MHcast: Metadata-hiding Multicast with Resilience against Malicious Clients and Servers

Yulong Ming, Mingyue Wang, Cong Wang, *Fellow, IEEE*, Wei Li, *Member, IEEE*, and Xiaohua Jia, *Fellow, IEEE*

***Abstract*—End-to-end encrypted messaging systems protect data content, but metadata, such as the source and destination of encrypted data, also require protection. Existing metadata-hiding systems primarily target unicast/broadcast. Some support multicast but either expose the recipient group size, potentially revealing the sender's identity, or experience reduced throughput as the recipient group size increases, making them inefficient for multicast. Metadata-hiding multicast remains challenging, particularly when malicious clients/servers deviate from the protocol, potentially corrupting the system or deanonymizing clients.**

In this paper, we present MHcast, a high-throughput metadata-hiding multicast system. MHcast uses two distributed comparison functions with a public mapping to construct a multicast scheme. MHcast enforces a mailbox access control scheme to prevent malicious clients from disrupting multicast through malformed or junk messages. Additionally, MHcast provides an identification scheme that allows honest clients to assist an honest server in identifying the malicious server. We implement and evaluate MHcast in comparison to other metadata-hiding systems. Experiments show that MHcast's throughput is 2–10$\times$ higher than repeatedly using the unicast protocol when the group size is only 16. MHcast supports groups of any size, from 0 to N, where N is the total number of clients.

***Index Terms*—End-to-end messaging, metadata-hiding multicast, multi-party computation.**

I. Introduction

END-TO-END encryption (E2EE) techniques have been widely used in secure messaging systems like Signal [1], Telegram, and WhatsApp. It can protect data confidentiality and integrity during transmission. However, E2EE offers almost no protection for sensitive *metadata* of communication, such as the source, destination, timing, and volume of encrypted data, which can unintentionally leak information about the data content. Anonymity systems in practice like Tor [2], I2P [3], and the whistleblowing tool SecureDrop [4], aim to hide metadata. However, they are vulnerable to traffic analysis if a powerful adversary controls specific network nodes or even the entire network [5], [6]. As a result, the adversary can collect metadata to establish intelligence gathering and surveillance programs [7], [8], which deanonymize clients in the system and reveal sensitive personal data.

To protect metadata against these powerful adversaries, many metadata-hiding communication systems [9], [10] have been proposed. Among them, some systems are designed for unicast [11], [12], where a message is sent to a single client. Some other systems are for broadcast [13], [14], where a message is made public to everyone, similar to a Twitter/Instagram post. Recently, some systems offer multicast or group messaging support [15], [16], where a client sends a message to a group of clients as the *recipient group*. Metadata-hiding multicast is a useful function. For example, in the whistleblowing scenario, it enables a whistleblower to send a message to multiple journalists without revealing their identities.

Metadata-hiding multicast can be trivially achieved by repeatedly using unicast protocols. However, such a design cannot protect the recipient group size. The adversary can observe the amount of exchanged data and the number of simultaneous connections to find this meta information and use this leakage to launch volumetric attacks to deanonymize a client [17]. For example, senders can be easily identified when multicasting to groups with different sizes because the adversary can distinguish between them from the group sizes. Additionally, repeatedly using the unicast protocol incurs high communication costs.

In this work, we aim to solve the *metadata-hiding multicast* problem. There are several challenges. First, it is difficult to keep multicast senders indistinguishable, particularly with various recipient group sizes. Some prior multicast systems [18] do not hide the recipient group size as a kind of metadata. Second, it is difficult to make the communication and computational cost independent from the recipient group size. Prior multicast systems either use heavy cryptographic tools like private information retrieval [19], [15] or incur high latencies because they pass messages across many servers [16], [20]. Third, it is difficult to make the system resilient against both malicious clients and servers. Malicious clients can corrupt the system by sending malformed or junk messages, as servers cannot see the content of the encrypted messages. If a server

Digital Object Identifier 10.1109/TIFS.2026.3711782

Yulong Ming, Cong Wang, and Xiaohua Jia are with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: myl.7@my.cityu.edu.hk, {congwang, csjia}@cityu.edu.hk).

Mingyue Wang is with the Peng Cheng Laboratory, Shenzhen, China (e-mail: wangmy03@pcl.ac.cn).

Wei Li is with the College of Computer Science and Technology, Harbin Engineering University, China (e-mail: wei.li@hrbeu.edu.cn).

The work of Yulong Ming and Xiaohua Jia was supported in part by the Hong Kong Research Grants Council (RGC) under Grants RFS2425-1S01 and R1012-21. The work of Mingyue Wang was supported in part by the Peng Cheng Laboratory under Grants PCL2025A16 and PCL2024A05. The work of Cong Wang was supported in part by the RGC under Grants 11219524, 11219025, RFS2122-1S04, C1029-22G, C6015-23G, and CRS_HKUST601/24. This work was also supported in part by the InnoHK initiative, the Government of the HKSAR, and the Laboratory for AI-Powered Financial Technologies.

is malicious, it can covertly deanonymize the recipients of a message by changing its output during the protocol. To address these challenges, we propose a high-throughput metadata-hiding multicast system (MHcast) with resilience against an arbitrary number of malicious clients and at most one malicious server.

System overview. The MHcast system consists of two servers and a set of clients. Two servers hold an array of *mailboxes* secret-shared between them. To receive messages via MHcast, a client generates an authentication *write key* and a *read key*. It registers a mailbox with the servers by sharing the read and write key. In return, it gets the mailbox *address* (i.e., an integer index of the mailbox array) and becomes the mailbox owner. The mailbox owner (message recipient) distributes the mailbox address and corresponding write key to all communication peers (message senders) via some out-of-band means.

Given the mailbox addresses and write keys, any client can use MHcast to write a multicast message to a group of mailboxes whose addresses are a subset of the given addresses. Such a message is not revealed to anyone except the receiving mailbox owners. For this operation, clients use efficient cryptographic tools called distributed comparison functions (DCFs) [21]. We design the multicast scheme using two DCFs and a public mapping, enabling a client to write a message to the mailboxes of multiple recipients. Two DCFs first handle the messages of some contiguous addresses (e.g., 3, 4, 5) in a batch manner. The public mapping then maps each address to a new address, where the old contiguous addresses are mapped to the addresses of the recipient group.

Messages sent to the same group reuse the same public mapping to fulfill privacy requirements and reduce communication costs. Finally, recipients retrieve their mailboxes regularly (e.g., once every 10 seconds). Only the recipient with the read key can decrypt the mailbox to read its content.

As malicious clients/servers are able to corrupt the multicast system, we design a mailbox access control scheme, along with an identification scheme to identify the malicious server (i.e., blame). The mailbox access control scheme prevents malicious clients from writing malformed or junk content to overwrite the mailboxes. If one malicious server covertly deviates from the protocol, the identification scheme allows an honest client to assist the honest server in identifying and eliminating the malicious server.

In summary, we make the following contributions:

- We propose a metadata-hiding multicast system, MHcast. It protects recipient group sizes, which means senders sending to groups with different sizes are indistinguishable, no matter how large the group is.
- We propose a mailbox access control scheme to prevent multicast disruption caused by malicious clients and an identification scheme to identify the malicious server.
- We implement the MHcast system. We evaluate it in comparison with other metadata-hiding multicast/unicast/broadcast systems. The experiment results show that MHcast's throughput is 2–10× higher than existing systems when the recipient group size is only 16. And MHcast supports groups of any size, from 0 to N, where N is the total number of clients.

II. Related Work

In this section, we introduce related systems of metadata-hiding messaging and discuss their support for multicast.

DC-nets. MHcast belongs to a category of metadata-hiding communication systems [22], [13], [14] that use the dining cryptographer networks (DC-nets) [23], together with function secret sharing (FSS) [24], [25], [26], [21] including distribution point/comparison functions (DPFs/DCFs). The most relevant systems with MHcast in them are Express [22] and Spectrum [14]. MHcast takes the architecture of Express and the idea of mailbox access control of Spectrum for the new multicast functionality.

Express [22] is a unicast system. To hide metadata, Express writes indistinguishable message shares to all mailboxes on each server. However, to verify client requests and defend against malicious clients, Express uses the cryptographic tool, secret-shared non-interactive proofs (SNIP) [27], [28], which cannot be extended to multicast. Spectrum [14] is a broadcast system. A Spectrum recipient group retrieves the same mailbox to get the broadcast message. However, to extend to multicast, the behavior of the recipient group, which retrieves the same mailbox, should be hidden.

Dissent [29] is a system that supports group messaging. Dissent uses DC-nets without FSS and incorporates a system for accountability. However, Dissent only hides a client among its group members other than all clients, so it does not fulfill our security requirements. Moreover, Dissent is not resilient against disruption. In the event of misbehavior from a malicious client, Dissent performs an $O(M^2)$ blame protocol to find it, where M is the group size.

Mix-nets. Another category of metadata-hiding communication systems uses the mix networks (mix-net) [30]. Mix-net collects messages from clients, shuffles these messages on servers, and forwards them to destinations in a random order to hide metadata. Messages are onion-encrypted (i.e., encrypted multiple times) and shuffled on several chained server hops to protect against compromised servers and a passive network adversary. Systems using mix-net [31], [32], [33] have good horizontal scalability (i.e., can increase client capacity by adding more servers) and process in batches. However, because messages are passed through many servers, they do not suit high-throughput applications.

Rollercoaster [16] and Polysphinx [20] are multicast systems. They replicate the message payload to multiple recipients during server shuffling to achieve multicast functionality. This replication causes the communication cost to increase linearly with the recipient group size, also decreasing the throughput linearly. Rollercoaster and Polysphinx run simulations as their evaluations to mainly measure message latencies. To estimate their throughput, we use the practical data of Loopix [32] as they are based on Loopix. For groups whose sizes are thousands, the throughput has become less than 1/s.

Other cryptographic methods. Some categories of metadata-hiding communication systems use other cryptographic tools such as private information retrieval (PIR) [34],

[35], oblivious RAM (ORAM) [36], [9], or oblivious message retrieval (OMR) [37], [18]. Pung [15] and Talek [19] use PIR, and GOMR [18] is adapted from OMR. They support group messaging. However, PIR and ORAM are both heavy cryptographic tools, resulting in less throughput than the systems based on DC-nets and FSS. GOMR takes a different method to outsource the message retrieval job of each group to untrusted servers in a privacy-preserving manner. However, its total amount of work is linear to the group numbers, and its group sizes can only scale to hundreds.

III. Problem Formulation

In this section, we describe MHcast's system model, threat assumptions, and security goals to formulate the metadata-hiding multicast problem.

