Simple and Efficient Approach for Achieving End-to-End Anonymous Communication

Wei Jiang* Adam Bowers† Dan Lin‡

Department of Electrical Engineering and Computer Science
University of Missouri
Columbia, MO 65211, USA

*wjiang@missouri.edu
†acbqbd@mail.missouri.edu
‡lindan@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 and Private Information Retrieval techniques, can be used to achieve anonymous communication. However, the existing solutions are very complex and difficult to implement in practice. More importantly, they do not offer security against malicious adversaries who can arbitrarily deviate from normal protocol execution, e.g., refusing to participate and modifying messages exchanged in the protocol. In this paper, we propose a simple and novel approach to establishing anonymous communication, easily implementable with servers having only communication and storage related capabilities. Our approach offers stronger security guarantee against malicious adversaries without incurring a great deal of extra computation and communication costs. We formally prove the security guarantee of the proposed solution and analyze its pros and cons comparing to the existing work.

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 [11, 33]. 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 encrypted. 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 specifically which two users are actually communicating. Anonymity degree depends on the size of the group.

Several methods based on Mix-net [8] have been proposed to create anonymous communication, such as Loopix [30] and Karaoke [24]. To prevent traffic analysis [19], 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 [15, 3] which offers a weaker anonymity protection due to its small information leakage. MCMix [1] and Pung [2] offer stronger anonymity guarantee by adopting Secure Multiparty Computation (SMC) [41, 17] and Private Information Retrieval (PIR) [10, 21] techniques respectively. These solutions have complicated structures which make real implementation very challenging.

More importantly, the existing solutions are not secure against arbitrary malicious behaviors from the server end. For example, they cannot prevent or efficiently detect the servers not following the prescribed steps of the protocols. In addition, if one of the servers fails, the communication service can be adversely affected or even interrupted. The main goal of this paper is to propose simple and novel solutions to implement anonymous communication with stronger security guarantees against malicious adversaries. Even if the existing solutions could be implemented and deployed into a real environment, a third party authority can still ban people from using these services. Lavabit was shut down because it refused to provide the authorities private keys used to secure user emails [29]. Therefore, in addition to being anonymous, it is also important to protect service availability from the influence of a third party authority. To achieve this, two properties can help:

- *Service obliviousness*: To prevent an authority from forcing the service providers to disclose user information, the service usage should disclose minimal information as to what the service is being used for. Ideally, the service providers should not even know if they have actually provided anonymous communication services to a group of users.
- *Cost of Closing*: The idea was initially presented in Cloudtransport [5] which argues that if the services of a company relied upon to provide a nation's infrastructure or enable business, like Amazon, can be used to provide anonymous communication, a country will have to hurt itself to shut down the communication network. That is, to stop the anonymous communication, services the country needs to run would be shut down or adversely affected.

To achieve a certain degree of service obliviousness and increase the cost of closing, we should only need rudimentary functionalities (e.g., message passing) from the servers in order to embed anonymous communication into the existing services without being detected easily.

1.1 Our Main Contribution

In this paper, we propose novel broadcasting and secret sharing based multi-server protocols for achieving anonymous communication, which possess the following desirable properties:

- *Stronger security*: Our proposed protocols are information theoretically secure when at most t out of n servers collude and behave maliciously to violate data confidentiality and connection anonymity. Specific malicious behaviors may include: modifying data, refusing to provide services (e.g., sending messages), disclosing user data, etc.
- *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 they are also lightweight on the user end. (3) No communications are needed between the servers. (4) Only two non-colluding servers are needed to guarantee message confidentiality and communication anonymity.
- *Asynchronous*: Users do not need to be online all the time, and no synchronization is needed between parties in communication.
- *Service obliviousness*: Due to the simplicity of our design, it is possible to hide anonymous communication service into existing cloud storage and social networking services.

1.2 Design Overview

The proposed solution requires at least two servers to achieve the security properties, and more servers are required against malicious adversaries and to increase the odds that one of the servers is honest. By utilizing secret sharing [34], symmetric encryption (such as AES) is no longer necessary to establish secure end-to-end communication. As shown in the figure, 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 pseudo-random (depending on the randomness of the generator), each alone does not leak any information 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 are able to retrieve the actual message m.

Our approach achieves data confidentiality against the service providers as long as one of the server is honest. However, the approach alone is not sufficient to achieve end-to-end secure and anonymous communication since everyone connected to Bob will receive his message. As a result, the key challenge is how to use secret sharing and the multi-server framework to build an anonymous communication network between any two users in a group. Along this line, we will present three protocols which trade-off between efficiency and security. At first glance, broadcasting may seems expensive. For Bob to send a message to Alice within a group of n users, the network will incur $O(kn)$ messages in our approach where k is the number of servers. Nonetheless, comparing to the existing solutions, our approach has similar complexity but possesses stronger security guarantees. In addition, the proposed approach is much easier to implement. Detailed comparison with the existing solutions is given in Section 2 and 5.3.

1.3 Threat Model

Similar to the existing work, we assume that (1) users are honest and follow the protocol, and (2) there exist authenticated communication channels between the users and the servers. However, unlike the existing solutions, we assume that (3) t out of k servers, with unlimited computing power, can behave maliciously or any way they want to break the security guarantee of the proposed protocol. In summary, our proposed three protocols provide the following security guarantees against a group of t malicious adversaries:

- *Anonymity and Confidentiality*: Connection anonymity and message confidentiality are achievable for up to t malicious servers, and t can be as large as $k - 1$ (under the additive secret sharing scheme), where k is the total number of servers. The degree of anonymity is bounded by the user's group size. Suppose the group size is n, then the anonymity degree is $\frac{1}{n}$ which is the same as the existing solutions. In addition, if we assume each pair of users share a secret key, then message confidentiality is guaranteed regardless how many servers collude. All three protocols achieve connection anonymity and message confidentiality.

- *Availability and Integrity*: Service availability and message integrity go hand in hand; that is, availability is irrelevant when the service cannot deliver the correct message. In our solutions, the actual message is guaranteed to be delivered to the intended user when $t < \frac{k}{3}$ under Protocol 2 and $t < \frac{k}{2}$ under Protocol 3.

- *Detectability*: As long as 1 out of k servers follows the protocol, violation of message integrity caused by any malicious behaviors can be detected by the message recipient. Protocol 1 achieves this detectability.

