Ring of Gyges: Accountable Anonymous Broadcast via Secret-Shared Shuffle

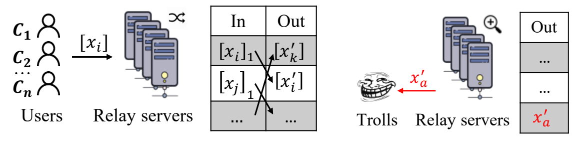
Wentao Dong[1], Peipei Jiang[1,2], Huayi Duan[3], Cong Wang[1,✉], Lingchen Zhao[2], and Qian Wang[2]
[1]City University of Hong Kong, [2]Wuhan University, [3]ETH Zurich
[1]{wentao.dong, pp.jiang}@my.cityu.edu.hk, [3]huayi.duan@inf.ethz.ch,
[1]congwang@cityu.edu.hk, [2]{lczhaocs, qianwang}@whu.edu.cn

***Abstract*—Anonymous broadcast systems, which allow users to post messages on a public bulletin board without revealing their identities, have been of persistent interest over the years. Recent designs utilizing multi-party computation (MPC) techniques have shown competitive computational efficiency (CCS' 20, NDSS' 22, PETS' 23). However, these systems still fall short in communication overhead, which also dominates the overall performance. Besides, they fail to adequately address threats from misbehaving users, such as repeatedly spamming the system with inappropriate, illegal content. These tangible issues usually undermine the practical adoption of anonymous systems.**

This work introduces *Gyges*, an MPC-based anonymous broadcast system that minimizes its inter-server communication while reconciling critical anonymity and accountability guarantees. At the crux of *Gyges* lies an honest-majority four-party *secret-shared relay*. These relay parties jointly execute two key protocols: 1) a "silent shuffling" protocol that requires no online communication but relies solely on non-interactive, local computations to unlink users from their messages, thereby ensuring sender anonymity; 2) a companion fast and lean tracing protocol capable of relinking a specific shuffled message back to its originator when the content severely violates moderation policy, without jeopardizing others' anonymity guarantees. Additionally, *Gyges* adheres to the *private robustness* to resist potential malicious disruptions, guaranteeing the output delivery while preserving sender anonymity. To better support a large user base, the system also supports both vertical and horizontal scaling. Our evaluation results show that *Gyges*'s communication-efficient shuffle designs outperform state-of-the-art MPC-based anonymous broadcast solutions, such as Clarion (NDSS' 22) and RPM (PETS' 23), while its shared trace technique can swiftly track down the misbehaving users (when necessary), giving orders of magnitude cost reductions compared to traceable mixnets (PETS' 24) that offers similar capabilities.

I. Introduction

Anonymous broadcast has long been a focal point of computer security research, allowing users to share messages while ensuring that network observers cannot learn their identities. Existing approaches in this area generally adhere to one of two classical paradigms: 1) *mix-nets*, where a chain of intermediary servers unlink the batched messages from senders by verifiably and sequentially rearranging them [1]; and 2) *DC-nets*, which obscure the true sender by having all users participate equally,



(a) Secure message mixing (b) Secure message tracking

Fig. 1. Overview of *Gyges*. Its core relay server group serves as the minimal unit capable of jointly performing secret-shared shuffling and tracing to ensure sender anonymity while enabling accountability for the misuse and abuse.

with only one sending a meaningful message while the rest submit empty dummies [2]. However, traditional designs often rely on computationally expensive public-key cryptography to ensure non-malicious participation of users and servers [3–5]. Despite recent advancements [6–9], these systems still suffer from relatively high latency, sometimes extending to tens of minutes, particularly when scaling to accommodate millions of users or very large messages.