A. System Model

A MHcast system consists of two servers and a set of clients. The two servers collaboratively maintain an array of mailboxes. That is, each server holds an array of mailbox shares of the same length. The two mailbox shares with the same index together form a mailbox and hold two secret-shares of a message. Each mailbox is registered to and owned by a client. The client can receive messages by retrieving messages from its mailbox, or send messages by writing messages to other clients' mailboxes through the MHcast protocol.

B. Threat Assumptions

We assume an adversary who controls at most one of the two MHcast servers and any number of MHcast clients. The adversary controls the entire network to observe all communication between servers/clients, which is the same as prior work [22], [13], [14]. We further consider a conservative setting in which the adversary has prior knowledge of the target recipient group size. This setting defines the strongest metadata-hiding goal that MHcast can target, as formalized in Section III-C. We have the following threat assumptions for clients and the two servers in MHcast:

1) Any semi-honest (honest-but-curious, e.g., compromised) or malicious clients/servers can collude with each other.
2) At least one server is honest. The honest server trusts no clients, and the other server can be semi-honest or malicious.
3) Honest clients send messages only to other honest clients. The remaining clients can be malicious. At least two honest clients are required, so that they can communicate with each other and make such a MHcast instance practical. We also discuss the situation where an honest client communicates with a malicious client in Section IX for completeness.

We also assume that honest clients generate cover traffic (i.e., send dummy messages) like prior work [22], [13], [14], [38]. Such clients act indistinguishably from clients sending real messages, providing plausible deniability for the real senders. In other words, the adversary cannot link each sender to the message it sends, because the sender can "deny" sending a real message by claiming it is a dummy message. Not all honest clients are required to generate cover traffic. However, more clients that generate cover traffic allow real clients to hide in a larger set of clients.

C. Security Goals

MHcast primarily aims to hide metadata for sending, not receiving. MHcast hides whom senders send messages to, but does not hide the fact that recipients check their mailboxes. Metadata have many aspects. Some aspects can be trivially protected. For example, data volume and timing can be protected by padding and regular sending/receiving. MHcast focuses on protecting the source and destination of messages. We summarize the security goals as follows:

- **Metadata-hiding.** We aim to hide the sender and the recipient group of each message. For the sender, the adversary cannot distinguish senders. That is, though it knows each sender sends a message (dummy or not), it does not know the sender of a particular message received by a recipient. To make senders indistinguishable during multicast, MHcast is required to hide the sizes of recipient groups because this information can be used to identify senders.

  For the recipient group, the adversary does not know whether any client is in the recipient group of a particular message within a particular group set (i.e., anonymity set [39]). Any group in this set may be the recipient group and is indistinguishable from the other groups, allowing each group to "deny" receiving this message. To avoid confusion with the sender anonymity used by prior work, we call this set *metadata-hiding set*.

  MHcast does not aim to hide a recipient group among all possible subsets of clients. Particularly, for N clients in total and the recipient group size $|R|$ known by the adversary, we aim for a metadata-hiding set size $N-|R|+1$. This is reasonable: when $|R| = N$, i.e., broadcast, the adversary can trivially know the group is all clients; when $|R| = 1$, i.e., unicast, MHcast hides the only recipient among all clients, matching prior unicast systems; and when the groups are small, i.e., $|R| \ll N$, the metadata-hiding set size remains close to N and is still large.
- **Availability.** MHcast remains available if both servers follow the protocol, no matter how many clients are malicious. In other words, these malicious clients do not prevent honest clients from communicating in MHcast. Availability is not achieved if any server deviates from the protocol or halts. However, we aim to provide a scheme that, when a malicious server covertly deviates from the protocol, honest clients can assist the honest server in identifying the malicious server.

D. Limitations

MHcast does not defend against denial-of-service (DoS) attacks from many clients. However, several techniques such as CAPTCHA [40], anonymous one-time-use tokens [41], and proof-of-work [42] can be used to mitigate it. Similar

to all other metadata-hiding/anonymous systems, long-term intersection attacks on participation in the protocol can identify clients [43]. MHcast has cover traffic and makes recipients online (i.e., making recipients retrieve regularly), which can mitigate intersection attacks.

IV. Multicast Scheme

During MHcast's multicast protocol, a client writes to *all* secret-shared mailboxes like prior work [22], [13], [14]. The client writes the same secret-shared message to the recipient group's mailboxes and secret-shared zero messages to the others' mailboxes. Each mailbox aggregates the written messages with addition. From the adversary's view, all mailboxes are changed, and all clients have the opportunity to receive the message, so the recipient group of the message is hidden.

To write to all N mailboxes, the client can naively send N messages, resulting in $O(N)$ communication cost. Function secret sharing (FSS) primitives are used to trade higher computational costs for lower communication costs, which include distributed point functions (DPFs) and distributed comparison functions (DCFs). MHcast's multicast scheme uses DCFs, which have $O(\log N)$ communication cost and $O(N \log N)$ computational cost. We introduce DCFs in Section IV-A.

Existing unicast systems can hide the only recipient by writing to all mailboxes. Repeatedly using the unicast protocol to achieve multicast functionality can also hide the recipient group. However, it cannot hide the sender because the recipient group size is leaked from the repeated times. The group size can be used to identify the sender: 1) a specific group size immediately causes its sender to be identified from the clients generating cover traffic; and 2) for a group size chosen by many clients, because only senders with the same group size can hide each other, clients generating cover traffic are fragmented across several sizes, and a client can only hide from fewer clients generating cover traffic. To protect the group size and hide the sender of the message, we propose the multicast scheme in Section IV-B.

A. Preliminary: Distributed Comparison Functions

In this section, we introduce distributed comparison functions (DCFs), which are one of the function secret sharing (FSS) primitives. We then combine two DCFs to a distributed interval function (DIF).

Definition IV.1 (Comparison functions)**.** *For the input domain $[N]$ and a group $(\mathbb{G}, \oplus)$ as the output domain, a comparison function $f^{<}_{a,b}$ with $a \in [N]$ and $b \in \mathbb{G}$ is a function that for any input $x \in [N]$, the evaluated output $y \in \mathbb{G}$ has $y = b$ only when $x < a$, otherwise $y = 0$.*

DCFs. A distributed comparison function [21] is a scheme to encode a comparison function in the secret-shared form. Because MHcast uses DCFs with two parties (i.e., two servers), the comparison function is encoded as two DCF keys. The two DCF keys are delivered to the two servers respectively.

Definition IV.2 (Distributed comparison functions)**.** *For the input domain $[N]$, the output domain $(\mathbb{G}, \oplus)$, the selected values $a \in [N]$ and $b \in \mathbb{G}$, and a security parameter λ, a DCF is a scheme consisting of the methods:*

- *Key generation:* $\mathsf{DCF.Gen}(1^\lambda, a, b) \to (k_0, k_1)$.
- *Evaluation:* $\mathsf{DCF.Eval}(k_i, j) \to y_{i,j}$ *for any* $i \in \{0, 1\}$ *and any* $j \in [N]$.

That satisfies:

- *Correctness:* $y_{0,j} \oplus y_{1,j} = b$ *only when* $j < a$, *otherwise* $y_{0,j} \oplus y_{1,j} = 0$.
- *Privacy: Neither k_0 nor k_1 reveals any information about a and b. Formally speaking, there exists a probabilistic polynomial time (PPT) simulator* $\mathsf{Sim_{dcf}}$ *that can generate output computationally indistinguishable from any strict subset of the keys output by* $\mathsf{DCF.Gen}$.

Definition IV.3 (Interval functions)**.** *Similar to comparison functions, for the input domain $[N]$ and the output domain $(\mathbb{G}, \oplus)$, an interval function $f_{[a_l, a_r), b}$ with $a_l, a_r \in [N]$ and $b \in \mathbb{G}$ is a function that for any input $x \in [N]$, the evaluated output $y \in \mathbb{G}$ has $y = b$ only when $a_l \le x < a_r$, otherwise $y = 0$.*

DIFs. A distributed interval function [26] is a scheme to encode an interval function in the secret-shared form like DCFs. To construct the DIF scheme, we use Algorithms 1 and 2 proposed by [25]. The algorithms combine two DCFs for a DIF. Due to the privacy of DCFs, the two interval endpoints a_l, a_r are kept private in DIFs.

Definition IV.4 (Distributed interval functions)**.** *For the input domain $[N]$, the output domain $\mathbb{G}$, the selected values $a_l, a_r \in [N]$ and $b \in \mathbb{G}$, and a security parameter λ, a DIF is a scheme consisting of the methods:*

- *Key generation:* $\mathsf{DIF.Gen}(1^\lambda, [a_l, a_r), b) \to (k_0, k_1)$.
- *Evaluation:* $\mathsf{DIF.Eval}(k_i, j) \to y_{i,j}$ *for any* $i \in \{0, 1\}$ *and any* $j \in [N]$.

That satisfies:

- *Correctness:* $y_{0,j} \oplus y_{1,j} = b$ *only when* $a_l \le j < a_r$, *otherwise* $y_{0,j} \oplus y_{1,j} = 0$.
- *Privacy: There exists a probabilistic polynomial time (PPT) simulator* $\mathsf{Sim_{dif}}$ *that generates output computationally indistinguishable from any strict subset of the keys output by* $\mathsf{DIF.Gen}$.

Algorithm 1 $\mathsf{DIF.Gen}(1^\lambda, [a_l, a_r), b) \to (k_0, k_1)$

1: $(k_{l,0}, k_{l,1}) \leftarrow \mathsf{DCF.Gen}(1^\lambda, a_l, b)$
2: $(k_{r,0}, k_{r,1}) \leftarrow \mathsf{DCF.Gen}(1^\lambda, a_r, b)$
3: $k_0 = k_{l,0} \| k_{r,0}$, $k_1 = k_{l,1} \| k_{r,1}$

Algorithm 2 $\mathsf{DIF.Eval}(k_i, j) \to y_{i,j}$

1: $k_{l,i} \| k_{r,i} \leftarrow k_i$
2: $y_{l,i,j} = \mathsf{DCF.Eval}(k_{l,i}, j)$
3: $y_{r,i,j} = \mathsf{DCF.Eval}(k_{r,i}, j)$
4: $y_{i,j} = -y_{l,i,j} \oplus y_{r,i,j}$

B. Design Overview of Multicast Scheme

Existing systems use DPFs to write N mailboxes with lower communication costs but write the message to only one mailbox. It is also true for existing multicast/broadcast systems, which make the recipient group retrieve from the same mailbox to achieve multicast/broadcast functionality. However, MHcast directly writes the message to multiple mailboxes. We first use one distributed interval function (DIF) introduced in Section IV-A to write the same message to an interval of mailboxes. The communication and computational costs of one DIF are the same, no matter how long the interval is. That is, the interval of mailboxes is written in a batch manner. We then propose a bijective public mapping for each address, i.e., $[N] \to [N]$, and ensure that the interval is mapped to the recipient group. By combining a DIF and a public mapping, we design MHcast's multicast scheme.