Note that to our knowledge, none of the existing solutions can formally guarantee service availability, message integrity and detectability of malicious behaviors without using digital signature schemes.

1.4 Organization

The rest of the paper is organized as follows: Section 2 presents the existing anonymous communication platforms and discusses their limitations. Section 3 presents the technical background on cryptographic primitives needed to construct the proposed protocols whose details are presented in Section 4. Section 5 provides security and complexity analyses, comparison to the existing solutions, as well as practical considerations of the proposed protocols. Section 6 concludes the paper with future research directions.

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. Mix-net [8] based solutions, Tor [14], Hornet [9], and information slicing [20], 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 [19], 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 [25], Stadium [36], and Karaoke [24] were published in series. They achieve Differential Privacy [15, 3] 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. Loopix [30] adopts time delay to hide traffic patterns and a number of servers to improve system response time. cMix [7] also employs mix-net design but can shuffle messages faster by not using public key operations. These solutions assume at least one server is honest and offer a weaker security because Different Privacy, by definition, leaks a small amount of information [1].

The DC-Net [6] inspired approaches, DiceMix [32], Verdict [13], Dissent-AT [40] and Riposte [12], 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 do not support anonymous 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 needs to decrypt every message. If one of the messages can be decrypted successfully, then the user know he or she is the receipt. This approach is not very efficient since all keys and messages pairs have to be examined to retrieve the actual messages.

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. MCMix [1] utilizes oblivious sorting algorithm [18] 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 one server is honest, anonymity is guaranteed and bounded by the group size. For instance, if the group size is n, the anonymity is $\frac{1}{n}$. Pung [2], a PIR based technique, offers the same anonymity, but it only requires one server. More servers can reduce the number of messages required to exchange one message. Riffle [22] 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 also provides verifiability regarding if the servers behaved correctly during the message shuffling phase, but it does not guarantee that the users will receive the correct messages.

Like our approach, Cloudtransport [5] 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.

2.1 Limitations of the Existing Work

Here we highlight the key limitations of the existing solutions that provide one-to-one or pairwise anonymous communication. Later, we will discuss how these limitations are addressed by our proposed protocols.

2.1.1 Weaker Security Guarantee

As mentioned in Section 1, the existing approaches guarantee anonymity when one of the servers is honest. However, they do not possess the following properties:

1. *Fault-tolerance*: If one server fails or refuses to participate, normal services can be disrupted. For mix-net based approaches, by using more servers and dividing users into sub-groups (e.g., Loopix and Karaoke), when failure occurs in a sub-group of servers, users in other groups can still send and receive messages. Nevertheless, the degree of fault-tolerance cannot be formally proven. In addition, the degree of anonymity is also reduced due to smaller group sizes. Although by adopting the Shamir secret sharing scheme, SMC and PIR based approaches could be theoretically (not straightforward in practice) made fault-tolerant, the computation complexity will increase dramatically which makes them even more impractical.

2. *Message integrity*: If a server modifies a message, the recipient will not receive the correct message. We are not aware of any efficient ways that can easily incorporated into the existing solutions to prevent this from happening or guarantee the correct messages will be received.

3. *Detecting malicious behaviors*: Using digital signatures under the existing solutions, it is possible for a user to detect if the received messages have been modified. However, public key infrastructure adds another layer of system complexity, and key management becomes complicated with a large group of users whose memberships change frequently. Our solution for detecting malicious behaviors is straightforward without the complexity of key management.

2.1.2 Round Synchronization

Most of the existing solutions are synchronous in that all users are required to participate or be online in each round. One message can be sent and received during each round. There are few exceptions: assuming sufficient number of clients communicating at any given time for anonymity, Loopix allows the protocol to proceed. It basically divides a group into smaller groups, and synchronization is still required within each sub-group. Users in DiceMix do not need to send message at the same time but communication is held till all participate, or they timeout and their messages will not be sent. Round synchronization makes the protocols hard to implement and reduces their usability as well.

2.1.3 Many-to-One Communication

By “many-to-one”, we mean that multiple users want to send messages to the same user. For mix-net based solutions, a user Bob needs to know the user and his or her intention to send Bob a message, and vice versa. Then the two users, from a shared key, establish an initial secret dead-drop/mailbox location in one of the servers. In each round, their messages will reach the current dead-drop location, be swapped and sent back to each user. It is not realistic to assume a user knows who the specific senders are in advance, and the user should only know the senders are from his or her group. In this case, to implement many-to-one communication, each user has to create $n-1$ dead-drop locations to prevent message collision which increases the system complexity dramatically without providing extra security. Under MCMix, when multiple users try to communicate with the same user Bob, some of the users will not be able to send message to Bob. Similarly, it is not obvious how PIR based approaches handle many-to-one communication.

2.1.4 Anonymous Sub-Group Communication

It is not efficient for the existing solutions to enable anonymous sub-group communication. Suppose the sub-group size is β. Then the existing solutions need to go through β rounds and incur β times of the baseline complexity. However, if $\beta \leq \lfloor \frac{n}{k} \rfloor$, our protocols can handle sub-group communication in one round without incurring any additional complexity.

3 Technical Background

Depending on the protocols, we utilize two types of secret sharing schemes: additive and Shamir [34]. Assuming m is a non-negative integer, to secretly share m among k servers $S_1, \dots, S_k$, first randomly select $k-1$ values $m_1, \dots, m_{k-1}$ from Z_N. Then m_k can be computed according to $m = m_1 + \cdots + m_k \mod N$. The secret share m_i is stored at S_i for $1 \leq i \leq k$. If Alice has permission to access the shares, S_i will send m_i to Alice. Combining the k secret shares together, Alice can learn m. Because $m_i, \dots, m_{k-1}$ are randomly generated, they do not leak any information regarding the actual message m. In addition, due to the "+" and "mod" operations, m_k is also randomly distributed in Z_N. That is, m_k alone does not reveal any information regarding m. Thus, $m_1, \dots, m_k$ are called secret shares of m.

The XOR operator $\oplus$ can also be used to generate secret shares in our protocol. Suppose the message m is l-bit long, to secretly share m, randomly generate $k-1$ l-bit random strings $m_1, \dots, m_{k-1}$. Then m_k can be computed according to $m = m_1 \oplus \cdots \oplus m_k$. In Protocol 1, both schemes work. We will make our choice more explicit when presenting the proposed protocols.

