate to n since managing the grouping information only needs to store user IDs once. Since each server receives two messages or $2s$ bits, the total message complexity for the entire system is the same as those of the sender. The protocol requires only one round of communication between a sender and the two servers.

For sending a message, the sender needs to generate four messages and two of them are shares of m. The senders sends two messages to each server. Thus, the total number of messages sent is four or $4s$ bits. For the other users (including the targeted recipient), each of them receives two messages which is also the message complexity for the users. Each server receives two messages from the sender and sends them to n users. Therefore, the message complexity for each server is $2 + n$ or $(2 + n)s$ bits.

5.3 Security Against Malicious Users

Our current design assumes that each user can only have one account at each server and has the same user ID at both servers. If a user has multiple accounts at each server, it is possible for the users to obtain messages intended for other users. Here we provide a brief description on how to prevent malicious users from receiving all secret shares of m. Under the Diffie-Hellman key exchange protocol, suppose g is a generator of $\mathbb{Z}_p^+$ where p is a prime, and g and p public parameters. If Bob wants to send a confidential message to Alice, the two parties first need to agree on a secret key which encrypts the subsequent messages. Bob generates a random value x from $\mathbb{Z}_p^+$, and sends g^x to Alice under the proposed communication framework. Bob also needs to embed some special information to indicate this particular message is for establishing a secret key. After receiving the message, Alice generates a random value y from $\mathbb{Z}_p^+$, and sends g^y to Bob. Then both parties derive g^{xy} as the secret key for encrypting confidential messages between Alice and Bob.

Another change needs to be made is related to the session key generation or group partitioning for Bob to send the message related to g^x. To ensure a malicious user who has multiple accounts does not receive g^x, instead of two partitions at each server, Bob creates κ partitions. The larger the κ, the less likely the malicious user obtains g^x. The probability of disclosing g^x to any malicious user becomes 0 when Bob sets $\kappa = n$. In this case, Bob needs to generate more messages locally. However, this is one time cost, and the amortized cost is negligible if Bob wants to establish shared keys with a number of users which can be done all at once. On the other hand, when Alice replies with g^y, she does not need to change the standard partition protocol on her side.

5.3.1 *Preventing Malicious Users from Framing Honest Servers.*

A malicious user can frame innocent servers by intentionally generating messages that do not follow the required format, e.g., append a bit to indicate the message type. To prevent this from happening, we can require that when sending a message to the servers, each user adds a digital signature of the message which can be verified by the servers. As discussed in Section 3, PKI has several drawbacks if adopted in our application domain. In what follows, we presents key ideas on how a user can establish a trusted signing-verifying key pair of any digital signature scheme without using PKI. The following steps are performed between a user (e.g., Bob) and the two servers S_1 and S_2 each of whom has a publicly know public-private key pair $\langle pu_{s_i}, pr_{s_i} \rangle$:

- Bob generates a public-private key pair $\langle pu_b, pr_b \rangle$ for a given digital signature scheme agreed by participating parties. Send pu_b to both S_1 and S_2.
- Suppose S_i receives k_b^i from Bob. If all three parties follow the protocol, $k_b^1 = k_b^2 = pu_b$.
- S_1 and S_2 exchange the keys received from Bob:
  - S_1 sends $k_b^1 || \text{Sign}_{pu_{s_1}} \left(k_b^1 \right)$ to S_2, where $\text{Sign}_{pu_{s_1}} \left(k_b^1 \right)$ is S_1's signature of k_b^1.
  - S_2 sends $k_b^2 || \text{Sign}_{pu_{s_2}} \left(k_b^2 \right)$ to S_1, where $\text{Sign}_{pu_{s_2}} \left(k_b^2 \right)$ is S_2's signature of k_b^2.
- If $k_b^1 \neq k_b^2$ and the signatures cannot be verified by both servers, reject the key and abort.