Setup. A MHcast client wishing to receive messages first registers a mailbox collectively maintained by the servers. During registration, the client sends the read and write key to the servers, which we leave for Section V. The client, as a recipient, receives a mailbox address $j \in [N]$ in return. The servers maintain the address space $[N]$ for at most N clients. To allow another client as a sender to send messages to it through its mailbox, the recipient shares its mailbox address and write key with the sender. Unlike prior systems [11], [44], [31] that have a standalone signaling protocol to establish connections among clients, MHcast only requires clients to exchange their mailbox addresses and write keys out-of-band. MHcast is agnostic to how this out-of-band exchange takes place. One possibility is to make users controlling clients meet in reality and share a QR code. Another is to use another metadata-hiding system like [13] that shares similar threat assumptions and does not require signaling. Notably, a sender does not need to get all addresses and write keys of a group out-of-band. Because each group member has all addresses and write keys of the group, the sender can communicate with any existing member through MHcast to obtain them and multicast its own address and write key to the group to join it.

DIFs for secret-sharing. MHcast first uses DIFs to write the same message m to a mailbox address interval $[a_l, a_r) \subseteq [N]$. The sender randomly selects this interval but ensures that its length equals the recipient group size, i.e., $a_r - a_l = |R|$. Two DIF keys k_0, k_1 are generated with Algorithm 1 for $b = m$ and this interval. They are then sent to the two servers respectively. Each server i then uses Algorithm 2 to get $y_{i,j}$ for each mailbox j and write it to the mailbox. Written messages in the interval can be used to recover m with $y_{0,j} \oplus y_{1,j} = m$.

Public mappings. A public mapping $pm : [N] \to [N]$ is a bijection. For each mailbox address j, the server writes the DIF output at address $pm(j)$, i.e., $y_{i,pm(j)}$, to mailbox j. To generate a public mapping for $[a_l, a_r)$ and the recipient group R, the client randomly maps R to $[a_l, a_r)$ and randomly maps $[N] \setminus R$ to $[N] \setminus [a_l, a_r)$. The size of a public mapping is $O(N \log N)$, which looks large, but for $N = 2^{11}$, 2^{14}, and 2^{17}, the actual sizes are 2.75 KB, 28.0 KB, and 272 KB, which are affordable. The public mapping is sent to both servers.

Reusing public mappings. We require clients to reuse a public mapping whenever possible, rather than generating a new one each time. That is, if an existing public mapping already fits a group, the sender should reuse it. It sends the hash $H(pm)$ of the original public mapping to the servers, and the servers use it to look up the corresponding cached public mapping. This prevents additional leakage and reduces communication costs. The public mapping leaks information about the recipient group because the group must be a contiguous interval of the sequence $pm^{-1}(1), \ldots, pm^{-1}(N)$ rather than an arbitrary subset of $[N]$. That is, unlike DCFs, which hide each interval endpoint among N mailboxes, each public mapping hides the recipient group whose size is $|R|$ among $N - |R| + 1$ groups. Moreover, the leakage from multiple public mappings to the same group can accumulate through intersections of their possible group sets. Reusing public mappings makes their possible group sets the same and avoids such accumulation. A sender sending cover traffic should also randomly choose an existing public mapping, so that reusing a public mapping does not distinguish real senders from cover traffic senders.

When a server $i \in \{0, 1\}$ receives (k_i, pm) from a client, it runs Algorithm 2 and applies the public mapping to get $y_{i,pm(j)}$ for each mailbox address $j \in [N]$. It writes $y_{i,pm(j)}$ to mailbox j by addition. For a client wishing to receive a message m, it retrieves its mailbox from the servers regularly. Both servers zero out the client's mailbox after its retrieval.

Handling overwriting. Because messages are written to mailboxes by addition and not concatenating, a new message to a particular mailbox can overwrite a prior message before the recipient's regular retrievals. However, a recipient can receive multiple messages in the same mailbox when the mailbox address is given to 1) multiple senders or 2) only one sender, but it sends multiple messages. For 1), a recipient can register multiple mailboxes and share each mailbox address with only one sender. For 2), because recipients retrieve their mailboxes regularly and this retrieving is lightweight, a recipient can retrieve its mailbox more frequently (e.g., once every 10 seconds) than a sender (e.g., sending once every hour) so as not to miss a message. If a sender needs to send multiple messages within a short period, the recipient can share multiple mailbox addresses with it for each message. This way, the overwriting problem can be mitigated [22].

V. Preventing Disruption

Because MHcast clients can write to *all* mailboxes and the written messages are aggregated with addition, malicious clients can multicast nonzero messages (that are $y_{0,pm(j)} \oplus y_{1,pm(j)} \neq 0$) to some or even all mailboxes to corrupt them and prevent their owners from receiving messages, which is called *disruption*. Preventing such disruption is the major challenge faced by many prior systems [29], [22], [14]. Such disruption can be made by either a malformed client request or a valid request but with junk messages written to some particular mailboxes that the adversary targets. Meanwhile, a malicious server can also disrupt a multicast. It can *overtly* disrupt the multicast, for example, by refusing to participate in MHcast, but the system can easily find it out. However,

the malicious server can also *covertly* disrupt the multicast to exclude a particular client in order to deanonymize it, which is called an "audit" attack [22], [14]. Such attacks exist because servers reject a disruption request to defend against malicious clients, but a malicious server can exclude a client by always framing it as malicious. To prevent disruption against both malicious clients and a malicious server, by enhancing the design of Spectrum [14], we design a mailbox access control scheme with Carter-Wegman MAC and an identification scheme to identify the malicious server (i.e., blame [29], [22], [14]) with verifiable encryption.

A. Preliminary: Carter-Wegman MAC

Carter-Wegman MAC [45], [46] is a secret-shared MAC tag for a secret-shared message. It is verified to demonstrate whether an authentication key (w, γ) given in generation is valid, i.e., the same as the one given in verification.

Definition V.1 (Carter-Wegman MAC generation and verification)**.** *For a finite field $(\mathbb{F}, +, *)$ whose size is large enough for security, a sampled authentication key $(w, \gamma) \in \mathbb{F} \times \mathbb{F}$, a linear function $\mathsf{MAC}_{(w,\gamma)}(x) = w * x + \gamma$ for $x \in \mathbb{F}$, and a message $m_{\mathsf{mac}} \in \mathbb{F}$ that is additively secret-shared as $m_{\mathsf{mac}} = m_0 + m_1$, a client generates t_0, t_1 via:*

- $t = \mathsf{MAC}_{(w,\gamma)}(m_{\mathsf{mac}})$
- *Sample* t_0, $t_1 = t - t_0$

t_0, t_1 are sent to the two servers respectively. To verify t_0, t_1, the servers (knowing w and γ) do respectively:

- *Server* 0*:* $\beta_0 = w * m_0 - t_0$
- *Server* 1*:* $\beta_1 = w * m_1 - t_1$

Finally, the servers exchange β_0, β_1 and verify whether $\gamma = \beta_0 + \beta_1$. The final condition holds if and only if the tag is valid. During this process, no server gains any information about the message m_{mac} (beyond the tag's validity) because both the message and tag are secret-shared between the servers.

B. Mailbox Access Control Scheme

In this section, we propose a mailbox access control scheme in MHcast to prevent disruption against malicious clients, particularly for multicast. This scheme controls writing by enforcing that only the client with all write keys of a recipient group can write a multicast message to their corresponding mailboxes. This scheme also controls reading by enforcing that only the client with the read key can retrieve the corresponding mailbox.

For simplicity, we omit j or $pm(j)$ in the subscripts when it is not ambiguous. For example, w_j becomes w and $y_{i,pm(j)}$ becomes y_i. For sets with index set R and sums over R, e.g., $\{w_j \mid j \in R\}$ and $\sum_{j \in R}$, we write them as $\{w\}_R$ and $\sum_R$. When the index set is $[N]$, we omit the subscript and write $\{w\}$ and $\sum$.

Unicast write control. We start from Spectrum's design. Spectrum applies Carter-Wegman MAC to each written message. Because Spectrum is for unicast, the only message written to the only recipient's mailbox has $m_{\mathsf{mac}} \neq 0$, and the additively secret-shared messages written to other mailboxes have $m_{\mathsf{mac}} = 0$. So Spectrum can sum up $\mathsf{MAC}_{(w,\gamma)}(m_{\mathsf{mac}}) = w * m_{\mathsf{mac}} + \gamma$ to generate a tag t. Only the recipient's $w \cdot m_{\mathsf{mac}}$ contributes to t while ignoring all γ. By checking if t is valid, Spectrum checks if the client knows the required write key w. Spectrum then applies two optimizations to prevent client-server collusion and allow clients to send dummy messages:

- Spectrum shifts the verification procedure "to the exponent" of a new group $\mathbb{G}'$ of prime order p, where the exponent forms a field $\mathbb{F}_p$, so that a semi-honest server knowing write keys cannot leak these keys to clients. The security of this technique relies on the computational intractability of the discrete logarithm problem in $\mathbb{G}'$ [47]. Following standard notation of the discrete logarithm problem, we write the group operation as multiplication rather than $\oplus$. Clients share g^w (instead of w) with the servers during registration, where g is a generator of $\mathbb{G}'$.
- Spectrum uses the method in [48] to discard γ. In Carter-Wegman MAC, γ acts only as a "nonce" to prevent forgeries on the message 0. So Spectrum sets γ to 0 while still having the desired unforgeability property of the MAC for all nonzero messages, i.e., $\mathsf{MAC}_{(w,0)}$ remains secure for all $m_{\mathsf{mac}} \neq 0$. Particularly, clients sending dummy messages write zero messages to all mailboxes. They can forge a valid tag $t = 0$ without knowing any write keys.

Multicast write control. We enhance Spectrum's unicast design for multicast. We also apply Carter-Wegman MAC to each written message. However, multiple recipients of the recipient group have $m_{\mathsf{mac}} \neq 0$. Particularly, though these recipients receive the same message m, their m_{mac} may differ after being encoded into $\mathbb{F}_p$. To handle these m_{mac} together for multicast, we notice that by aggregating $\mathsf{MAC}_{(w,0)}(m_{\mathsf{mac}})$ with addition on the exponent to generate t, such t cannot yet be forged by malicious clients. Each $w * m_{\mathsf{mac}}$ contributing to t, i.e., corresponding to a recipient group member, still needs the corresponding w to be computed. Guessing such an aggregated t is equivalently difficult to guessing t in unicast.