3.1 Shamir's Secret Sharing

A well known threshold based secret sharing scheme was proposed by Shamir [34]: an algorithm for dividing a secret into parts among multiple participants/servers such that each participant gets its own unique part. The algorithm requires either some of the parts or all of them to reconstruct the secret. The scheme allows setting a threshold on the number of parts required to construct the secret. To create a (t, k) threshold scheme to share a secret $m \in \mathcal{F}$, where $\mathcal{F} = \mathbb{Z}_p$ for some prime p, we can follow the steps given below:

- Choose random t coefficients $r_1, \dots, r_t \in \mathcal{F}$
- Construct polynomial, $f(x) = r_t x^t + \cdots + r_1 x + m$
- Compute shares $m_i = (i, f(i))$, for $1 \leq i \leq k$.

Given at least $t+1$ of these shares indexed by $\{i_1, \dots, i_{t+1}\}$, the secret can be re-constructed using:

$$m = \sum_{j=1}^{t} f(i_j) \cdot \lambda_{i_j}, \text{where } \lambda_{i_j} = \prod_{z=1}^{t+1, z \neq j} \frac{i_z}{i_z - i_j} \tag{1}$$

3.2 Pedersen Commitment

To detect if a sever tampered with a user's message, our solution adopts Pedersen commitment [28]. Here we present an overview of this commitment scheme. Let p and q denote large primes and $q|p-1$. $\mathbb{G}_q$ is the unique subgroup of $\mathbb{Z}_p^*$ of order q, and g is the generator of $\mathbb{G}_q$. The discrete logarithm of $x \in \mathbb{G}_q$ to the base g is denoted by $\log_g x$ and believed to be hard when q is sufficiently large. To commit to a message $m \in \mathbb{Z}_q$, suppose that $\log_g x$ is unknown and r is randomly selected from $\mathbb{Z}_q$, the commitment is computed as:

$$E(m, r) = g^m h^r \tag{2}$$

The commitment can be opened by disclosing m and r. It has been proven that $E(m, r)$ is uniformly distributed in $\mathbb{G}_q$ and does not disclose any information about m.

4 The Proposed Protocols

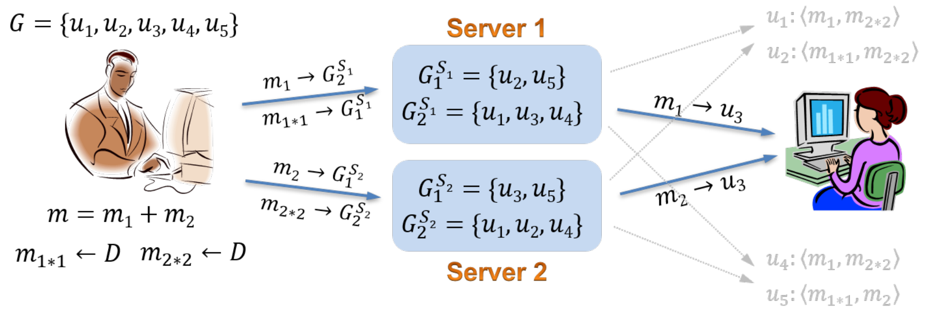
In this section, we present three protocols based on secret sharing and the multi-server framework. An overview of each protocol is given below:

- *Protocol 1*: k-server protocol uses additive secret sharing for maximum efficiency and $k \geq 2$. As long as one server is honest, the protocol preserves anonymity and confidentiality. Also, by utilizing the Pedersen commitment, the protocol achieves detectability.
- *Protocol 2*: There are k servers, where $k > 2$. The protocol adopts the Shamir secret sharing scheme, and at most t parties are malicious. It guarantees anonymity, confidentiality, availability and integrity when $t < \frac{k}{3}$ when the actual messages are reconstructed with the Berlekamp and Welch algorithm [39].
- *Protocol 3*: There are k servers, where $k > 2$. The protocol adopts the Shamir secret sharing scheme and the Pedersen commitment. At most t parties are malicious. The protocol guarantees anonymity, confidentiality, availability and integrity when $t < \frac{k}{2}$.

Each protocol consists of two phases: session key generation and communication. 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 between the two users.

4.1 Illustration of the Main Idea

We consider a group of n users registered with k servers, and their IDs are unique and cannot be linked across these servers. To enable secure and anonymous pairwise communication using broadcasting, the main idea is to divide this group of users into k randomly generated sub-groups. Place the message recipient into different sub-groups across the k servers. The other users will be in the same sub-groups. In this way, it becomes possible that only the recipient gets k shares of the actual message, and the other users get at most one share of m.



$G = \{u_1, u_2, u_3, u_4, u_5\}$
$m = m_1 + m_2$
$m_{1*1} \leftarrow D \quad m_{2*2} \leftarrow D$
$m_1 \rightarrow G_2^{S_1}$
$m_{1*1} \rightarrow G_1^{S_1}$
$m_2 \rightarrow G_1^{S_2}$
$m_{2*2} \rightarrow G_2^{S_2}$
Server 1
$G_1^{S_1} = \{u_2, u_5\}$
$G_2^{S_1} = \{u_1, u_3, u_4\}$
$G_1^{S_2} = \{u_3, u_5\}$
$G_2^{S_2} = \{u_1, u_2, u_4\}$
Server 2
$m_1 \rightarrow u_3$
$m_2 \rightarrow u_3$
$u_1: \langle m_1, m_{2*2} \rangle$
$u_2: \langle m_{1*1}, m_{2*2} \rangle$
$u_4: \langle m_1, m_{2*2} \rangle$
$u_5: \langle m_{1*1}, m_2 \rangle$

Figure 1: Simple illustration

Here we provide a concrete example, highlighted in Figure 1, to illustrate the above idea. Suppose there are two servers (S_1 and S_2) and 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 one of the five users:

- $G \leftarrow \pi(G)$, where π is a random permutation.
- Partition G into two sub-groups with roughly equal size, denoted by G_1 and G_2 respectively.

Suppose $u \in G$ is the 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 $u \in G_1$. Swapping u with a randomly chosen element in G_2, and 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 $G_1 = \{u_2, u_5\}$ and $G_2 = \{u_1, u_3, u_4\}$. The following partition is possible by swapping $u_3 \in G_2$ with $u_2 \in G_1$:

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

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:

- $G_1^{S_1} = 01001$ and $G_2^{S_1} = 10110$
- $G_1^{S_2} = 00101$ and $G_2^{S_2} = 11010$

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_{i*j} indicates randomly generated values from the same distribution of the secret shares of m, where i is the server index and j is the sub-group index at S_i.