If the above steps executed successfully, the key pu_b is established between Bob and the two servers. Afterwards, the servers are required to verify the subsequent messages they receive from Bob before forwarding them to the other users. If the messages cannot be verified, they will be rejected. When forwarding the messages to other users, each server also needs to append its signature to the messages. The users verify the signatures before accepting the messages from the servers. If the messages cannot be verified, the protocol aborts. Next we analyze why the above steps can prevent a malicious user from framing an honest server.

Without loss of generality, assume S_1 is honest, and Bob sends Alice a message m. Let m_1 and m_2 denote the shares of m. If m cannot be verified by Alice and S_1 gets blamed for it, the following steps have occurred:

- S_1 received $m_1||\text{Sign}_{pu_b}(m_1)$ from Bob, and the signature was verified.
- S_1 sent $m'_1||\text{Sign}_{pu_{s_1}}(m_1)$ to Alice who verified the signature and accepted m_1

The verification, that proves S_1 innocence, proceeds as follows:

- Alice broadcasts $m'_1||\text{Sign}_{pu_{s_1}}(m_1)$.
- S_1 broadcasts $k_b^2||\text{Sign}_{pu_{s_2}}\left(k_b^2\right)$, pu_b, and $m_1||\text{Sign}_{pu_b}(m_1)$.
- Honest users or servers can check that $k_b^2 = pu_b$, $m_1 = m'_1$ and the signatures can be verified. Then this proves that S_1 followed the protocol when sending the shares of m to Alice.

5.4 One-to-Many Communication

The existing solutions do not support efficient one-to-many anonymous communication in the sense that the message has to be sent (received) to (from) one person within each round. Additionally, message collisions cannot be avoided in MCMix. In other words, if two or more users want to send messages to the same recipient, some of these users will not be able to send any messages to their intended recipients. This starvation behavior can persist through each round of protocol execution. Clearly, assuming each user only communicates with a different user (to avoid message collision) during each round is not realistic and impractical, and we do not know if this is fixable based on the current design of MCMix.

Our solution does not have the message collision problem and can be easily modified to achieve efficient subgroup communication. Under one-to-many communication model, we consider two different situations:

- Subgroup communication: a user wants to send the same message to a subgroup of users.
- General one-to-many communication: a user wants to send different messages to a subgroup of users.

5.4.1 Subgroup Communication. The goal of our previously proposed partition algorithm is to generate random partitions given a single message recipient. However, it is not clear if the algorithm can produce the required randomness given a subgroup of users whose size is a fraction of n. Here we propose a different partition algorithm that works for a single user or a sub-group G_τ of users whose size is bounded by $\frac{n}{2}$ (i.e., $|G_\tau| \le \lfloor \frac{n}{2} \rfloor$). The key ideas for the new partition algorithm work as follows assuming n is divisible by 2 for illustration purpose:

- First, we permute the group of users and evenly divide them into two disjoint subgroups. During the permutation and group division, G_τ remains in the same subgroup.
- Then, we swap G_τ with a subset of users with the same size in another subgroup to produce the needed partitions for different servers.

Algorithm 4 Group-Partition*$(G, G_\tau) \rightarrow G^{S_1}, G^{S_2}$

```
Require: $G$ is a group of $n$, $G_\tau \in G$ (message recipients) and $|G_\tau| \le \lfloor \frac{n}{2} \rfloor$
 1: Permute $G$ and divide it into two equal size subgroups $G_1$ and $G_2$ while treating $G_\tau$ as an atomic element, i.e., $|G_1| = |G_2|$ and either $G_\tau \subseteq G_1$ or $G_\tau \subseteq G_2$
 2: $G_1^{S_1} \leftarrow G_1$ and $G_2^{S_1} \leftarrow G_2$
 3: $G^{S_1} \leftarrow \left\langle G_1^{S_1}, G_2^{S_1} \right\rangle$
 4: if $G_\tau \subseteq G_1$ then
 5:     $G_{\hat{\tau}} \leftarrow_R G_2$ and $|G_{\hat{\tau}}| = |G_\tau|$
 6:     $G'_1 \leftarrow (G_1 - G_\tau) \cup G_{\hat{\tau}}$
 7:     $G'_2 \leftarrow (G_2 - G_{\hat{\tau}}) \cup G_\tau$
 8: else
 9:     $G_{\hat{\tau}} \leftarrow_R G_1$ and $|G_{\hat{\tau}}| = |G_\tau|$
10:     $G'_1 \leftarrow (G_1 - G_{\hat{\tau}}) \cup G_\tau$
11:     $G'_2 \leftarrow (G_2 - G_\tau) \cup G_{\hat{\tau}}$
12: end if
13: $G_1^{S_2} \leftarrow G'_1$ and $G_2^{S_2} \leftarrow G'_2$
14: $G^{S_2} \leftarrow \left\langle G_1^{S_2}, G_2^{S_2} \right\rangle$
```