Setup. While the written messages are elements of the group $\mathbb{G}$ in Section IV, we encode them as elements of the field $\mathbb{F}_p$ in this section to apply the Carter-Wegman MAC to them. We fix the group of DCFs $\mathbb{G}$ to bytes with byte-wise XOR operations. For two XOR-shared written messages y_0 and y_1, we define the encoding method $\mathsf{Enc} : \mathbb{G} \to \mathbb{F}_p$ for their encodings $\tilde{y}_0$ and $\tilde{y}_1$ as follows: both servers hash the bytes and view the hash as a large integer, while server 1 negates its integer. p must be large enough so that the integer representation fits in $\mathbb{F}_p$. That is, if $y_0 \oplus y_1 = 0$, then $y_0 = y_1$, and the encodings have $\tilde{y}_0 + \tilde{y}_1 = 0$. If $y_0 \oplus y_1 = m \neq 0$, then $\tilde{y}_0 + \tilde{y}_1 = m_{\mathsf{mac}}$, which the client can precompute.

When a client registers a mailbox, it samples a write key $w \in \mathbb{F}_p$ and a read key. That is, for the generator g of the new group $\mathbb{G}'$, the client has $g^w \in \mathbb{G}'$. It then shares the write key (on the exponent) $g^w \in \mathbb{G}'$ and the read key with the servers during mailbox registration. Though the malicious server can remember clients during registration, such registration leaks the equivalent information (the identities of the recipients) to message retrievals. The adversary cannot yet link a message to any recipient group or sender, so MHcast remains secure.

A client wishing to send messages to a recipient group needs to get all the group's write keys. We have described the method in Section IV-B. To send a message, the client first runs the multicast scheme. It then does DIF evaluation to compute m_{mac} *only* for the recipient group. It applies the Carter-Wegman MAC scheme (i.e., $\mathsf{MAC}_{(w,0)}$) to *all* m_{mac} and adds them up, i.e., $t = \sum_R w \cdot m_{\mathsf{mac}}$. This MAC generation is also described in Algorithm 3. Finally, it secret-shares the tag as $t = t_0 + t_1$ and sends t_0, t_1 to the servers respectively.

Algorithm 3 $\mathsf{MAC.Gen}(\{w\}_R, \{m_{\mathsf{mac}}\}_R) \to (t_0, t_1)$

1: $t = \sum_R w \cdot m_{\mathsf{mac}}$
2: Sample t_0, $t_1 = t - t_0$

After the server i receives t_i, it first runs the multicast scheme to get y_i for each mailbox. It then encodes each y_i as $\tilde{y}_i$ and generates β_i according to Algorithm 4, which uses *all* $\tilde{y}_i$. The servers exchange β_i to verify if $\beta_0 \cdot \beta_1 = 1$. If true, the servers write each y_i to the corresponding mailbox by addition. Otherwise, the servers reject the message.

Algorithm 4 $\mathsf{MAC.Commit}(t_i, \{g^w\}, \{\tilde{y}_i\}) \to \beta_i$

1: $\beta_i = g^{-t_i + \sum w \cdot \tilde{y}_i} = (\prod (g^w)^{\tilde{y}_i}) / g^{t_i}$

Proof. Assuming the servers follow the protocol (malicious servers are discussed in Section V-C), the MAC verification passes if and only if $\beta_0 \cdot \beta_1 = 1$. Because R as the mailbox addresses of the recipient group are also the addresses of all mailboxes written with nonzero messages (so $m_{\mathsf{mac}} \neq 0$), this condition holds as Equation 1.

$$\begin{aligned} \beta_0 \cdot \beta_1 &= \frac{g^{\sum w \cdot \tilde{y}_0}}{g^{t_0}} \cdot \frac{g^{\sum w \cdot \tilde{y}_1}}{g^{t_1}} \\ &= \frac{g^{\sum w \cdot m_{\mathsf{mac}}}}{g^t} = \frac{g^{\sum_R w \cdot m_{\mathsf{mac}}}}{g^t} = 1 \end{aligned} \tag{1}$$

Read control. A client gives its read key to the servers to retrieve its mailbox. Each server verifies if the read key is the same as the one shared during registration. A separate read key other than g^w is used for message retrievals because MHcast does not expect that a client with the write key can read messages sent by other clients.

C. Identifying the Malicious Server

Though the mailbox access control scheme prevents disruption against malicious clients, a malicious server can modify the exchanged β_i to covertly disrupt the multicast. It can modify β_i to make the mailbox access control scheme reject a message from an honest client. To defend against such attacks, Spectrum [14] gives a lightweight black-box blame protocol. We adapt it to our identification scheme, which allows an honest client to identify the malicious server (i.e., blame). An honest client with a rejected message can pay the cost of revealing the metadata of the rejected message (its content can be end-to-end encrypted, so only its metadata is revealed) to start the scheme. The scheme results in either the client or a server being identified as malicious by any honest server. If the client is identified as malicious, the rejected message is dropped. If a server is identified as malicious, any honest server should abort and immediately eliminate this server.

Verifiable encryption. We reuse Spectrum's design where the client generates a proof with verifiable encryption [49] to show its honesty. The verifiable encryption scheme can prove that a public-key-encrypted ciphertext decrypts to a particular plaintext without the private key. That is, for a key pair (pk_i, sk_i), PEnc encrypts the plaintext to the ciphertext with a public key encryption scheme, $\mathsf{PDecProof}$ decrypts the ciphertext to the plaintext and outputs a proof, and $\mathsf{PVerProof}$ verifies a proof without the private key sk_i.

Setup. Each server generates a key pair (pk_i, sk_i) and publishes the public key pk_i.

A client with a rejected message encrypts two request shares $\tau_i = (k_i, pm, t_i)$ to $C_i = \mathsf{PEnc}(pk_i, \tau_i)$ and broadcasts C_0, C_1 to both servers to start this scheme. After receiving C_0, C_1, each server i does $(\tau_i, \pi_i) = \mathsf{PDecProof}(C_i, sk_i)$ to decrypt to τ_i with a proof π_i. Now, the servers have committed to the encryption of their shares (i.e., committed to C_0, C_1). The client can go offline at this point. The servers then exchange (τ_i, π_i) and do identification: If π_i cannot be verified with $\mathsf{PVerProof}(pk_i, \tau_i, C_i, \pi_i)$, the server identifies the other server as malicious. Otherwise, the server checks if the request shares τ_0, τ_1 can pass the mailbox access control scheme. If true, the server identifies the other server as malicious. Otherwise, the server identifies the client as malicious.

Communication optimization. Both (τ_i, π_i) contain the public mapping pm, which is specific to MHcast. The two pm should be the same to pass the mailbox access control scheme. To reduce the communication cost, we hash pm during client broadcast and server share exchange. Each server i uses its own pm to get $\tau_{1-i} = (k_{1-i}, pm, t_{1-i})$ if the two hashes are equal. If not, the servers can immediately conclude that the shares cannot pass the mailbox access control scheme.

VI. Construction of MHcast Protocol

In this section, we integrate the MHcast protocol described in Sections IV and V. Figure 1 [50] shows an example of a multicast operation and the information held by clients/servers. We also simplify the notation as described in Section V-B.

Setup. The MHcast system has two servers $i \in \{0, 1\}$. Each server has an array of mailboxes. MHcast publishes $\mathbb{G}$, $\mathbb{F}_p$, $\mathbb{G}'$, a generator $g \in \mathbb{G}'$, a verifiable encryption scheme $\mathsf{PEnc/PDecProof/PVerProof}$, a hash function H, and an encoding method $\mathsf{Enc} : \mathbb{G} \to \mathbb{F}_p$. p is a prime number and large enough for Enc. For a client wishing to receive messages, it samples a write key $w \in \mathbb{F}_p$ and a read key. It then shares $g^w \in \mathbb{G}'$ and the read key with the servers to register and own a mailbox. The servers return a mailbox address (i.e., index) $j \in [N]$ to it. To allow another client to send messages to it, it shares the mailbox address j and the write key w with the client out-of-band. Moreover, each server generates a key pair (pk_i, sk_i) and publishes the public key pk_i for the identification scheme.

Multicast. For a client wishing to send a message $m \in \mathbb{G}$ to some clients $R \subseteq [N]$ as the recipient group, it needs to have all the group's write keys as $\{w\}_R$. It does the following:

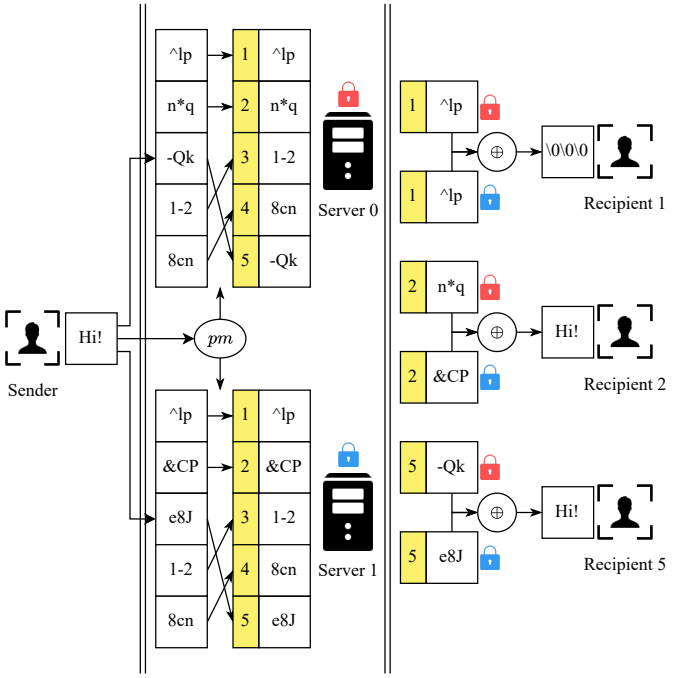


Fig. 1. An example of a multicast operation. It shows that a sender sends "Hi!" to the recipients 2 and 5. Indistinguishable messages are written to all mailboxes secret-shared between the two servers. The recipients 2 and 5 can recover "Hi!" from their mailboxes. The recipient 1 recovers a zero message.