- $m_1 \leftarrow_R \{0,1\}^l$ and $m_2 \leftarrow m \oplus m_1$
- $m_{1*1} \leftarrow_R \{0,1\}^l$ and $m_{2*2} \leftarrow_R \{0,1\}^l$

In order for u_3 to receive m, Bob sends each server the following messages.

- $m_1 \rightarrow \langle S_1, G_2^{S_1} = \{u_1, u_3, u_4\}\rangle$
  $m_2 \rightarrow \langle S_2, G_1^{S_2} = \{u_3, u_5\}\rangle$
- $m_{1*1} \rightarrow \langle S_1, G_1^{S_1} = \{u_2, u_5\}\rangle$
  $m_{2*2} \rightarrow \langle S_2, G_2^{S_2} = \{u_1, u_2, u_4\}\rangle$

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

- u_1: $\langle m_1, m_{2*2}\rangle \rightarrow m_1 \oplus m_{2*2}$
- u_2: $\langle m_{1*1}, m_{2*2}\rangle \rightarrow m_{1*1} \oplus m_{2*2}$
- u_3: $\langle m_1, m_2\rangle \rightarrow m_1 \oplus m_2$
- u_4: $\langle m_1, m_{2*2}\rangle \rightarrow m_1 \oplus m_{2*2}$
- u_5: $\langle m_{1*1}, m_2\rangle \rightarrow m_{1*1} \oplus m_2$

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 4.3. The generated disjoint sub-groups/partitions serves as a session key. Our formal group partition algorithm is presented next.

Algorithm 1 Group-Partition$(G, u, k) \to G^{S_1}, \ldots, G^{S_k}$

```
Require: $G$ is a group of $n$, $u \in G$ is the message recipient, $k$ denotes the number of servers.
 1: $d \leftarrow \lfloor \frac{n}{k} \rfloor$
 2: $G \leftarrow \pi(G)$
 3: for $i = 1$ to $k$ do
 4:     $G_i = \emptyset$
 5:     for $j = (i-1) \cdot d + 1$ to $i \cdot d$ do
 6:         $G_i = G_i \cup \{u_j\}$
 7:     end for
 8:     while $i = k$ and $j \leq n$ do
 9:         $G_i = G_i \cup \{u_j\}$
10:     end while
11: end for
12: $G \leftarrow \langle G_1, \ldots, G_k \rangle$
13: $G^{S_1} \leftarrow G$
14: $j \leftarrow$ Group-Index$(u, G)$
15: for $i = 0$ to $k-2$ do
16:     $j' = (j + i \mod k) + 1$
17:     $u_* \leftarrow_R G_{j'}$
18:     $\langle \hat{G}_j, \hat{G}_{j'} \rangle \leftarrow$ Swap$(u_j, u_*)$
19:     $G^{S_{i+2}} \leftarrow G - \langle G_j, G_{j'} \rangle + \langle \hat{G}_j, \hat{G}_{j'} \rangle$
20: end for
```

4.2 Group Partition

Here we generalize the steps given in the previous example. The key steps are given in Algorithm 1. The number of sub-groups is the same as that of the servers. Thus, d at step 1 denotes the size of the first $k-1$ partitions if n is not divisible by k. Before partitioning starts, G is randomly permuted. Steps 3-12 produce the partitions by scanning G from the first element (indexed by 1) to the last one (indexed by n). Partition 1, denoted by G_1, consists of the first d elements of G, and so on. Steps 8-10 take care of the situation where n is not divisible by k. That is, the last partition, G_k, will contain more than d elements.

Without introducing an excessive number of notations, we also use G to represent the set of produced partitions. As indicated at step 13, the partitions for server S_1 is the same as G from which the rest of the $k-1$ different partitions will be produced according to steps 14-20. The index j, at step 14, indicates the sub-group/partition in G where u belongs. The main step here is to swap u with a randomly chosen element in the next sub-group which has yet to be considered. This swap modifies two sub-groups G_j and $G_{j'}$ resulting $\hat{G}_j$ and $\hat{G}_{j'}$. At step 19, G_j and $G_{j'}$ will be replaced by $\hat{G}_j$ and $\hat{G}_{j'}$ to create a new partition for the next server.

4.2.1 Adopting Unlikable User IDs

If each server has the same set of user IDs, the message recipient u can be identified by its distribution in the sub-groups when the servers collude. To prevent this, the users IDs at each server must be independent and cannot be linked directly based on the IDs themselves which leads to identical partition distribution across all servers. As a result, when the servers collude, they will not be able to break the security guarantee based on the ID and partition information alone. The above attack only considers server collusion combined with group partition information, and additional attacks will be addressed in subsequent sections. Note that as long as the user keeps a local mapping on the IDs among the servers, the same group partitioning algorithm is applicable with very little changes.

4.3 Message Construction and Distribution

As stated previously, we propose three protocols to achieve anonymous communication, and the key difference among the three protocols is how the messages are constructed. As illustrated in the example, the users in the same partition receive the same message, and the number of message a user receives is the same as the number of servers, denoted by k. In addition, the number of partitions is also equal to k. Therefore, the total number of messages that need to be generated by a sender is k^2, k 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 other users know the messages they received cannot be the actual messages.

To solve the above problems, we append a special message $\star$ of q bits indicating the next l-bits represent a valid message. In other words, when the prefix of a message is not $\star$, the rest of the message is ignored by the recipient. The larger the q, the less likely the prefix of a message constructed from a subset of $t+1$ or k (depending on the underlying secret sharing schemes) randomly generated messages matches the $\star$ prefix of a legitimate message. Alternatively, we can use the Pedersen commitment to distinguish real or random messages. Following the steps given in Section 3.2, the message m is appended with $E(m,r)||r$. However, using the Pedersen commitment will lead to a longer message and incur more computation at the user end. The main steps are summarized below:

- Without the Pedersen commitment:
  - $\vec{m} = \langle m_1, \ldots, m_k \rangle \leftarrow \text{Gen-Shares}(\star||m, k)$
- With the Pedersen commitment:
  - $\vec{m} = \langle m_1, \ldots, m_k \rangle \leftarrow \text{Gen-Shares}(E(m,r)||r||m, k)$

The Gen-Shares function produces secret shares of m according to a specific secret sharing scheme which will be clarified in our proposed protocols. In practice, to make sure collision with the $\star$ prefix never happens, during message construction, only random messages that do not cause the collision will be chosen.