In light of them, there has been growing interest in utilizing *multi-party computation* (MPC) techniques to build more efficient anonymous communication applications. For instance, MCMix [10] and Clarion [11] develop multi-party shuffle protocols to mimic the behavior of mix-nets chains; Riposte [12] and Express [13] refine server-assisted DC-nets variants based on recent multi-server function secret sharing primitive [14]. Compared to traditional solutions, MPC-based systems primarily rely on symmetric and/or information-theoretic cryptography, delivering competitive computational efficiency. However, they remain suboptimal in terms of inter-server communication costs, which is another dominating factor for the overall performance. Meanwhile, although anonymity plays a critical role in protecting privacy and promoting free expression, it has been repeatedly criticized for disinhibition and harmful abuse [15–19]. Internet trolls, for instance, would exploit anonymity as a smokescreen to disseminate racism, misinformation, or illicit content (e.g., child pornography and terrorist propaganda), all while evading accountability. In addition, ensuring in-protocol robustness – specifically *guaranteed output delivery* (G.O.D) against malicious disruptions from both compromised users and servers [20–22] – and scaling the system to serve a large, increasing user base [8, 9, 23], are essential for the long-term viability of anonymous services yet pose new domain-specific challenges. Failure to address them could substantially impede the practical adoption of such systems.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240268
www.ndss-symposium.org

System architecture. This work introduces *Gyges*, a secret-shared accountable anonymous broadcast system built upon a family of communication-efficient and privately robust MPC techniques over the finite ring.[1] At its core lies a non-colluding four-party *secret-shared relay*, as shown in Fig. 1. These relay servers collect client message shares and jointly execute two key protocols: 1) secret-shared shuffle that unlinks users from messages; and 2) secret-shared trace, triggered only when very inappropriate content is identified, that relinks the shuffled message(s) to their originating sender(s).

Cast in modern MPC terms, *Gyges* follows classical client-server, offline-online paradigms and adheres to recent popular small-party settings, specifically an honest-majority four-party model. Such setups are recognized for providing many most performant and deployable solutions [11, 24–26]. *Gyges* offers cryptographic security and guarantees output delivery against malicious adversaries capable of corrupting both the client and server. During the offline phase, relay parties precompute some message-independent correlated randomness – termed *shuffle correlation* and *trace correlation*. In the online phase, parties use these precomputed correlations to perform secure shuffling over user message shares and, when necessary, trace abusive content. Users (i.e., both broadcasters and subscribers) can interact with *Gyges* in a manner akin to how they engage with non-anonymous broadcast platforms like X/Twitter and Weibo, yet with enhanced anonymity and accountability features.

Technical overview. In *Gyges*, we reexamine and address the inherent tension among user anonymity, misbehavior accountability, and system performance in the context of MPC-based anonymous broadcast. we start from a basic observation that "a message vector $\mathcal{X}$ can be shuffled by applying a random row permutation matrix $\mathcal{M}$," a well-established shuffle paradigm in the literature [20–22, 26], denoted as

$$\mathcal{X}' = \mathcal{M} \circ \mathcal{X}.$$

Tracing, in turn, can be aptly modeled as a partial inversion of the shuffling operation. Specifically, we deem each column of the message-independent, secret-shared permutation matrix to serve as a partial shuffle correlation, while each row encodes a trace correlation. This framework 1) reduces secure shuffling to a sharing-based multiplication problem; and 2) simplifies secure tracing to a specialized sharing reconstruction problem, enhancing both efficiency and viability.

Our primary contribution lies in a pair of fast and lean four-party shuffling and tracing protocols (4PS and 4PT) over the finite ring $\mathbb{Z}_2$ and $\mathbb{Z}_{2^\ell}$, specifically optimized to minimize its inter-server communication. Central to this design is a novel *Boolean permutation correlation* (BPC), sampled from $\mathbb{Z}_2^{N\times N}$ instead of naïve $\mathbb{Z}_{2^\ell}^{N\times N}$ space. This retains somewhat inherent sparsity to lower both communication and computation costs. Our optimized protocols avoid lifting the secret-shared BPC elements from $\mathbb{Z}_2$ to $\mathbb{Z}_{2^\ell}$ during the online computation and, crucially, achieve *zero* online communication or interaction – enabling what we term "silent shuffling" (refer to Table I).