1) It samples $a_l \in \{1, \ldots, N - |R| + 1\}$ to select a random interval $[a_l, a_r = a_l + |R|)$ on $[N]$.
2) It constructs the public mapping $pm : [N] \to [N]$ as follows. It first checks if any existing public mapping satisfies $\{pm(j) \mid j \in R\} = [a_l, a_r)$. If such a public mapping exists, the client reuses it, e.g., by providing its hash $H(pm)$. Otherwise, the client randomly maps R to $[a_l, a_r)$ and $[N] \setminus R$ to $[N] \setminus [a_l, a_r)$ to construct a new public mapping. With this public mapping, each recipient address is mapped to one point of the contiguous interval, i.e., $\{pm(j) \mid j \in R\} = [a_l, a_r)$.
3) It does DIF key generation as $(k_0, k_1) \leftarrow \mathsf{DIF.Gen}(1^\lambda, [a_l, a_r), m)$. The output k_i can be used to recover the messages written to all the mailboxes of the server i.
4) It does DIF evaluation for the recipient group as $y_i = \mathsf{DIF.Eval}(k_i, pm(j))$. It prepares $m_{\mathsf{mac},j} = \tilde{y}_0 + \tilde{y}_1$ and $\tilde{y}_i = \mathsf{Enc}(y_i)$.
5) It generates the MAC shares as $(t_0, t_1) = \mathsf{MAC.Gen}(\{w\}_R, \{m_{\mathsf{mac}}\}_R)$. The MAC shares can be used by the mailbox access control scheme to check if the client follows the protocol.
6) It sends the request $\tau_i = (k_i, pm/H(pm), t_i)$ to the server i respectively.

Writing the multicast. Each server has g^w and the read key of all mailboxes. Each server i initializes each recipient's mailbox $M_i \in \mathbb{G}$ to 0. Upon receiving τ_i from a client, each server i does the following:

1) It extracts the request as $(k_i, pm/H(pm), t_i) \leftarrow \tau_i$. If pm is given, it caches the public pm. Otherwise, it finds the cached pm by $H(pm)$.
2) It does DIF evaluation for all mailboxes as $y_i = \mathsf{DIF.Eval}(k_i, pm(j))$. Before writing all the recovered messages y_i to the corresponding mailboxes, the server needs to check the MAC.
3) It encodes y_i to $\tilde{y}_i$ and computes $\beta_i = \mathsf{MAC.Commit}(t_i, \{g^w\}, \{\tilde{y}_i\})$.
4) It computes the hash of the public mapping as $h_i = H(pm)$.
5) It exchanges (β_i, h_i) with the other server and gets β_{1-i} in return.
6) It verifies if $\beta_0 \cdot \beta_1 = 1$ and $h_0 = h_1$. If not, it rejects this client request and skips the following steps.
7) It writes to all mailboxes as $M_i \leftarrow M_i \oplus y_i$.

Retrieving the mailbox. For a client owning a mailbox and wishing to retrieve a message from the mailbox, it sends the read key to the servers. The servers return the mailbox content M_i if the read key matches and zero out M_i. The client recovers a message as $m = M_0 \oplus M_1$.

Identifying the malicious server. For a client having a rejected message and wishing to do identification, it and the servers do the following:

1) The client computes the hash of the public mapping as $h_0 = h_1 = H(pm)$.
2) The client encrypts (k_i, h_i, t_i) to $C_i = \mathsf{PEnc}(pk_i, (k_i, h_i, t_i))$ and Byzantine broadcasts C_0, C_1 to both servers. It can go offline at this point.
3) Each server i decrypts C_i to $((k_i, h_i, t_i), \pi_i) \leftarrow \mathsf{PDecProof}(sk_i, C_i)$ with a proof of correct decryption π_i.
4) Each server i publishes its share with the proof of decryption as $((k_i, h_i, t_i), \pi_i)$. Each server i can get the other τ_{1-i} by combining its own pm with the published k_{1-i} and t_{1-i}, i.e., $\tau_{1-i} = (k_{1-i}, pm, t_{1-i})$.
5) Each server does identification: If $\mathsf{PVerProof}(pk_i, \pi_i, C_i, (k_i, h_i, t_i)) = 0$ (i.e., fails to verify the proof), it aborts and eliminates the other server; Otherwise, if $h_0 = h_1$ and the requests τ_0, τ_1 can pass the mailbox access control scheme, it aborts and eliminates the other server; Otherwise, it drops this request.

Communication complexity. The complexity of each DCF key size is $O(\lambda \log N)$ [21]. λ is the bit length of the output domain of DCFs, which should not be less than $|m|$ in MHcast. A MHcast client sends two requests $(k_i, pm/H(pm), t_i)$ as the client-server communication cost per message. When pm is given, i.e., this is the first message to a particular group, its complexity is $O((\lambda + N) \log N)$, which does not depend on the recipient group size $|R|$. This complexity is higher than $O(\lambda \log N)$ of prior systems, but we will show the actual sizes in Section VIII, which are affordable. When $H(pm)$ is given, the complexity becomes $O(\lambda \log N)$. During the mailbox access control scheme, each MHcast server i sends a β_i as the inter-server communication cost per message, which is $O(1)$. During the identification scheme, the client sends C_0, C_1 and each server sends $((k_i, h_i, t_i), \pi_i)$, which are both $O(\lambda \log N)$. These are communication costs per rejected message and are only paid if a client additionally starts the

identification scheme.

Computational complexity. The complexity of each DIF key generation or evaluation operation is $O(\lambda \log N)$ [21]. Each MHcast server does DIF evaluation N times and verifies the MAC as the computational cost per message. Its complexity is $O(\lambda N \log N)$, which also does not depend on the recipient group size $|R|$. It is equivalent to prior systems, even though prior systems only support unicast/broadcast as special multicast cases. A MHcast client does DIF key generation once, does DIF evaluation $|R|$ times, and generates the public mapping and MAC shares as the computational cost per message. Its complexity is $O(\lambda |R| \log N + N)$.

VII. Security Analysis

In this section, we do security analysis for MHcast concerning the threat assumptions and security goals outlined in Section III.

Metadata-hiding. We argue that MHcast hides metadata by constructing simulators for the view of probabilistic polynomial time (PPT) adversaries corrupting any strict subset of the servers [51]. Notably, the view contains information from multiple requests from the same client, and we denote the number of messages by Q. For a public mapping pm and recipient group size $|R|$, we define the metadata-hiding set as $\mathsf{MHS}(pm, |R|) = \{\{pm^{-1}(x) \mid x \in [a, a+|R|)\} \mid a \in \{1, \ldots, N-|R|+1\}\}$.

Claim VII.1 (Sender metadata-hiding of MHcast)**.** *If at least one server is honest, then no PPT adversary $\mathcal{A}_s$ observing the entire network, corrupting any strict subset of the servers, corrupting an arbitrary subset of clients, and knowing the recipient group size $|R|$, can distinguish between two honest senders who send multiple messages and select different recipient groups.*

Claim VII.2 (Recipient group metadata-hiding of MHcast)**.** *If at least one server is honest, and an honest sender sends multiple messages with a public mapping pm, then no PPT adversary $\mathcal{A}_r$ observing the entire network, corrupting any strict subset of the servers, corrupting an arbitrary subset of clients, and knowing the recipient group size $|R|$, can distinguish among the groups in the recipient group metadata-hiding set $\mathsf{MHS}(pm, |R|)$.*

Proof. We first prove the privacy property of DIFs to use the DIF simulator $\mathsf{Sim}_{\mathsf{dif}}$ when constructing simulators for the view of adversaries.

Claim VII.3 (Privacy of DIFs)**.** *With DIFs defined by Algorithms 1 and 2, there exists a PPT simulator $\mathsf{Sim}_{\mathsf{dif}}$ that generates output computationally indistinguishable from strict subsets of the keys output by* $\mathsf{DIF.Gen}$.

Proof. We construct $\mathsf{Sim}_{\mathsf{dif}}$ by using the DCF simulator $\mathsf{Sim}_{\mathsf{dcf}}$. $\mathsf{Sim}_{\mathsf{dif}}$ takes $[N]$, $\mathbb{G}$, and a strict subset of corrupted server indices $I \subset \{0,1\}$ as input. $\mathsf{Sim}_{\mathsf{dif}}$ then uses $\mathsf{Sim}_{\mathsf{dcf}}$ as $\{k_{l,i} \mid i \in I\} \leftarrow \mathsf{Sim}_{\mathsf{dcf}}(I)$ and $\{k_{r,i} \mid i \in I\} \leftarrow \mathsf{Sim}_{\mathsf{dcf}}(I)$. It outputs the DIF keys $\{k_i \mid i \in I\}$ as $k_i = k_{l,i} \| k_{r,i}$ for $i \in I$.

Each DIF key is constructed by concatenating two DCF keys. Because DCF keys are computationally indistinguishable from real ones by the privacy of DCFs, DIF keys are also computationally indistinguishable from real ones. □

We then construct the simulator Sim for the adversaries $\mathcal{A}_s$ and $\mathcal{A}_r$ as follows:

1) Sim takes $[N]$, $\mathbb{G}$, $\mathbb{F}_p$, $(\mathbb{G}', g)$, $\{g^w\}$, and a strict subset of corrupted server indices $I \subset \{0,1\}$ as input. For $\mathcal{A}_r$, it additionally takes pm as input.
2) $\{k_i \mid i \in I\} \leftarrow \mathsf{Sim}_{\mathsf{dif}}([N], \mathbb{G}, I)$ for each message.
3) For $\mathcal{A}_s$, it samples $pm : [N] \to [N]$ by randomly mapping $[N]$ to $[N]$. For $\mathcal{A}_r$, it uses the same pm.
4) It samples $t_i \in \mathbb{F}_p$ such that $t_0 + t_1 = 0$ for each message.
5) $\beta_i = g^{t_i}$ for $i \in \{0,1\} \setminus I$.
6) It outputs the view $((\{(k_i, t_i) \mid i \in I\}, \{\beta_i \mid i \in \{0,1\} \setminus I\})_0, \ldots, (\ldots)_Q, pm)$.

The view includes 1) each DIF key k_i for the corrupted server i, 2) each MAC tag share t_i for the corrupted server i, 3) each β_i generated by $\mathsf{MAC.Commit}$ for the honest server i, and 4) the public mapping pm.

For such a simulator, the DIF keys are computationally indistinguishable from real ones by the privacy of DIFs. The MAC tag shares and β_i are computationally indistinguishable from real ones, because while y_i is the output of DIFs, $\tilde{y}_i$ is its field-encoded value, t_i is a secret share of 0, and $\beta_i = (g^{\sum w \cdot \tilde{y}_i})/g^{t_i}$ is a random multiplicative secret share of 1, which matches how the simulator samples β_i (by sampling t_i). We discuss the computational indistinguishability of the public mapping pm as follows.

The public mapping pm is equivalent to an array v_{pm} with $v_{pm}[j] = pm^{-1}(j)$ for $j \in [N]$, and the sender selects an interval $[a_l, a_r)$ on it with $\{v_{pm}[j] \mid j \in [a_l, a_r)\} = R$. As discussed in Section IV-B, pm leaks only that R is a contiguous interval in v_{pm}. For a group size $|R|$, there are $N - |R| + 1$ such intervals, which are exactly the groups in $\mathsf{MHS}(pm, |R|)$.