4.3.1 Distributing Shares

Once the partitions and secret shares are produced, during communication phase, the shares are distributed to the users based on the steps presented in Algorithm 2. We need to pay special attention to the k shares of m, each of which is distributed to one partition at a particular server as specified by the steps 1-6. During the rest of the steps, random messages are produced according to the distribution of the secret shares of m to make sure that all messages sent to the users are identically distributed.

To prevent other users from receiving m, these users shall not receive more than t shares of m. The way that the partitions are formed and the shares are distributed enforces that only the message recipient obtains more than t shares. We have the following claim:

Claim 1. *Suppose the group partitions are generated using Algorithm 1 and the shares of m and other random values are distributed according to Algorithm 2. Then only u can receive all k distinct shares of m, and the other users receive at most one of the k shares of m.*

Proof. To prove the claim, we classify the users into three categories: u, $\{u_i\}$ and $\{u_*\}$ where u is the targeted message recipient, $\{u_i\}$ denotes the set of users who remain in the same partition across all servers, and $\{u_*\}$ denotes the users who were swapped with u to generate the partitions. According to how m_i is distributed, 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 only one valid share of m. This is because u_* is swapped with u to create a new partition at each server. Regardless which partition u_* belongs, u_* and u are never in the same partition at any server. Thus, users in $\{u_*\}$ do not receive any shares of m. □

Algorithm 2 Dist-Msg($\vec{m}, G^{S_1}, \dots, G^{S_k}$)

```
1: $j \leftarrow$ Group-Index$(u, G)$
2: Send $m_1$ to $G_j^{S_1}$
3: for $i = 0$ to $k - 2$ do
4:     $j' = (j + i \mod k) + 1$ and $i' = i + 2$
5:     Send $m_{i'}$ to $G_{j'}^{S_{i'}}$
6: end for
7: $\mathcal{D} \leftarrow$ distribution of secret shares of $m$
8: for $i = 1$ to $k$ do
9:     for $j = 1$ to $k$ do
10:        if $u \notin G_j^{S_i}$ then
11:            $m_{i*j} \leftarrow_R \mathcal{D}$
12:            Send $m_{i*j}$ to $G_j^{S_i}$
13:        end if
14:    end for
15: end for
```

4.4 Protocol Variations

Group partitions and share distribution consist of the main part of the protocols, and they remain the same. The key differences among these protocols are:

- The specific scheme used to generate the shares with or without the Pedersen commitment.
- How the shares are combined to produce the actual message and how the Pedersen commitment is used in the process.

Next we present share generation and message re-construction for each protocol.

4.4.1 Protocol 1

Under this protocol, the shares are constructed with additive secret sharing scheme, with or without the Pedersen commitment. Therefore, message reconstruction considers two cases:

- Without the Pedersen commitment: the recipient u has to receive all k shares of m. XORing all these shares will produce $\star||m$. When the prefix $\star$ is observed, u knows m is the actual message.
- With the Pedersen commitment: the recipient u still needs to receive all k shares of m. XORing all these shares will produce $E(m, r)||r||m$. u extracts m and r and verifies if $E(m, r)$ is a commitment of m. u accepts m if the verification passes.

4.4.2 Protocol 2

Implementation of this protocol is straightforward, and the protocol only uses the Shamir secret sharing scheme. The message reconstruction is different from the one we presented in Section 3. In this protocol, we use the Berlekamp and Welch algorithm [39] to reconstruct m. The algorithm works as follows:

- Suppose that the shares $\epsilon_1, \dots, \epsilon_{t'}$ are maliciously modified by t' servers, and $\mathcal{E}(x) = (x - \epsilon_1) \cdots (x - \epsilon_{t'})$, where $t + 2t' + 1 = k$. In addition, let $\mathcal{P}(x)$ be the polynomial that secretly shares m. Define $\mathcal{Q}(x) = \mathcal{P}(x)\mathcal{E}(x)$, which is a polynomial of degree $t + t'$.
    - $\mathcal{Q}(x) = a_{t+t'}x^{t+t'} + \cdots + a_1 x + a_0$
    - $\mathcal{E}(x) = x^{t'} + b_{t'-1}x^{t'-1} + \cdots + b_1 x + b_0$

- Since $\mathcal{Q}(i) = m_i\mathcal{E}(i)$, from the k shares $m_1, \ldots, m_k$, a system of k linear equations can be established with k unknowns: $a_{t+t'}, \ldots, a_0$ and $b_{t'-1}, \ldots, b_0$. Note that any missing shares (up to t' in number) can be replaced with a value from the domain of these shares.
- Solving the k unknowns leads to $\mathcal{Q}(x)$ and $\mathcal{E}(x)$, from which $\mathcal{P}(x)$ is obtained by $\frac{\mathcal{Q}(x)}{\mathcal{E}(x)}$.

4.4.3 Protocol 3

This protocol combines the Shamir secret sharing and the Pedersen commitment. Suppose u obtains k' shares where $t < k' \leq k$. Message reconstruction requires the following steps:

- Select one of $\binom{k'}{t+1}$ share combinations from the k' received shares $m_1, \ldots, m_{k'}$.
- Based on previously chosen $t+1$ shares, adopting the message reconstruction steps (given in Section 3) derives $E(m, r)||r||m$:
  - If the commitment verification passes, then accept m and return *true*.
  - Otherwise, go to the first step and repeat.
- If all $\binom{k'}{t+1}$ combinations are tried without passing any commitment verification, then return *false*.

5 Protocol Analyses and Extensions

In this section, we present our detailed security analysis of each protocol according to criteria given in Section 1.3. Additionally, we will also discuss protocol complexity, practical issues of sending large messages and how to efficiently handle many-to-one and one-to-many (sub-group) anonymous communications.

5.1 Security Analysis

Due to the simplicity of the design, to prove the security of our protocols is straightforward. The key functionality required at the servers is passing the messages around and no additional computations are needed. As a result, the security guarantee of each protocol is directly related to those of the underlying secret sharing and commitment schemes. Note that the proposed protocols achieve pairwise communication according to Claim 1. Next we prove several claims following the previously adopted notations and assume that the servers have unlimited computing power with at most t of them colluding maliciously.