Meanwhile, *Gyges* efficiently maintains robustness against malicious disruptions while fully preserving anonymity. MPC systems upholding *security with abort* are vulnerable to censorship, as adversaries may abort the protocol to block message delivery [20]. Protocols with *traditional robustness* either incur high system overhead or may risk anonymity by relying on a process that reveals secrets to a non-malicious party [24]. To efficiently achieve G.O.D for anonymous services, we draw on recent *private robustness* notion by Dalskov *et al.* [25] and develop a suite of customized robust protocols with minimal overhead, and without compromising anonymity.

Thirdly, we explore the scalability of *Gyges* with a focus on enhancing anonymous communication service throughput. Specifically, we note that our silent shuffling protocol design works by having relay parties silently perform multiple independent computations in parallel. This facilitates the effective scaling of *Gyges* in two ways: 1) *horizontally*, by adding more computing servers per party and distributing workload, and 2) *vertically*, by utilizing GPU devices for faster processing.

Main contributions. Technically summarized below:

- **Communication-efficient silent protocol design:** In §IV and §V, we propose a pair of shared four-party shuffling and tracing protocols over the finite ring(s), specifically designed to compress offline communication, minimize online interaction, and optimize computation cost, resulting in concretely efficient solutions.
- **Efficient privately-robust protocol design:** Building on the above semi-honest constructions, we extend privately robust variants in §VI. This can efficiently guarantee the output delivery of shuffled (or traced) messages, while fully preserving sender anonymity.
- **Throughput-oriented scalable protocol design:** In §VII, we revisit the scalability goal for anonymous communication and explore scaling our MPC-based designs both vertically and horizontally for better performance.

Evaluation results. Putting all ideas together, we get *Gyges*, a secret-shared accountable anonymous broadcast framework. To demonstrate it, we evaluate *Gyges* and compare the results against state-of-the-art MPC-based anonymous broadcast systems such as Clarion [11] and RPM [21], and recent traceable mixnets [28] that provides a similar tracking capability. For mixing 10^5 messages, each ranging from 32 B to 1 KB, *Gyges* outperforms Clarion by $1.41\times$ - $2.32\times$ regarding the end-to-end latency, while significantly lowering the online inter-server communication cost. When tasked with tracking down a (or a set of) shuffled message(s) flagged for abusive content, *Gyges* delivers several orders of magnitude improvement over the traceable mixnets (see details in §VIII and §IX).

Use cases, limitations, and ethical issues. *Gyges* can support a variety of practical applications, such as anonymous social network platform [16], metadata-hiding document sharing [29, 30] and instantiating other primitives like point-to-point two-

[1]The term "Ring of Gyges" derives from Plato's *Republic*, where the Gyges ring grants its wearer invisibility (anonymity). In alignment with the ethical stance that such power should be wielded responsibly for greater social good, we propose the secret-shared accountable anonymous broadcast system *Gyges* and uphold that "with great anonymity comes great accountability.

TABLE I
COMPARISON OF THE CORE MULTI-PARTY SHUFFLE PROTOCOLS BETWEEN *Gyges* AND OTHER MPC-BASED ANONYMOUS BROADCAST SCHEMES.