For $\mathcal{A}_s$, each sender builds its public mapping by placing a random permutation of R in the selected interval $[a_l, a_r)$ and a random permutation of $[N] \setminus R$ outside it. Because $\mathcal{A}_s$ has no information about R beyond $|R|$, the resulting v_{pm} is indistinguishable from a random permutation of $[N]$. Therefore, the view depends on neither the recipient group nor its size, so $\mathcal{A}_s$ cannot distinguish two senders that select different recipient groups.

For $\mathcal{A}_r$, the public mapping pm is fixed and taken as the simulator's input. The simulator reconstructs the rest of the view without the selected interval. Therefore, its output distribution is the same for every group in $\mathsf{MHS}(pm, |R|)$, making the groups computationally indistinguishable. Moreover, MHcast requires a sender to reuse the same pm for the same group, so the Q messages share a single public mapping. Thus, all candidate groups remain within the same metadata-hiding set, revealing no information about R beyond its membership in $\mathsf{MHS}(pm, |R|)$. □

Availability. We prove that MHcast has availability by proving that: 1) MHcast has disruption resistance. In other words, a client cannot disrupt a mailbox at j without knowing

its write key w_j; and 2) the identification scheme of MHcast never results in an honest client/server being identified as malicious.

Claim VII.4 (Disruption resistance of MHcast)**.** *Assuming the hardness of the discrete logarithm problem [47] in the group $\mathbb{G}'$, no probabilistic polynomial time (PPT) client can write to mailboxes of a recipient group $R \subseteq [N]$ and pass the mailbox access control scheme performed by the servers without knowledge of all write keys $\{w_j \mid j \in R\}$ of the recipient group.*

Proof. Assuming towards contradiction that an adversarial client can generate a request with non-negligible probability. Such a client request is potentially ill-formed. It can pass the mailbox access control scheme to write nonzero messages to the mailboxes of a particular recipient group $R \subseteq [N]$ while missing only one write key w_{j^*} that $j^* \in R$. It can get w_j for $j \in R \setminus \{j^*\}$. Particularly, there can be only one such request, which only exists for a particular recipient group R instead of any recipient group, and for particular $\{w_j \mid j \in R \setminus \{j^*\}\}$. We can use the client to extract the discrete logarithm for any $\mathbb{F}_p$ exponent as follows. Given g^{w^*} for $w^* \in \mathbb{F}_p$, we give the client $\{g^{w_j} \mid j \in R \setminus \{j^*\}\}$ (without the write key at j^*) and get in return the client request (τ_0, τ_1) generated according to the MHcast protocol described in Section VI, which includes t. Given the request, we can compute $m_{\mathsf{mac},j}$ for $j \in [N]$ by evaluating the DIF, applying Enc and the public mapping. Because the request passes the mailbox access control scheme, we have $\beta_0 \cdot \beta_1 = 1$, so $t = w^* \cdot m_{\mathsf{mac},j^*} + \sum_{R \setminus \{j^*\}} w_j \cdot m_{\mathsf{mac},j}$. With knowledge of $m_{\mathsf{mac},j}$ for $j \in [N]$ and w_j for $j \in [N] \setminus \{j^*\}$, we can solve for w^*. We conclude that the client has knowledge of w^*, which is a contradiction. □

For the identification scheme, we prove that incorrect identification indicates a failure of the verifiable encryption scheme or the mailbox access control scheme. Because this identification scheme is adapted from the blame protocol of Spectrum and the proof is the same, we only sketch the proof and refer to Spectrum [14] for the full proof.

Claim VII.5. *The identification scheme of MHcast is sound and private:*

- *Soundness: The identification scheme of MHcast is sound if no honest client/server would be identified as malicious: (1, honest client) For all honest C_i, no probabilistic polynomial-time (PPT) adversary can create a request share τ_i and proof of decryption π_i such that the identification scheme results in identifying the client as malicious when running with (τ_i, π_i, C_i). (2, honest server) For all honest request share τ_i and proof of decryption π_i, no PPT adversary can create C_i such that the identification scheme results in identifying a server as malicious when running with (τ_i, π_i, C_i).*
- *Privacy: The identification scheme of MHcast is private if for both randomly sampled key pairs (pk_i, sk_i) and any strict subset $I \subset \{0, 1\}$, the distributions of $(\{(pk_i, sk_i) \mid i \in I\}, \{C_i \mid i \in \{0, 1\}\})$ are computationally indistinguishable. Informally speaking, both C_i reveal no information about the request.*

Proof (sketch). We sketch the proof for the soundness and privacy of the identification scheme:

- Soundness: 1) for an honest client, the result identifying it as malicious means the decryption proof verification succeeds, and the request does not pass the mailbox access control scheme. However, the successful decryption proof verification means the request is created by this honest client and should pass the mailbox access control scheme, which is a contradiction; and 2) for an honest server, the result identifying it as malicious means either the proof of decryption fails or the request passes the mailbox access control scheme. However, honest proof of decryption generated by the honest server does not fail, and the request is previously rejected when the malicious client sends the message. It is a contradiction.
- For privacy, C_i as encryption output of PEnc does not reveal any information about the encrypted τ_i.

□

VIII. Evaluation

In this section, we build and evaluate MHcast. We compare it with state-of-the-art metadata-hiding communication systems: Express [22], Spectrum [14], Talek [19], and GOMR [18]. Particularly, Talek and GOMR support multicast. We build MHcast in ˜600 lines of Rust code, which is open-source at https://github.com/myl7/mhcast. We build the library for DCFs in ˜2000 lines of Rust code, which is open-source at https://github.com/myl7/fss. We use Matyas-Meyer-Oseas one-way compression functions with AES128 as cryptographically secure PRGs of DCFs [52] for speed. We use the library Rustcrypto/aes for AES128. We use the Jubjub curve points as $\mathbb{G}'$ and its scalar field on the exponent as $\mathbb{F}_p$. We use the library zkcrypto/jubjub for the Jubjub curve.

We do the evaluation on two AWS `c5a.4xlarge` servers in the regions `ap-northeast-1` (Tokyo) and `us-west-1` (California) respectively. Each server has 8 cores and 16 threads with hyper-threading. Its memory is 32GiB and sufficient for the evaluation. Its operating system is Ubuntu 24.04.2. The client runs on a laptop and connects to the servers to use MHcast. The round-trip time (RTT) between the two servers is 108ms. The RTT between the client and servers is 55.3ms and 159ms. The bandwidth between the two servers is 278Mbps and large enough to ignore the transmission time of exchanged data. Each server runs 16 threads, and each thread runs an instance of the evaluated systems. The message size is fixed to 1KiB. The last bit of messages is always unset due to the DCF algorithm [21], so only 1023 bits carry information. The DCF algorithm requires the security parameter λ to match the message size, resulting in $\lambda = 1023$. We do not instantiate cover traffic as a separate workload because honest clients generate cover traffic in MHcast by running the same protocol as real senders. Therefore, in an evaluation with one real sender and $M-1$ cover traffic clients, the aggregate computation and communication costs are M times the values measured below. The same cost accounting

applies to the compared systems, so the comparison remains fair.

A. Throughput

Figure 2 compares MHcast with Spectrum, Express, and Talek. For the unicast system Express, we repeatedly use its unicast protocol for multicast. For the broadcast system Spectrum, because its broadcast protocol only writes to a single mailbox (i.e., broadcast channel), we repeatedly use its broadcast protocol to different mailboxes for multicast. As a result, Spectrum and Express's throughput decreases linearly with the recipient group size $|R|$. Meanwhile, MHcast's throughput does not decrease while $|R|$ increases. For Spectrum, MHcast's throughput is equivalent when $|R| = 1$ and 15× higher when $|R| = 16$. For Express, MHcast's throughput is its 1/7–2/3 when $|R| = 1$ and the number of mailboxes $N = 2^{14}$ or 2^{17}. This is because Express has a weaker threat model that assumes no client-server collusion [22], [14]. With this threat model, Express can use a lightweight cryptographic tool that is particularly for unicast, secret-shared non-interactive proofs (SNIPs), leading to this result. However, MHcast's throughput still becomes higher when $|R| \geq 4$ and 2–10× higher when $|R| = 16$.

For Talek, MHcast's throughput is 10–1000× higher. To match our security goals, we evaluate Talek by making all recipients receive a message after senders send each message. Talek's original evaluation also does this but with a different number of recipients. Unlike MHcast, which does work for each sender and primarily hides senders, Talek makes a different design decision to do work for each recipient, leading to this result. Talek is more suitable for scenarios where sending is more frequent than receiving, and recipients require primary protection.

We also evaluate MHcast with large recipient groups whose sizes are up to the total client number. The right segment of the broken x-axis in Figure 2 shows the results. We evaluate this to show that MHcast's throughput does not decrease when the group sizes increase, no matter how large the group is.

Figure 3 compares MHcast with GOMR, which has two variants, AGOMR and FGOMR. GOMR makes a different design decision to do work for each group, no matter whether the group is the recipient group. As a result, GOMR's throughput decreases linearly with the group number. We also observe that GOMR's throughput does not decrease when the group size increases, matching the results of GOMR's original evaluation [18], so we only change $|R|$ from 1 to 16. For $N = 2^{14}$ and $N = 2^{17}$, MHcast's throughput is 1/13–1/2 of GOMR's throughput when GOMR is evaluated with only one group. Because GOMR's effective throughput is divided by the number of groups, MHcast's throughput would be higher once the system has more than 20 groups.

We further evaluate equivalent parts of MHcast, Express, and Spectrum. We do this evaluation to reason their throughput differences and highlight MHcast's high-throughput implementation. These systems can be divided into two parts: multicast and access control (corresponding to DPFs and audit of Express and Spectrum). We measure the time of these parts for a single thread. Figure 4 shows the results.

Multicast. For Express, MHcast's multicast is 7–17× faster even when $|R| = 1$. During multicast, Express does one DPF evaluation for each recipient, and MHcast does two DCF evaluations for the whole recipient group. One DPF evaluation is theoretically faster than one DCF evaluation [25], [21]. However, due to our fast DCF implementation, MHcast's multicast is still faster when $|R| = 1$. For Spectrum, MHcast's multicast is faster when $|R| \geq 4$. The original Spectrum implementation does not follow its protocol to use DPFs but sends all N written messages to the servers. This implementation decision considerably reduces its computational cost, leading to the results. However, it also increases its communication cost from $O(\lambda \log N)$ to $O(\lambda N)$. Spectrum assumes the mailbox number N is small because Spectrum clients can share the same mailbox as a broadcast channel. Such a trade-off is suitable for Spectrum's scenario (i.e., broadcast) but not for multicast. MHcast can still be faster with Spectrum's advantage, highlighting our optimized implementation.