Claim 2. *All three proposed protocols achieve message confidentiality and connection anonymity of* $\frac{1}{n}$*, up to* t *out of* k *malicious servers, where* $1 \leq t \leq k-1$.

Proof. Because $t+1$ shares are needed to reconstruct the message m under the secret sharing schemes, t colluding servers are not able to derive m. Thus, confidentiality of m is preserved. In addition, if the users IDs are not linkable across the servers, the t colluding servers cannot identify the message recipient u since every user receives t random messages from the view point of these servers and every user is equally likely to be u. On the other hand, if the colluding servers were able to reconstruct m, then u would be identified. This could only happen if the number of colluding servers is more than t. □

Claim 3. *Protocol 2 achieves service availability and message integrity when* $t < \frac{k}{3}$.

Proof. In our description of the Berlekamp and Welch algorithm, $t + 2t' + 1 = k$ where t is the degree of the polynomial and t' is the total errors (caused by malicious behaviors or network faults) that can be corrected. Since t also represents the number of colluding servers, we can set $t = t'$. Thus, $3t+1 = k$ implies $t < \frac{k}{3}$. □

Claim 4. *Protocol 1 achieves detectability with probability* $1 - \frac{1}{q}$ *when at least one server is honest and the Pedersen commitment is used.*

Proof. When adopting the Pedersen commitment, the shares are generated based on $E(m,r)||r||m$. As stated in Section 3.2, in order to produce the commitment, m is in $\mathbb{Z}_q$. Thus, all three values $E(m,r)$, r and m are bounded by q. In this proof, we consider the worst case where there are $k-1$ malicious servers indexed by $S_1, \ldots, S_{k-1}$. In addition, two cases guide our analysis:

- The malicious servers want the recipient u to receive and accept a specific $\hat{m}$ whose k shares are denoted by $\hat{m}_1 \ldots, \hat{m}_k$.
- The malicious servers want the recipient u to receive and accept any $\hat{m}$ that is different from m.

In either case, the malicious servers modify the first $k-1$ shares to replace the actual shares of m received from the sender. As a result, u receives $\hat{m}_1 \ldots, \hat{m}_{k-1}, m_k$. For u to accept a specific $\hat{m}$ as a legitimate message, it must be the case that $\hat{m}_k = m_k$. Since m_k is uniformly random, $Prob(\hat{m}_k = m_k) = \frac{1}{|m_k|} = \frac{1}{q^3}$. 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, being a legitimate message is given by $\frac{q^2}{q^3} = \frac{1}{q}$ where q^2 is the number of valid m and r pairs. Thus, 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. □

Claim 5. *Protocol 3 achieves achieves service availability and message integrity with probability being approximately* $1 - \frac{2^k}{\sqrt{\pi k/2 \cdot q}}$ *when* $t < \frac{k}{2}$.

Proof. This proof is similar to that of Claim 2. To avoid redundancy, we directly dive into the worst case where the malicious servers want to trick u into accepting any message $\hat{m}$ other than m. Suppose there are only $t+1$ honest servers out of k servers. Then there are $\binom{k}{t+1}$ combinations to consider, only one of which consists of shares from the honest servers. For the attack to be successful, at least one of $\binom{k}{t+1} - 1$ combinations produces a message that passes the verification. Since $\binom{k}{t+1}$ is maxed when $t+1 \approx \frac{k}{2}$, the probability that none of these message passes the verification is bounded by:

$$\begin{aligned}\left(1 - \frac{1}{q}\right)^{\binom{k}{t+1}-1} &\approx 1 - \frac{\binom{k}{t+1} - 1}{q} \\ &\approx 1 - \frac{\frac{2^k}{\sqrt{\pi k/2}}}{q} = 1 - \frac{2^k}{\sqrt{\pi k/2} \cdot q}\end{aligned}$$

In the above, we approximate $\binom{k}{k/2}$ by $\frac{2^k}{\sqrt{\pi k/2}}$ and $\left(1 - \frac{1}{q}\right)^x$ by $1 - \frac{x}{q}$ since q is much greater than x. □

5.2 Complexity Analysis

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. Table 1 summarizes the complexity for group partition. Because other users are not involved, the table only lists the complexities for the sender and the server. In the columns, three metrics are listed corresponding to local computation (Local Comp), the number of messages (# of Msgs) and message size in bits (Bits / Msg). The local computation does not include message processing time (e.g., send, receive and store messages) due to that fact that the time varies dramatically according to the message size. However, the message processing time can be estimated from the number of messages and the size of each message.

On the sender side, the local computation complexity is linear on the number of users or user IDs in the group. More precisely, the users IDs are scanned three times at most to permute, partition the IDs and swap the memberships. We adopt two formats or representations for group memberships: set and bit-vector. For each server, k partitions are produced, each of which is a set of user IDs. Assuming each set can be packed into one message, then the sender sends k messages per server. Thus, the total number of messages sent is k^2 regardless of the format. Under the set representation, each set or message contains at most $\frac{n}{k}$ IDs. If

Table 1: Protocol Complexities - Group Partitions

Parties	Formats	Local Comp	# of Msgs	Bits / Msg
Sender	Set	O(n)	k^2	$\frac{n}{k} \cdot d$
	Bit-vec	O(n)	k^2	n
Server	Set	O(1)	k	$\frac{n}{k} \cdot d$
	Bit-vec	O(1)	k	n

each ID is d-bit long, each message is $\frac{n}{k} \cdot d$ bits. Multiplying it with k^2, we get the total number of bits sent by a sender. If bit-vector is used for representing memberships, it is possible to represent each set with n bits. Thus, the total number of bits is $k^2 n$, and bit-vector representation is more efficient when $d > k$.

On the server side, because we separate the message processing time and the server does not perform direct computations on the user IDs, we treat its local computation time as a constant. Each server receives k messages, and message sizes are calculated the same way as before. The total message complexity for the entire system is the same as those of the sender, and the whole protocol requires only one round of communication between the sender and the servers.