Ref.†	Adversary Structure	Security Guarantees	Sharing Scheme‡	User-server		Server-Server (Offline)$^\natural$	Server-Server (Online)			Selective Traceability
				Comp.	Comm.	Comm.	Comp.	Comm.	Round$^\diamond$	
AsynchroMix [22]	$r < n/3$	Fairness	Shamir	$O(1)$	$O(\ell)$	$O(N \log^2 N\ell)$	$O(N \log^2 N)$	$O(N \log^2 N\ell)$	$\log^2 N$	✗
PowerMix [22]	$r < n/3$	Fairness	Shamir	$O(1)$	$O(\ell)$	$O(N\ell)$	$O(N^3)$	$O(N\ell)$	2	✗
Blinder [20]	$r < n/4$	Robustness	Shamir	$O(\sqrt{N})$	$O(\sqrt{N}\ell)$	$O(N\ell)$	$O(N^2)$	$O(N\ell)$	4	✗
Rabbit-Mix [26]	$r < n/4$	Robustness	Shamir	$O(\sqrt{N})$	$O(\sqrt{N}\ell)$	$O(N\ell)$	$O(N^2)$	$O(N\ell)$	4	✗
RPM-I [21]	$r < n/3$	Robustness	Shamir	$O(1)$	$O(\ell)$	$O(N^2\ell)$	$O(N^2)$	$O(N\ell)$	2	✗
RPM-II [21]	$r < n/3$	Robustness	Shamir	$O(1)$	$O(\ell)$	$O(N^2\ell)$	$O(N^2)$	$O(N\ell)$	2	✗
RPM-III [21]	$r < n/3$	Robustness	Shamir	$O(1)$	$O(\ell)$	$O(N^{1.5}\ell)$	$O(N^{1.5})$	$O(N\ell)$	q	✗
Ruffle [24]	$n=3;r=1$	Robustness	Replicated	$O(1)$	$O(\ell)$	$O(N\ell)$	$O(1)$	$O(N\ell)$	2	✗
Clarion [11]	$n=2;r=1$	Security with abort	Additive	$O(1)$	$O(\ell)$	$O(N\ell)$	$O(N)$	$O(N\ell)$	6	✗
Gyges	$n=4;r=1$	Private robustness	Replicated	$O(1)$	$O(\ell)$	$O(N^{1.5})$	$O(\alpha N)$	0	0	✓

† Let n and r denote the number of computing parties and corrupted parties, respectively; N denote the message batch size; ℓ denote the bit length of each message; α represent the non-zero element number in each column of a BPC matrix instance, which depends on how BPC is sampled from the unbiased (or biased) distribution of $\{0,1\}$; ✗ indicates that the work does not explicitly consider the accountability issue, while ✓ indicates that our work does.
‡ Replicated refers to replicated secret sharing (RSS) [27]. Both *Gyges* and Ruffle leverage replicated sharing techniques over the finite ring in a small-party setting to achieve lightweight robustness guarantees, though they employ very different sharing semantics and protocol specifications (details are discussed in §II and §VI).
$^\natural$ Inspired by Clarion's use of a semi-honest helper party to facilitate shuffling, *Gyges* adopts a similar setup to boost the entire system. Compared to the traditional dealer, this helper party is solely responsible for generating base materials, which are subsequently processed across relay parties as target correlations (see §V-A for details).
$^\diamond$ By definition, the online round (or communication) complexity typically excludes the message sharing and reconstruction phases. RPM-III requires q rounds, where q corresponds to the depth of a square permutation network in its design. Clarion requires 2 rounds in the semi-honest setting and 6 rounds in the malicious setting. As a clear separation, *Gyges* enables silent shuffling with *zero* inter-server communication/interaction in the online phase.

way anonymous message exchanging [10, 31], particularly in the environment prioritizing communication efficiency.

In line with many prior anonymous communication systems, we render a few rational trade-offs among system assumption, performance, and security in *Gyges*. Importantly, we position the accountability capability as a necessary mechanism to deter the misuse of anonymity and promote responsible behavior, rather than as a loophole to undermine anonymity itself. This aligns with emerging research on content moderation [17, 32–35]. Further discussions on the system limitations and ethical considerations of *Gyges* are provided in §X.

Broader interest. The secret-shared shuffle protocol also acts as a versatile building block applicable to other domains, such as oblivious sorting [36], cryptocurrency tumblers [37], shuffle differential privacy [38], and other metadata-sensitive applications where participants wish to hide their involvement [29]. The lightweight shuffling and tracing techniques developed in *Gyges* may offer heuristic value for them.

II. RELATED WORK

Anonymous communication has come a long way, emerging with foundational paradigms such as mix-nets or DC-nets [1, 2] and evolving significantly to deployed systems like Tor or I2P [39, 40]. We next categorize and review a few systems and techniques most relevant to *Gyges*.