Access control. For Express, MHcast's access control is slower when $|R| \leq 8$ and then faster. We have explained it in the throughput evaluation. For Spectrum, MHcast's access control is equivalent when $|R| = 1$ as expected because the computation of their access control schemes is similar. When $|R| = 32$, because Spectrum is required to do access control for each recipient, MHcast's access control becomes 15× faster.

DCFs&DPFs. Finally, to highlight our high-performance DCF implementation that contributes to MHcast's high throughput, Figure 5 shows the time to do DPF/DCF evaluation for all mailboxes, i.e., DPF/DCF full domain evaluation [25]. Even though DCF evaluation is theoretically slower than DPF evaluation [25], [21], MHcast's DCF evaluation is 8× faster than Express's DPF evaluation. This is because Express does DPF evaluation for discrete addresses (e.g., $1, 100, 201$), but MHcast does DCF evaluation for contiguous addresses (e.g., $1, 2, \ldots, N$). Contiguous addresses can share some computations during DPF/DCF evaluation, contributing to this result.

B. Communication Costs

Table I shows MHcast's communication costs. The client-server communication costs in the "with client-sent pm" row are large but affordable because even the largest value of the table is close to the size of a web image. While senders reuse public mappings as mentioned in Section IV-B, the communication costs in the "with server-cached pm" row considerably decrease. The inter-server communication cost during access control is constant and small.

IX. Discussion

Honest-malicious client communication. We assume honest clients send messages only to other honest clients in MHcast. This matches our primary deployment scenario, whistleblowing, as whistleblowers only send messages to trusted journalists. In this section, we discuss the situation where an honest client communicates with a malicious client for completeness. When the malicious client is the sender, the

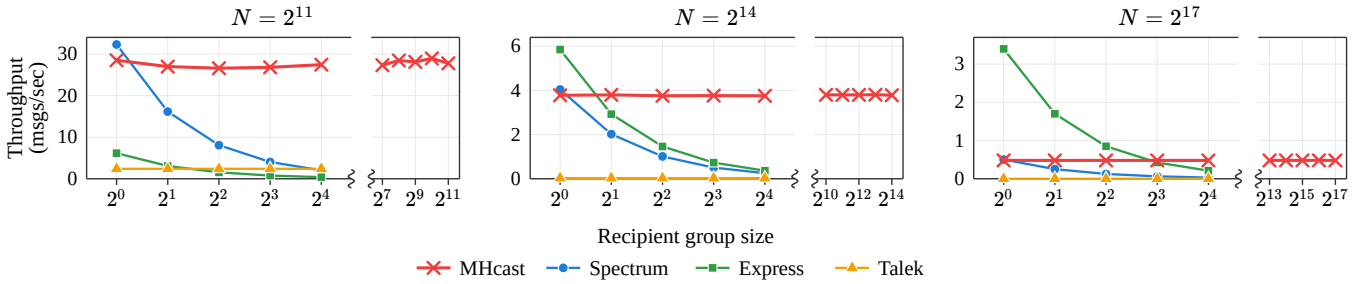


Fig. 2. Throughput (higher is better) of MHcast, Spectrum, Express, and Talek with varying recipient group size $|R|$ and mailbox number N. The right segment of the broken x-axis shows MHcast's throughput for large $|R|$, up to $|R| = N$.

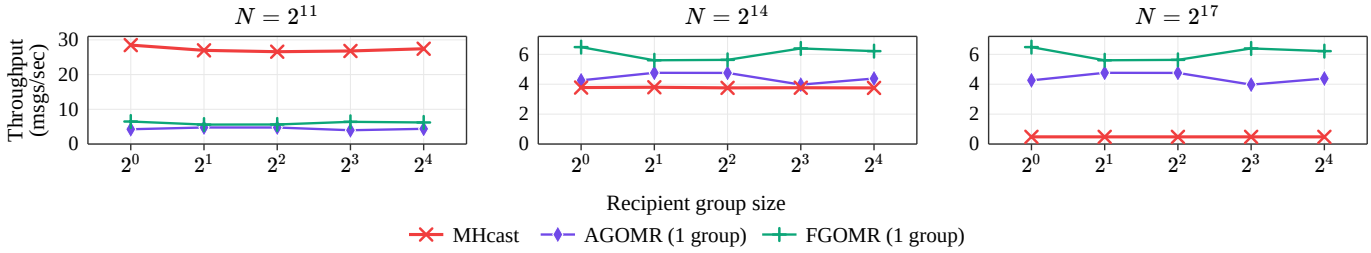


Fig. 3. Throughput (higher is better) of MHcast and GOMR's two variants, AGOMR and FGOMR. GOMR's variants are evaluated with only one group, so their effective throughput should be divided by the number of groups.

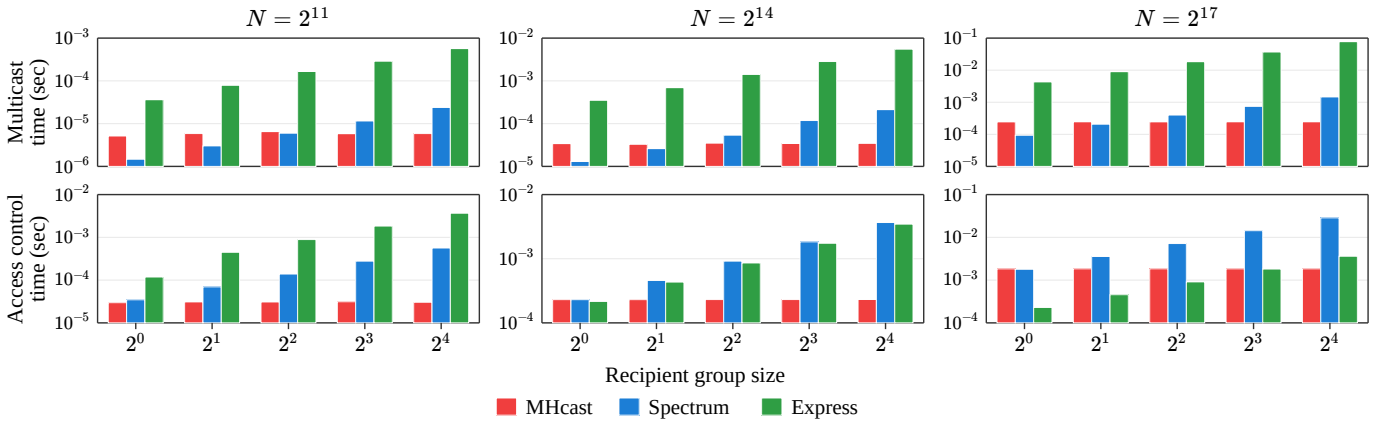


Fig. 4. Multicast (top row) and access control (bottom row) time (lower is better) of MHcast, Spectrum and Express, which explain the throughput differences in Figure 2.

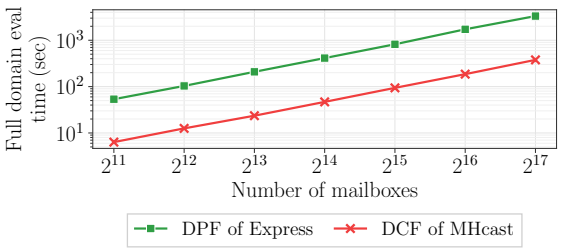


Fig. 5. DCF/DPF full domain evaluation time (lower is better) of MHcast and Express, showing MHcast's high-performance DCF implementation.

TABLE I
MHCAST COMMUNICATION COSTS

	# of mailboxes N		
	2^{11}	2^{14}	2^{17}
Client-server (per msg)			
with client-sent pm	102KB	179KB	703KB
with server-cached pm	48.4KB	60.8KB	73.1KB
Inter-server (per msg)		64B	

honest recipient should not give its write key to the sender. As a result, this matches the disruption case discussed in Section V. MHcast prevents this disruption, i.e., the servers reject the sender's message and frame it as malicious. When the malicious client is the recipient, the adversary can identify the honest sender when it obtains the write key from the malicious recipient out-of-band. If the honest client obtains the address and write key without revealing its identity to the adversary (e.g., from a public board), the malicious recipient can still

retrieve its mailbox regularly. Once it observes a message, it can narrow the real sender to the clients who sent requests between its previous retrieval and the current retrieval. The malicious recipient may also obtain the sender's information from the message content. Overall, neither case matches the whistleblowing scenario and both require the honest client to actively expose itself.

Security for dynamic groups. MHcast primarily supports fixed groups due to the metadata-hiding security attribute for the sender. This matches the whistleblowing scenario, as the journalists trusted by a whistleblower do not change often. We discuss the method to exchange the addresses and write keys when a group changes in Section IV-B. However, while MHcast uses public mappings to arrange groups to intervals, group changes (including group creation) cause public mappings to change, which can be observed by the adversary. Cover traffic also produces group changes, but the group change pattern can also be used to identify real senders and is beyond MHcast's scope. One trivial solution that trades efficiency for security is to reuse the same public mapping but send additional requests to the new group members.

X. Conclusions

We have presented MHcast, a high-throughput metadata-hiding multicast system with resilience against malicious clients and servers. MHcast uses DCFs and public mappings to build a metadata-hiding multicast scheme. To defend against malicious clients during multicast, MHcast uses Carter-Wegman MAC to enforce a mailbox access control scheme. Additionally, MHcast provides an identification scheme that allows honest clients to assist an honest server in identifying the malicious server. Our experiments show that compared to existing systems, MHcast's throughput is high and does not decrease when the sizes of recipient groups increase.

References

[1] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A Formal Security Analysis of the Signal Messaging Protocol," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, Apr. 2017, pp. 451–466. [Online]. Available: https://ieeexplore.ieee.org/document/7961996

[2] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USA: USENIX Association, Aug. 2004, p. 21.

[3] "I2P Anonymous Network," https://geti2p.net. [Online]. Available: https://geti2p.net

[4] "SecureDrop: Share and accept documents securely," https://securedrop.org. [Online]. Available: https://securedrop.org

[5] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson, "Users get routed: traffic correlation on tor by realistic adversaries," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 337–348. [Online]. Available: https://doi.org/10.1145/2508859.2516651

[6] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: passive deanonymization of tor hidden services," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. USA: USENIX Association, Aug. 2015, pp. 287–302.