The key steps of the modified group partition algorithm are given in Algorithm 4.

- Step 1: G is randomly permuted while keeping G_τ as a single element in G, and then divide G into two equal partitions G_1 and G_2 such that G_τ belongs to one of the sub-partitions.
- Steps 2-3: Assign G_1 and G_2 as the group partitions for server S_1, denoted by $G_1^{S_1}$ and $G_2^{S_1}$ respectively.
- Steps 4-7: If G_τ belongs to G_1, then randomly select a subset $G_{\hat{\tau}}$ from G_2, and swap G_τ and $G_{\hat{\tau}}$ to create G'_1 and G'_2.
- Steps 9-11: If G_τ belongs to G_2, then randomly select a subset $G_{\hat{\tau}}$ from G_1, and swap G_τ and $G_{\hat{\tau}}$ to create G'_1 and G'_2.
- Steps 13 and 14: Create partitions G^{S_2} for S_2.

To enable subgroup communication, we also need to make a small modification to message generation. Based on the partition algorithm, we can observe that the partitions will not be the same across the servers due to the swap operation. However, we can still make it work by the strategies discussed below. In Algorithm 2, the senders generate two pairs of messages: $\langle r_1, m_1 \rangle$ and $\langle r_2, m_2 \rangle$, such that

- $m_1 \oplus m_2 = C(1||m, r)||r||1||m$
- $r_i \oplus m_i = m'_i = C(0||r'_i, r''_i)||r''_i||0||r'_i$, for $i \in \{1, 2\}$

The targeted user or message recipient u obtains both m_1 and m_2. The users in the same partition receives either $\langle r_1, m_1 \rangle$ or $\langle r_2, m_2 \rangle$ Due to the modified partition (Algorithm 4), the users in $G_{\hat{\tau}}$ receives r_1 and r_2. Since $r_1 \oplus r_2$ produces random value, the users in $G_{\hat{\tau}}$ cannot distinguish a real message from a message modified by one of the malicious servers. In other words, these users lose the capability of verifying message integrity. Thus, we need to modify the message generation algorithm by adding one more pair of messages denoted by $\langle r_3, m_3 \rangle$, such that $r_1 \oplus r_2 \oplus r_3 = C(0||r'_3, r''_3)||r''_3||0||r'_3$:

- $r_3 = r_1 \oplus r_2 \oplus C(0||r'_3, r''_3)||r''_3||0||r'_3$

m_3 is a randomly generated dummy value. Then Protocol 3 can be easily modified to ensure users in $G_{\hat{\tau}}$ to receive r_1, r_2 and r_3. The message reconstruction method discussed in Section 4.5 needs to be slightly changed. Instead of receiving two shares in the original protocol, now each user receives three shares, e.g., s_1, s_2 and s_3. Thus, the users try four combinations, denoted by $s_i \oplus s_j$ $(i \neq j)$ and $s_1 \oplus s_2 \oplus s_3$. If none of the combinations return a valid message, the users detect cheating by the servers.