Table 2: Protocol Complexities - Sending One Message

Parties	Measures	Proc. 1	Proc. 2	Proc. 3
Sender	# of Msgs	k^2	k^2	k^2
	Bits / Msg	$l' + l$ ($3l_q$)	l_q	$3l_q$
	Local Comp	O(k)	O(k^2)	O(k^2)
User	# of Msgs	k	k	k
	Bits / Msg	$l' + l$ ($3l_q$)	l_q	$3l_q$
	Local Comp	O(k)	O(k^3)	$O\left(k^2 \cdot \binom{k}{t+1}\right)$
Server	# of Msgs	$k + n$	$k + n$	$k + n$
	Bits / Msg	$l' + l$ ($3l_q$)	l_q	$3l_q$
	Local Comp	O(1)	O(1)	O(1)

To send one message to a recipient, the complexities of the proposed protocols are summarized in Table 2. The common measures presented in the table are the number of messages (# of Msgs), the number of bits per message (Bits / Msg) and local computation (Local Comp). As discussed before, the sender needs to generate k^2 messages and k of them are shares of m. The senders sends k messages to each server. Thus, the total number of messages sent is k^2 for all three protocols. For the other users (including the targeted recipient), each of them receives k messages which is also the message complexity for the users. Each server receives k messages from the sender and sends them to n users. Therefore, the message complexity for each server is $k + n$.

The message sizes are different for each protocol. For the first protocol, there are two ways to generate the shares: with or without the Pedersen commitment. Without the commitment, assuming l' bits to represent the $\star$ message, and l bits to represent the actual message, then the message size is $(l' + l)$ bits. On the other hand, with commitment, let l_q denote the size of q (the domain of the commitment). Since the commitment, the random value and the message are bounded by q, the message size is $3l_q$ as shown between the parentheses in the table. The second protocol does not require the commitment and the shares are bounded by q, so the message size is l_q bits. The third protocol adopts the Pedersen's commitment; thus, the message size is $3l_q$ bits. In practice, a message can be very large, and it is not practical to bound the commitment and the message in the same domain. In Section 5.4, we discuss simple modifications to the share generation for large messages in combination of the Pedersen commitment.

In our analysis for local computation, we exclude the complexity associated with sending and receiving messages because it can be estimated from the message complexity. Although the basic operations for each protocol are different, since k is generally very small, we ignore the computation discrepancy among these operations. Thus, here we only provide asymptotic complexity in terms of k. The servers merely perform sending and receiving messages, so their location computations are constant for all three protocols, denoted by O(1) in the table. Next our analysis primarily focuses on the sender and other users.

For Protocol 1, the main operation is XOR, and the number of this operations is precisely determined by k. As a result, the local computation complexity for the sender and other users is bounded by $\mathrm{O}(k)$. Both Protocols 2 and 3 adopt the Shamir secret sharing scheme related to polynomial evaluation and interpolation. The complexity for the sender is bounded by $\mathrm{O}(k^2)$ for both protocols. The local complexity for other users is different. For Protocol 2, the users need to evaluate a system of at most k linear equations with at most k unknowns. Therefore, the complexity is bounded by $\mathrm{O}(k^3)$. For Protocol 3, the users need to interpolate at most $\binom{k}{t+1}$ polynomials, so their complexity is $\mathrm{O}\left(k^2 \cdot \binom{k}{t+1}\right)$.

5.3 Complexity Comparison to the Existing Solutions

As stated in Section 2.1, the existing solutions do not offer nearly the same security guarantee against malicious adversaries, but they may offer some advantage regarding message complexity. Here we compare our complexities with that of the existing work. Since many of them do not provide precise complexity analysis, the number given below are the best estimates according to our understanding of the existing protocols. We start with MCMix [1] which is mainly based on oblivious sorting (Obliv-Sort) proposed in [18]. Thus, the complexity of Obliv-Sort will lead to that of MCMix within a constant factor.

According to [18], Obliv-Sort is built on top of secure comparison [27], and the number of secure comparisons is bounded by $\mathrm{O}(n \log n)$ and $\log n$ rounds, where n is the number of users as in our case. The complexity of the secure comparison protocol is 15 rounds and $k(k-1)(279l+5)$ messages, and l denotes the number of bits for the values being compared and k is the number of servers. Therefore, the total messages in the entire system is at least $k(k-1)(279l+5)n \log n$. The local computation to perform one secure comparison is at least l, so the total computation at each server is $\Omega(ln \log n)$. These complexities are for sending one or "more" messages. However, how many more messages can be delivered each round is not clearly defined due to the reason that MCMix cannot handle the situation where multiple senders want to communicate with the same recipient. Therefore, for sending one or a constant number of messages, our protocols are clearly more efficient. If the "many-to-one" problem could be solved, MCMix would be more efficient for sending a fraction of n messages in terms of message complexity. Regardless, it has much higher round complexity: at least $30 \log n$ rounds in the communication phase compared to 1 round in our case.

Pung [2] is based on the private information retrieval (PIR) technique given in [26]. To retrieve one message, the sender need to perform at $\log n$ PIR operations. For each PIR, the sender needs to compute and produce n NxN matrices, and the total message complexity for receiving one message is bounded by $n \log n \cdot N^2$. In [26], N was chosen to be 50 but the reason behind this choice was not clear. Therefore, even if we ignore the value for N, its message complexity is much higher than ours as well as the local computations on both server and user sides. In addition, the search space is not bounded by n, but bounded by the number of mailbox labels which can be much bigger than n. Furthermore, it has the "many-to-one" problem, and users need to participate in each round.

Riffle [23] is hybrid of mix-net and PIR. Its message complexities for the servers and users are at least $3(k-1)n$ and $n^{1+1/\log_2 k}$ respectively assuming one to one communication. Its complex would be bounded by n^2 under the broadcasting mode. Thus, for sending one or a constant number of messages each round, our protocols are more efficient. Similar to other existing solutions, it cannot handle the "many-to-one" situation because it does not specify clearly how clients will learn the location to use for PIR to retrieve a particular message. Therefore, it is not realistic to claim Riffle has better message complexities than ours when considering sending a fraction of n number of messages per round. For local computations, the servers need to perform expensive zero-knowledge proofs during the shuffling phase, and their computations are much more expensive than those of our protocols. Local computation for the users is similar to ours.

As acknowledged in [1], mix-net based the solutions, such as Vuvuzela [38], offer weaker security guarantee (compared to SMC and PIR based solutions) with the total message complexity bounded by kn per round. Later works, e.g., Karaoke [24] and Loopix [30], attempt to reduce the message complexity by introducing more servers. However, this reduces the degree of connection anonymity by reducing the group size. In addition, these solutions also have the "many-to-one" problem. In general, they are more efficient than SMC and PIR based techniques and have similar complexity to our protocols when sending one or a constant number of messages each round.