[7] "Snowden Speaks: A Vanity Fair Special Report | Vanity Fair," Apr. 2014, https://www.vanityfair.com/news/politics/2014/05/edward-snowden-politics-interview. [Online]. Available: https://www.vanityfair.com/news/politics/2014/05/edward-snowden-politics-interview

[8] "Planned NSA reforms still leave journalists reason to worry - Columbia Journalism Review," Apr. 2014, https://www.cjr.org/behind_the_news/two_hops_reporters.php. [Online]. Available: https://www.cjr.org/behind_the_news/two_hops_reporters.php

[9] W. Chen, T. Hoang, J. Guajardo, and A. A. Yavuz, "Titanium: A Metadata-Hiding File-Sharing System with Malicious Security," in *Network and Distributed System Security (NDSS) Symposium*, Feb. 2022. [Online]. Available: https://par.nsf.gov/biblio/10388342-titanium-metadata-hiding-file-sharing-system-malicious-security

[10] R. Lian, Y. Ming, C. Cai, Y. Zheng, C. Wang, and X. Jia, "Nemesis: Combating abusive information in encrypted messaging with private reporting," in *29th European Symposium on Research in Computer Security*, Bydgoszcz, Poland, Sep. 2024.

[11] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: scalable private messaging resistant to traffic analysis," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 137–152. [Online]. Available: https://dl.acm.org/doi/10.1145/2815400.2815417

[12] P. Jiang, Q. Wang, J. Cheng, C. Wang, L. Xu, X. Wang, Y. Wu, X. Li, and K. Ren, "Boomerang: Metadata-Private Messaging under Hardware Trust," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 877–899. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/jiang

[13] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An Anonymous Messaging System Handling Millions of Users," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 321–338, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7163034

[14] Z. Newman, S. Servan-Schreiber, and S. Devadas, "Spectrum: High-bandwidth Anonymous Broadcast," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 229–248. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/newman

[15] S. Angel and S. Setty, "Unobservable Communication over Fully Untrusted Infrastructure," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 551–569. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[16] D. Hugenroth, M. Kleppmann, and A. R. Beresford, "Rollercoaster: An Efficient Group-Multicast Scheme for Mix Networks," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3433–3450. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/hugenroth

[17] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill, "Generic Attacks on Secure Outsourced Databases," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1329–1340. [Online]. Available: https://doi.org/10.1145/2976749.2978386

[18] Z. Liu, E. Tromer, and Y. Wang, "Group Oblivious Message Retrieval," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Feb. 2024, pp. 118–118, iSSN: 2375-1207. [Online]. Available: https://www.computer.org/csdl/proceedings-article/sp/2024/313000a115/1Ub23ocBmKI

[19] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. Anderson, A. Krishnamurthy, and B. Parno, "Talek: Private Group Messaging with Hidden Access Patterns," in *Proceedings of the 36th Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 84–99. [Online]. Available: https://dl.acm.org/doi/10.1145/3427228.3427231

[20] D. Schadt, C. Coijanovic, C. Weis, and T. Strufe, "PolySphinx: Extending the Sphinx Mix Format With Better Multicast Support," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Oct. 2023, pp. 48–48, iSSN: 2375-1207. [Online]. Available: https://www.computer.org/csdl/proceedings-article/sp/2024/313000a044/1RjEaiu0Ehy

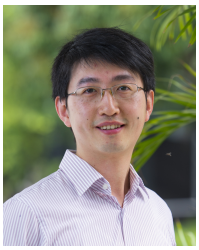
[21] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation," in *Advances in Cryptology – EUROCRYPT 2021*, A. Canteaut and F.-X. Standaert, Eds. Cham: Springer International Publishing, 2021, pp. 871–900.

[22] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1775–1792. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/eskandarian

[23] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, Jan. 1988. [Online]. Available: https://doi.org/10.1007/BF00206326
[24] E. Boyle, N. Gilboa, and Y. Ishai, "Function Secret Sharing," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer, 2015, pp. 337–367.
[25] ——, "Function Secret Sharing: Improvements and Extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1292–1303. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978429
[26] ——, "Secure Computation with Preprocessing via Function Secret Sharing," in *Theory of Cryptography*, D. Hofheinz and A. Rosen, Eds. Cham: Springer International Publishing, 2019, pp. 341–371.
[27] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 259–282. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs
[28] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Zero-knowledge proofs on secret-shared data via fully linear pcps," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 67–97.
[29] H. Corrigan-Gibbs and B. Ford, "Dissent: accountable anonymous group messaging," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, Oct. 2010, pp. 340–350. [Online]. Available: https://doi.org/10.1145/1866307.1866346
[30] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981. [Online]. Available: https://doi.org/10.1145/358549.358563
[31] D. Lazar, Y. Gilad, and N. Zeldovich, "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 711–725. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/lazar
[32] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix Anonymity System," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1199–1216. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska
[33] A. Kwon, D. Lu, and S. Devadas, "XRD: Scalable Messaging System with Cryptographic Privacy," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 759–776. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/kwon
[34] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *J. ACM*, vol. 45, no. 6, pp. 965–981, Nov. 1998. [Online]. Available: https://doi.org/10.1145/293347.293350
[35] I. Ahmad, Y. Yang, D. Agrawal, A. E. Abbadi, and T. Gupta, "Addra: Metadata-private voice communication over fully untrusted infrastructure," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/ahmad
[36] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, Jan. 1987, pp. 182–194. [Online]. Available: https://doi.org/10.1145/28395.28416
[37] Z. Liu and E. Tromer, "Oblivious Message Retrieval," in *Advances in Cryptology – CRYPTO 2022*, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 753–783.
[38] H. Corrigan-Gibbs and B. Ford, "Conscript your friends into larger anonymity sets with JavaScript," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, ser. WPES '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 243–248. [Online]. Available: https://dl.acm.org/doi/10.1145/2517840.2517866
[39] A. Pfitzmann and M. Köhntopp, "Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology," in *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*, H. Federrath, Ed. Berlin, Heidelberg: Springer, 2001, pp. 1–9. [Online]. Available: https://doi.org/10.1007/3-540-44702-4_1
[40] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Berlin, Heidelberg: Springer, 2003, pp. 294–311.
[41] B. Kreuter, T. Lepoint, M. Orrù, and M. Raykova, "Anonymous Tokens with Private Metadata Bit," in *Advances in Cryptology – CRYPTO 2020*, D. Micciancio and T. Ristenpart, Eds. Cham: Springer International Publishing, 2020, pp. 308–336.
[42] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *Advances in Cryptology — CRYPTO' 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer, 1993, pp. 139–147.
[43] O. Berthold and H. Langos, "Dummy Traffic against Long Term Intersection Attacks," in *Privacy Enhancing Technologies*, R. Dingledine and P. Syverson, Eds. Berlin, Heidelberg: Springer, 2003, pp. 110–128.
[44] D. Lazar and N. Zeldovich, "Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 571–586. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/lazar
[45] J. L. Carter and M. N. Wegman, "Universal classes of hash functions (Extended Abstract)," in *Proceedings of the ninth annual ACM symposium on Theory of computing*, ser. STOC '77. New York, NY, USA: Association for Computing Machinery, May 1977, pp. 106–112. [Online]. Available: https://dl.acm.org/doi/10.1145/800105.803400
[46] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, no. 3, pp. 265–279, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022000081900337
[47] Y. Tsiounis and M. Yung, "On the security of ElGamal based encryption," in *Public Key Cryptography*, H. Imai and Y. Zheng, Eds. Berlin, Heidelberg: Springer, 1998, pp. 117–134.
[48] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer, 2012, pp. 643–662.
[49] J. Camenisch and V. Shoup, "Practical Verifiable Encryption and Decryption of Discrete Logarithms," in *Advances in Cryptology - CRYPTO 2003*, D. Boneh, Ed. Berlin, Heidelberg: Springer, 2003, pp. 126–144.
[50] "Icons by icons8," https://icons8.com. [Online]. Available: https://icons8.com
[51] R. Canetti, "Universally composable security," *J. ACM*, vol. 67, no. 5, Sep. 2020. [Online]. Available: https://doi.org/10.1145/3402457
[52] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical Private Queries on Public Data," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 299–313. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wang-frank

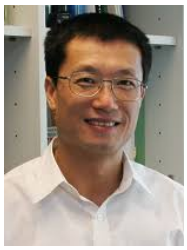
Yulong Ming received the B.E. degree from University of Science and Technology of China, Hefei, China, in 2022. He is working toward the Ph.D. degree in the Department of Computer Science at City University of Hong Kong. His current research interests include network security, applied cryptography, and artificial intelligence security.

Mingyue Wang received the B.E. degree from Harbin Engineering University, Harbin, China, in 2018. She received the Ph.D. degrees in the Department of Computer Science from both Harbin Institute of Technology, Shenzhen, and City University of Hong Kong, in 2025. She is now an Assistant Research Fellow in Peng Cheng Laboratory, Shenzhen, China. Her current research interests include network security, applied cryptography, and trustworthy artificial intelligence.



Cong Wang (F'21) is currently the Chair Professor in the Department of Computer Science, City University of Hong Kong. His research interests include data security and privacy, AI security, software security and systems, and blockchain and decentralized applications. He is a co-recipient of the Best Paper Award of IEEE ICDCS 2020, ICPADS 2018, MSN 2015, the Best Student Paper Award of IEEE ICDCS 2017, and the IEEE INFOCOM Test of Time Paper Award 2020. He has served as associate editor for IEEE TSC, IEEE IoT-J, IEEE Networking Letters, and TPC co-chairs for a number of IEEE conferences and workshops. He has also been the Editor-in-Chief of IEEE TDSC. He is a fellow of the IEEE, and member of the ACM.

Wei Li received the B.E. and M.Sc. degrees in Computer Science and Technology from Wuhan University and Harbin Institute of Technology, in 2012 and 2015, respectively. He received the Ph.D. degree from the School of Computer Science and Engineering, the University of New South Wales, in 2019. He is currently a Professor in the College of Computer Science and Technology, Harbin Engineering University, China. His main research interests include Spatio-temporal Data Mining, Networking and Computer Vision. He is also an IEEE and ACM Member.



Xiaohua Jia (F'13) received the BSc and MEng degrees in 1984 and 1987, respectively, from the University of Science and Technology of China, and the DSc degree in 1991 in information science from the University of Tokyo. He is currently the Chair Professor with the Department of Computer Science at City University of Hong Kong. His research interests include data privacy and security, cloud computing and distributed systems, computer networks, wireless sensor networks and mobile wireless networks. He is an editor of the IEEE TPDS (2006-2009), Wireless Networks, Journal of World Wide Web, Journal of Combinatorial Optimization, etc. He is the general chair of ACM MobiHoc 2008, TPC co-chair of IEEE MASS 2009, area-chair of IEEE INFOCOM 2010, TPC co-chair of IEEE GlobeCom 2010-Ad Hoc and Sensor Networking Symposium, and Panel co-chair of IEEE INFOCOM 2011. He is a fellow of the IEEE Computer Society.