5.4.2 General One-to-Many Communication. To support general one-to-many communication is straightforward in our design. Suppose there are n total users and Alice wants to send different messages to κ $(1 \leq \kappa \leq n)$ users denoted by $\eta = \{\eta_1, \ldots, \eta_\kappa\}$ users. Alice performs the following:

- Randomly partition the n users into κ disjoint subgroups $G^{\eta_1}, \ldots, G^{\eta_\kappa}$ and each subgroup contains around $\frac{n}{\kappa}$ users and $\eta_i \in G^{\eta_i}$ for $1 \leq i \leq \kappa$.
- In parallel and for each G^{η_i}, applies the steps given in Algorithms 1 and 2 Protocol 3.

Following these steps, the servers can learn the number of users (e.g., κ) Alice is communicating in each round. In order to hide κ, Alice can pad G^{η_i} with additional users out of the n users. These *noise* users will only receive authenticated random messages. For each round, we can randomize the number of *noise* users. In addition, to further randomize the process, Alice could distribute these κ users into multiple rounds.

5.4.3 Amortized Complexity. According to our modified protocol for one-to-many communication, in each round, a user can send and receive from many other users. The complexity of the existing solutions is described per message sent. In this case, the communication complexity for our protocol becomes (kn^2/κ) where k is the number of servers and $k = 2$ for our protocol. If κ is a fraction of n, then the complexity becomes $O(kn)$ which is the same as most mix-net based solutions. Whereas, the communication complexity of MCMix is $O(kln \log n)$ for sending one message per party assuming that each user sends message to a different recipient to avoid message collisions. The round complexity is one round for our protocol, and κ and $O(\kappa \log n)$ rounds for mix-net based solutions and MCMix respectively where κ is the subgroup size.

6 PERFORMANCE EVALUATION

The efficiency and scalability of our proposed protocol is evaluated in throughput and bandwidth with respect to the number of users in the anonymity set. We also compare our performance with MCMix [2] that offers information theoretic security.

6.1 Hardware Specifications

We have implemented the complete end-to-end logic for our protocol in C++ with no GUI elements. This was done in C++ gcc version 9.1.1, with GMP for secure random number generation along with bit operations and Boost Asio 1.69.0 for network communication. To evaluate the computational and bandwidth burden of our approach, we utilized Chameleon Cloud [23] to handle deploying instances of our image, Ubuntu 18.04.3 LTS. We ran instances each with 128GB RAM, 48 Intel Xeon E5-2670 v3 2.3GHz processors, and 10 Gbps network bandwidth. Two of these instances ran as a server to receive and distribute messages. The other 5 instances simulated clients in each experiment. These simulated clients shared a TCP connection to avoid overloading the servers, but no batching of request or message distribution was done in order to more realistically simulate the cost of many communicating clients.

6.2 Experiment Set-up and Outcomes

In each experiment, active clients generate a grouping and sends a message to themselves which allows us to easily estimate one round of sending and receiving time. The evaluation adopts the following parameters:

- Message size: fixed to 256 bytes to simulate common Twitter message size.
- The p and q values in Algorithm 2 are 2048-bit and 6144-bit respectively to accommodate the 256-byte message size.
- Latency: each client records message generation time, round-trip message travel time, and message reconstruction time.
- Bandwidth: network traffics incurred by a 256-byte message.
- Group size: the number of users in a sub-group.

Each client simulator is scheduled as a cron task to start at precisely the same time. First simulators generate and send a grouping for every active user. When all groupings have been successfully sent, the simulators generate and send an encrypted message for each user. Throughout this process, the total run time is recorded for each simulator and divided by its total simulated users to estimate the average message latency for each user.