5.4 Practical Considerations

The proposed protocols are information theoretically secure. However, for a large message m (e.g., multi-media files), they may not be practical when the Pedersen commitment is adopted due to the fact that the size of m is bounded by q. For practical purpose, instead of committing to m directly, the commitment can be computed based on a cryptographic hash of m. When this happens, to preserve the protocol security, we need to assume the servers are computationally bounded.

Another consideration is the use of pairwise secret keys. Most existing solutions assume the users share pairwise secret keys to encrypt their private messages. It is possible to incorporate pairwise secret keys in our proposed protocols. Nevertheless, adopting pairwise keys is not needed in our scheme as long as one server is trustworthy to avoid the complexity of key management. Furthermore, it is not efficient to achieve anonymous subgroup communication. For the proposed protocols, the only benefit of using pairwise secret keys to encrypt the messages is to guarantee message confidentiality and connection anonymity even if all servers collude. Adopting private encryption to secure the messages may also prevent the wide adoption of the proposed solutions because restrictions may exist on exporting implementation of cryptographic tools.

5.5 Extension to One-to-Many Communication

The existing solutions require round synchronizations to avoid message collisions to the same user. As a result, it very likely that a sender is not able to establish communication with a targeted message recipient. For example, in MCMix, its dial protocol cannot prevent this kind of starvation if more than two users want to send messages to the same recipient. On the other hand, our proposed protocols do not have this starvation problem because the protocols are asynchronous; that is, the communication link does not need to be established before sending or receiving messages.

The existing solutions do not support efficient one-to-many anonymous communication in the sense that messages have to be sent one person a time. In our scheme, supporting one-to-many communication is straightforward and very efficient without incurring any additional message and computation complexity if the sub-group size is bounded by $\lfloor \frac{n}{k} \rfloor$. The only required modification in our protocols is the group partition. We can treat the targeted sub-group as a single user during group permutation and partition. In other words, these users would be placed into the same partition, and during membership swapping, these users will be swapped together with the same number of users from a selected partition.

5.6 Sender Anonymity

Another criterion for anonymous communication is sender anonymity which seems to be extraneous when recipient anonymity is achieved. That is, how to prove Bob is the sender if the actual message and the recipient are not known and no one can recover the actual message except the recipient. As a result, many existing solutions focus on recipient anonymity. The only existing way to achieve sender anonymity is by introducing fake or random messages. Among the synchronized solutions, e.g., Vuvuzela [38] and MCMix [1], every user has to participate in each round by sending either fake or real messages. In our solutions, achieving sender anonymity is straightforward. Since our solutions are asynchronous, users can send messages at any moment. For example, users can periodically send some random messages.

To maximize sender anonymity, the frequency for each sender needs to be uniform. The easiest but most costly way to realize this uniformity is for all users send a message either real or random per system-defined round. Although this approach can increase message complexity to $O(n^2)$, n messages can be sent each round, and unlike the existing solutions, our protocols allow multiple users to send message to the same recipient without creating any conflicts. The quadratic total message complexity is asymptotically expensive than some of the existing solutions, but our protocols are practically better due to their simple design, stronger security guarantee, better amortized complexity and capability of many-to-one communication.

6 Conclusion

In this paper, we proposed simple and cost-effective ways to achieve anonymous communication secure against malicious adversaries. Comparing to the existing solutions, our approaches are very easy to implement and offer additional security guarantees, such as fault-tolerance, efficient detectability of message modification without complicated key management, service obliviousness and high cost of closing. The message complexities of the proposed protocols are still high although they are comparable to the existing solutions that offers weaker security guarantee. As part of our future work, to reduce the message complexities, we will explore random sampling techniques that may allow selection of a subset users to produce the needed partitions without decreasing the degree of anonymity.

References

[1] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. Mcmix: Anonymous messaging via secure multiparty computation. In *USENIX Security Symposium. USENIX Association, Vancouver, BC*, pages 1217–1234, 2017.

[2] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, Savannah, GA, 2016. USENIX Association.

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

[4] Boost. Free peer-reviewed portable c++ source libraries. `https://www.boost.org`.

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

[6] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, Jan 1988.

[7] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. Cryptology ePrint Archive, Report 2016/008, 2016. `https://eprint.iacr.org/2016/008`.

[8] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

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

[10] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, November 1998.

[11] David Cole. We kill people based on metadata.

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

[13] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 147–162, Washington, D.C., 2013. USENIX.

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

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

[16] GMP. The gnu multiple precision arithmetic library. `https://gmplib.org`.

[17] Oded Goldreich. *The Foundations of Cryptography*, volume 2, chapter General Cryptographic Protocols. Cambridge University Press, 2004.

[18] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically efficient multi-party sorting protocols from comparison sort algorithms. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology – ICISC 2012*, pages 202–216, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[19] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. 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, pages 337–348, New York, NY, USA, 2013. ACM.

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

[21] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 364–, Washington, DC, USA, 1997. IEEE Computer Society.

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

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

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

[25] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *OSDI*, pages 571–586, 2016.

[26] Carlos AGUILAR MELCHOR and Philippe GABORIT. A lattice-based computationally-efficient private information retrieval protocol, 2007. Short version presented in WEWORC, in July 2007, Bochum, Germany carlos.aguilar@unilim.fr 13844 received 27 Nov 2007, last revised 27 Nov 2007.

[27] Takashi Nishide and Kazuo Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *International Workshop on Public Key Cryptography*, pages 343–360. Springer, 2007.

[28] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[29] Michael Phillips and Matt Buchanan. How lavabit melted down. The New Yorker, October 2013. `https://www.newyorker.com/tech/elements/how-lavabit-melted-down`.

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

[31] Qt. Complete software development framework. `https://www.qt.io`.

[32] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2P mixing and unlinkable bitcoin transactions. In *NDSS*, 2017.

[33] BRUCE SCHNEIER. NSA doesn't need to spy on your calls to learn your secrets.

[34] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612 – 613, November 1979.

[35] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, Oct 1949.

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

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

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

[39] Lloyd R Welch and Elwyn R Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[40] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 179–182, 2012.

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