Anonymity from MPS. The multi-party shuffle (MPS) protocol simulates the functionality of a mix-nets chain, securely shuffling batched user messages while avoiding high computational overhead from public-key computations. Early designs, such as MCMix [10] and AsynchroMix [22], adopt a switching network paradigm to obliviously swap elements multiple times, typically requiring $O(\log N)$ interaction rounds and less efficient when N is large. To reduce it, Clarion [11] draws on a constant-round shuffle paradigm by Chase *et al.* [41] (aka CGP protocol) and introduces a semi-honest assisting server to facilitate shuffling. However, it obtains security with abort, which is vulnerable to in-protocol DoS attacks – the adversary can arbitrarily abort to prevent honest users from delivering messages – and may be of other insecurity issues [42].

To obtain additional robustness guarantees against malicious servers, we see another rich line of work, e.g., AsynchroMix (PowerMix) [22], Blinder [20], and RPM [21], adopts standard (n,t)-Shamir secret sharing. Despite featuring very succinct forms and built-in robustness guarantees, these works over the finite field $\mathbb{F}_p$ imply massive hidden computational costs in 1) frequent modulo reductions with additional checks [43] and 2) frequent interpolations and polynomial degree reductions. For better computational efficiency and simplicity, some MPC deployers advocate for small-party setups over the finite ring. For example, Ruffle [24] demonstrates a concrete three-server solution with lightweight G.O.D. However, this is achieved by utilizing a recent joint message passing primitive to identify a non-malicious party and having it learn all secrets to perform computations in the clear, which works well in the traditional MPC context, yet risks violating anonymity guarantees. In particular, to prevent malicious users from submitting malformed messages, many prior works generally assume the availability of reliable broadcast channels [20–22, 24], which implies non-negligible hidden costs yet are largely ignored.

Anonymity from PIR. The private information retrieval (PIR) protocol allows users to obliviously read from or write somewhat virtual addresses, which functionally enables a DC-nets and motivates another rich line of studies, such as Riffle [44], XPIR [45], and Pung [46]. Recent solutions like Riposte [12], Express [13], and Sabre [47] further utilize distributed point function (DPF) [14] to lower communication costs.

Compared to MPS-based approaches, PIR systems usually excel at handling individual messages but are vulnerable to severe message collisions when multiple users are writing to the same address. Techniques like Reed-Solomon coding [12, 48] and large-domain virtual address sampling [13] can partially mitigate collisions, but adversaries capable of compromising both servers and users can still exploit this weakness. Besides, PIR systems usually impose much higher client-side costs and are non-robust to server deviations.

Anonymity with scalability. Supporting a large user base is essential for all anonymous communication systems to ensure robust security, as encapsulated in the adage *anonymity loves company*. Building on this principle, recent works Atom [23], XRD [9], Trellis [8], and Streams [49] explored the horizontal scaling of mix-nets by running multiple small mixing chains in parallel to boost message shuffling throughput. Meanwhile, Talek [50] and Blinder [20] employ vertical scaling strategies, showcasing GPU acceleration in performance optimizations.

In contrast, MPC systems typically consider another flavor of scalability notion for better flexibility and robustness against malicious disruptions (or t-out-of-n adversary structure [20]). However, the aforementioned "system scaling" and traditional "MPC scaling" designs do not inherently comply. In the latter, extending base protocols to an n-party variant also increases the total share size and computational complexity, thus often improving no service throughput. To bridge this gap, we thus explore them both in our system contexts.[2]

Anonymity with accountability. A substantial body of literature has explored mitigating the potential misuse of anonymity by integrating accountability mechanisms. Von Ahn *et al.* [51] introduce the concept of *selective traceability*. Recent traceable mixnets [28] technically enable mapping encrypted egress traffic back to its ingress source. Dissent [7] and its follow-ups [52, 53] extend DC-nets to trace potential malicious actors. Meanwhile, several works enable content moderation in end-to-end encrypted messaging services [17, 33–35, 54], relying on accountability strategies like threshold-based user reporting or server auditing. In contrast to the above solutions relying on costly public-key cryptography, Eskandarian [32] introduces a shared message franking mechanism, demonstrating significant performance improvements.

Inspired by them, we propose a secret-shared tracing protocol that can swiftly identify the origin of broadcast messages containing highly inappropriate content. To mitigate the inherent risks of accountability misuse, such as the false reporting of benign users, we also introduce necessary pre- and post-tracing safeguards (see details in §V-D).