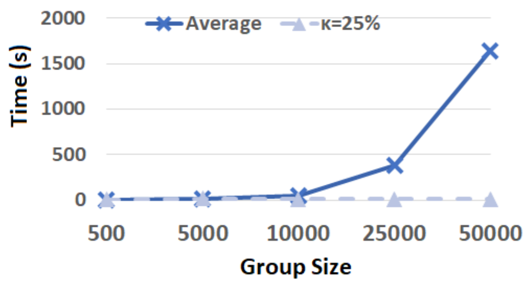

Figure 2: Message Latency for users within a group

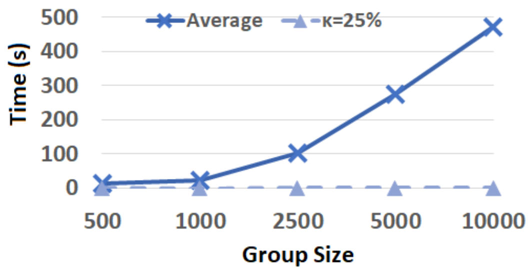

Figure 3: Message Latency for a million users active users divided into groups of given size

In the first experiment, we want to see how the group sizes affect the overall performance. Given a specific group size, the experiment simulates one round of communication where each user sends a message to one other user or a subgroup of users. The average running time for delivering a message is shown in Figure 2. The solid line indicates the time for sending one message per user, and the dashed line shows the average amortized cost for each user to send messages to a subgroup of κ users. We set κ to be 25% of each group size which is well within the $\lfloor \frac{n}{2} \rfloor$ threshold. When κ increases, the amortized cost decreases. These results show that when the group size increases, the running time increases quadratically. However, the amortized cost remains linear and is only affected by the subgroup size. Although the dashed line seems flat, the actual cost increases slightly as the group size grows.

In the second experiment, we show a situation where a million total users are divided into multiple groups of sizes varying from 500 to 10000. For example, for the group size of 500, the one million users are distributed to 2000 groups, and the users only communicate within their own groups. Since the total number of users is fixed, the larger the group size, the smaller number of groups will be produced. As shown in Figure 3, the latency increases with the size of the anonymity set. The high communication latency is in part caused by our simulation where we could only use several computers to simulate interactions among a million users. In addition, all simulations were performed by a single thread. In a real environment, the two storage servers would have very high parallelization capability (with hundreds or thousands of computers running in parallel), and users can interact with the servers simultaneously. The latency should be much less in the real world. Furthermore, as shown in Figures 2-4, the amortized latency is very small. Thus, our solution is efficient for high traffic applications.

6.3 Comparing to MCMix

As mentioned early, mix-net based protocols are generally more efficient but less secure comparing to SMC-based solutions. MCMix is the only other SMC-based approach, and it has several key differences comparing to our proposed solution as discussed in Section 2.2. We implemented MCMix using the source provided at [1] and calculated average messaging time shown in Figure 4. Note that the figure only includes message latency and not the setup phase for users. Clearly, above 10000 active users, MCMix starts to run faster than our protocol. However, our amortized cost (by setting κ to be 25% of the total users within a group) is lower, especially for large n. This experiment does not include the cost of MCMix's dialing phase which has significant cost. While dialing is not implemented at the given source, the complexity is at least as expensive as the messaging protocol itself. Figure 5 demonstrates the cost of establishing communication (or the dialing phase) in MCMix and our protocol. Clearly, our dialing protocol (i.e., group partitioning) is significantly more efficient.

As acknowledged in our theoretical analysis, the main limitation of our approach is its high bandwidth for very large anonymity set as shown in Figure 6. The bandwidth is generated by each message sent within a group of users with a specific size. As expected, the bandwidth increases as does the group size.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed a simple solution to achieve anonymous communication secure against malicious adversaries. Comparing to the existing solutions, our approaches are very easy to implement (e.g., requiring two independent servers with storage and communication capabilities) and offer additional security guarantees, such as, efficient detectability of message modification. To show the feasibility of our approach, we implemented our solution using Chameleon Cloud. The result shows the solution is salable for smaller group sizes. For subgroup communications, our protocol is efficient even for a large group.

The message complexity of our protocol for sending one message per user is still high. As a future work, we will reduce the message complexity by exploring random sampling techniques that may allow selection of a subset users to produce the needed partitions without decreasing the degree of connection anonymity. Another direction is to add additional servers to achieve fault-tolerance. That is, when one or more server fails, the rest of the servers can

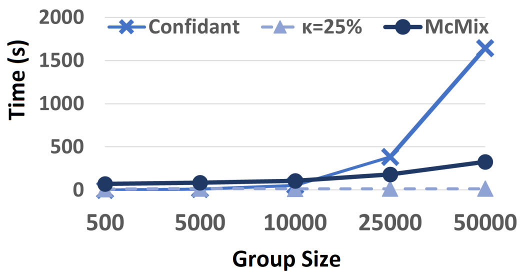

Figure 4: Message Latency in a round of MCMix, excluding setup phase

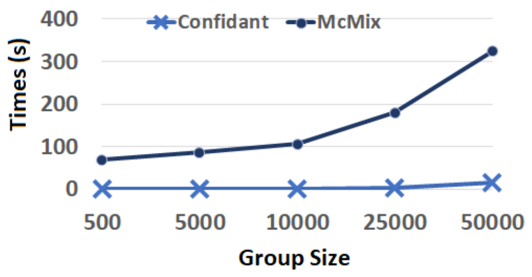

Figure 5: The average time for a user to establish information needed to message in both approaches.

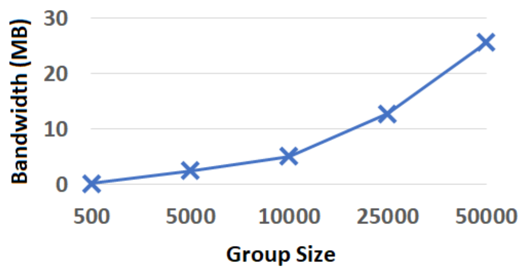

Figure 6: Server Bandwidth per message sent by a user within a group of given size

still provide anonymous communication services. Although the communication complexity will also increase, the trade-off between efficiency and security may be necessary for certain situations.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Dan Bogdanov and Cybernetica for providing the sharemind software and Chameleon Cloud for providing the computing resource. The authors would also like to thank Dr. Yuhong Nan and the anonymous referees for their valuable comments and constructive suggestions. This work is supported by the National Science Foundation under Grant No.: DGE-1946619.

REFERENCES

[1] [n.d.]. MCMix benchmarking code. https://github.com/druid/mcmix-benchmark.

[2] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous messaging via secure multiparty computation. In *The USENIX Security Symposium*. 1217–1234.

[3] Sebastian Angel and Srinath Setty. 2016. Unobservable Communication over Fully Untrusted Infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA, 551–569. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[4] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. 2013. AnoA: A Framework for Analyzing Anonymous Communication Protocols. In *2013 IEEE 26th Computer Security Foundations Symposium*. 163–178. https://doi.org/10.1109/CSF.2013.18

[5] Avrim Blum, Cynthia Dwork, Frank Mcsherry, and Kobbi Nissim. 2005. Practical privacy: The SuLQ framework. In *Proceedings of the 24^{th} ACM SIGMOD International Conference on Management of Data / Principles of Database Systems*. 128–138.

[6] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. 2014. Cloudtransport: Using cloud storage for censorship-resistant networking. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 1–20.

[7] David Chaum. 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology* 1, 1 (01 Jan 1988), 65–75. https://doi.org/10.1007/BF00206326

[8] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. 2016. cMix: Mixing with Minimal Real-Time Asymmetric Cryptographic Operations. Cryptology ePrint Archive, Report 2016/008. https://eprint.iacr.org/2016/008.

[9] David L. Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (Feb. 1981), 84–90. https://doi.org/10.1145/358549.358563

[10] Chen Chen, Daniele E Asoni, David Barrera, George Danezis, and Adrain Perrig. 2015. HORNET: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1441–1454.