III. Background

Secure multi-party computation (MPC). Client-server MPC allows multiple users C_i to secretly share their private data x_i with a group of servers P_j, who then jointly compute a function f without revealing the inputs. While many MPC protocols operate over finite fields $\mathbb{F}_p$, trading off interaction and communication, recent protocols have increasingly focused on the finite ring $\mathbb{Z}_{2^\ell}$ due to better computational efficiency on modern hardware [43, 55–57]. Additionally, many modern MPC protocols adhere to the classic preprocessing model [58], especially in small-party settings [11, 24], where a few servers precompute some input-independent correlated randomness to enhance concrete online efficiency and simplicity.

Replicated secret sharing (RSS). A t-out-of-n RSS scheme divides a secret x into n pieces, distributing them to n parties such that any t of them can reconstruct the secret, while fewer cannot [27]. A value $x \in \mathbb{Z}_{2^\ell}$ is said to be $(4, 2)$-RSS or $[\cdot]$-shared, if there exist $[x]_i \in \mathbb{Z}_{2^\ell}$, such that $x = \sum_i [x]_i$ and each P_j holds $[x]_i$ where $i, j \in \{1, 2, 3, 4\}$ and $i \neq j$. To distinguish between the arithmetic and Boolean sharing, the latter is said to be $\langle x \rangle_i$-shared, where $x_i \in \mathbb{Z}_2$. Similarly, we say arithmetic $(4, 3)$-RSS to be $[\![\cdot]\!]$-shared. For simplicity, we may slightly abuse the sharing notations and will specify it when necessary.

Compared to the additive sharing scheme, RSS offers extra robustness potential via redundancy. Unlike Shamir sharing, RSS achieves much better computational efficiency, albeit with increased storage and communication costs as n and t grow.

Four-server one-bit distribute point function (DPF). A DPF scheme among n parties with 1-bit output, evaluated over the full domain, consists of an algorithm pair $(\mathsf{Gen}, \mathsf{Evalall})$ [14], where $k_{i \in \{1,\ldots,n\}} \leftarrow \mathsf{Gen}(m, 1)$ samples a tuple of DPF keys for a binary point function and $\mathsf{Evalall}(k_i, x)$ evaluates on every point x in a full domain of size N. The result of $\mathsf{Evalall}$ is the binary shares of an N-dimension one-hot vector $e^{(m)} = [0, ..., 1, ..., 0]^T$, where m is the index of 1, denoted as,

$$e^{(m)} = \bigoplus_{i=1}^{i=4} \mathsf{DPF.Evalall}(k_i, x)$$

The most efficient n-party DPF scheme for $n > 2$ remains Boyle *et al.*'s original matrix-based construction [14], with the compressed key size of $O(\sqrt{N})$. *Gyges* makes the black-box use of a minimal 4-server 1-bit DPF to build BPC correlation.

IV. System Model

A. Design Goals and Non-Goals

The system is expected to exhibit following key properties:

- *Sender anonymity:* The system must ensure that senders remain securely unlinked from their messages within each broadcast round, as defined by [59, 60]. In this process, no external observer or system participant can attribute specific mixed messages to their originators.
- *Selective traceability:* The system must enable authenticated tracing of abuse messages back to their sources, while maintaining the anonymity of others, following the model in [28, 51]. This process should be confined to moderation policies, ensuring traceability is not misused.
- *Private robustness:* The system must resist the in-protocol DoS attack by malicious adversaries, ensuring guaranteed output delivery. In doing so, this should also preserve the anonymity of honest participants (both clients and servers), in line with principles established in [25, 61].
- *System scalability:* To accommodate practical service demands, the system must scale well to enhance throughput and serve a large user base, as suggested by [9, 20].

[2] To emphasize scalability designs in *Gyges*, we differentiate the term *party* from *server*. Each party can host multiple computing servers. For simplicity, however, we may use them interchangeably unless otherwise specified.