[11] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. 2020. Talek: Private Group Messaging with Hidden Access Patterns. In *Annual Computer Security Applications Conference (ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 84–99. https://doi.org/10.1145/3427228.3427231

[12] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private Information Retrieval. *J. ACM* 45, 6 (Nov. 1998), 965–981. https://doi.org/10.1145/293347.293350

[13] David Cole. [n.d.]. We Kill People Based on Metadata. https://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/

[14] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An anonymous messaging system handling millions of users. *arXiv preprint arXiv:1503.06115* (2015).

[15] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. 2013. Proactively Accountable Anonymous Messaging in Verdict. In *The 22nd USENIX Security Symposium*. Washington, D.C., 147–162. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/corrigan-gibbs

[16] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router.* Technical Report. Naval Research Lab Washington DC.

[17] Cynthia Dwork. 2009. *Theory of Cryptography.* Springer Berlin / Heidelberg, Chapter The Differential Privacy Frontier.

[18] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. 2021. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 1775–1792. https://www.usenix.org/conference/usenixsecurity21/presentation/eskandarian

[19] Oded Goldreich. 2004. *The Foundations of Cryptography.* Vol. 2. Cambridge University Press, Chapter General Cryptographic Protocols.

[20] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. 2013. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Information Security and Cryptology – ICISC 2012*, Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 202–216.

[21] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 2013. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security (CCS '13)*. ACM, New York, NY, USA, 337–348. https://doi.org/10.1145/2508859.2516651

[22] Sachin Katti Jeff Cohen Dina Katabi. 2007. Information slicing: Anonymity using unreliable overlays. (2007).

[23] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. 2019. Chameleon: a Scalable Production Testbed for Computer Science Research. In *Contemporary High Performance Computing: From Petascale toward Exascale* (1 ed.), Jeffrey Vetter (Ed.). Chapman & Hall/CRC Computational Science, Vol. 3. CRC Press, Boca Raton, FL, Chapter 5, 123–148.

[24] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, New York, NY, USA, 830–842. https://doi.org/10.1145/2976749.2978357

[25] Panayiotis Kotzanikolaou, George Chatzisofroniou, and Mike Burmester. 2017. Broadcast anonymous routing (BAR): scalable real-time anonymous communication. *International Journal of Information Security* 16, 3 (June 2017), 313–326. https://doi.org/10.1007/s10207-016-0318-0

[26] E. Kushilevitz and R. Ostrovsky. 1997. Replication is Not Needed: Single Database, Computationally-private Information Retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*. IEEE Computer Society, Washington, DC, USA, 364–. http://dl.acm.org/citation.cfm?id=795663.796363

[27] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2016. Riffle. *Proceedings on Privacy Enhancing Technologies* 2016, 2 (2016), 115–134.

[28] Albert Kwon, David Lu, and Srinivas Devadas. 2019. XRD: Scalable Messaging System with Cryptographic Privacy. *CoRR* abs/1901.04368 (2019).

[29] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed private messaging immune to passive traffic analysis. In *The 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 711–725.

[30] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata.. In *OSDI*. 571–586.

[31] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and Its Application to Anonymous Communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, New York, NY, USA, 887–903. https://doi.org/10.1145/3319535.3354238

[32] Zachary Newman, Sacha Servan-Schreiber, and Srinivas Devadas. 2021. Spectrum: High-Bandwidth Anonymous Broadcast with Malicious Security. Cryptology ePrint Archive, Report 2021/325. https://ia.cr/2021/325.

[33] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The loopix anonymity system. In *The 26th USENIX Security Symposium*. 16–18.

[34] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2017. P2P Mixing and Unlinkable Bitcoin Transactions. In *NDSS*.

[35] Bruce Schneier. [n.d.]. NSA Doesn't Need to Spy on Your Calls to Learn Your Secrets. ([n. d.]). https://www.wired.com/2015/03/data-and-goliath-nsa-metadata-spying-your-secrets/

[36] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. 2017. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 423–440.

[37] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. 2015. SoK: secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 232–249.

[38] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 137–152.

[39] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Dissent in numbers: Making strong anonymity scale. In *The 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 179–182.

[40] Andrew C. Yao. 1986. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*. IEEE, 162–167.