These goals, along with the ideal functionality of our system *Gyges*, are formally described in §IV-C. It is important to note that *Gyges* focuses primarily on preserving sender anonymity within a single broadcast round. Addressing intersection attacks, where adversaries correlate multiple broadcast rounds to narrow the anonymity set, falls outside the scope of this work. However, this limitation is not unique to *Gyges*; it is a common constraint shared by similar systems [10, 11, 20–22, 24, 26]. Furthermore, as with most prior MPC systems, *Gyges* does not aim to fully resist network-level failure attacks. Non-malicious relay parties are expected to stay online and interact faithfully throughout the protocol execution. To accommodate potential failures, we allow each relay party to operate multiple servers, requiring only that at least one relay server per party remains consistently available (see §VII).

These proposed goals and non-goals generally outline the operational contexts and ideal deployment scenarios of our system. While a determined nation-state adversary could still censor users or disrupt anonymous broadcast services by disabling the relay server, addressing such a powerful adversary is outside our scope. Instead, *Gyges* strikes a pragmatic balance between sender anonymity, misbehavior accountability, and system availability, making it particularly well-suited for social network platforms that seek to offer usable anonymity guarantees while adhering to legal and ethical standards.

B. Threat Model and Security Assumption

We let $C_{i\in\{1,2\ldots,N\}}\in\mathcal{C}$ be client, $x_{i\in\{1,2\ldots,N\}}\in\mathcal{X}$ be C_i's message, $P_{i\in\{1,2,3,4\}}\in\mathcal{P}$ be relay party, H be semi-honest helper party, $\mathcal{M}\in\mathbb{Z}_2^{N\times N}$ denote a permutation matrix, $\mathcal{S}/\mathcal{S}'$ be the input and output sharing schemes, specifically referring to $(4,2)/(4,3)$-RSS in our work, $\mathcal{X}'$ denote mixed messages, and $\mathcal{A}$ be malicious adversaries that can actively deviate.

We consider $\mathcal{A}$ capable of compromising both the user and relay party in a rational honest-majority setting. Let $\mathcal{C}_c$ be the corrupted user set and $\mathcal{P}_c$ be the corrupted party set. We assume that the number of corrupted users satisfies $|\mathcal{C}_c|<\frac{|\mathcal{C}|}{2}$ (for a reasonable reporting threshold) and the number of corrupted parties is $|\mathcal{P}_c|=1<\frac{|\mathcal{P}|}{2}$ (ensuring in-protocol robustness). Following many prior MPC-based systems [11, 21, 22, 24], we assume that relay parties do not collude with one another or with the additional helper party. However, a corrupted relay party may collude with (or control) multiple compromised clients. Despite this, we note that neither the corrupted users nor the corrupted relay server can successfully de-anonymize a benign user by abusing the tracing function. In addition, unlike some prior works that assume the availability of expensive (reliable and consistent) broadcast channels, all client-server and server-server communications in *Gyges* are conducted via minimal point-to-point channels [62]. Further details regarding the attacker's capabilities and the de-anonymization attack analysis can be found in Appendix C and Appendix D.

C. Shared Accountable Anonymous Broadcast Functionality

Definition 1 (SSAAB): Secret-shared accountable accountable anonymous broadcast (SSAAB) functionality interacts with clients $\mathcal{C}$, relay parties $\mathcal{P}$, helper party H, and malicious adversary $\mathcal{A}$, defined as a tuple of algorithms,

- $\mathsf{SSAAB.Bpc}()$: Upon receiving the preprocessing signal,
  - H generates a secret-shared random Boolean permutation matrix $\mathcal{S}(\mathcal{M}')$, sends corresponding shares to $\mathcal{P}$.
  - $\mathcal{P}$ agree on a common random permutation π and compute correlation as $\mathcal{S}(\mathcal{M})=\pi(\mathcal{S}(\mathcal{M}'))$.
- $\mathsf{SSAAB.Msg}()$: Upon receiving the messaging signal,
  - C_i sends message shares $\mathcal{S}(x_i)$ to $\mathcal{P}$.
  - $\mathcal{P}$ collect, combine all $\mathcal{S}(x_i)$ and generate consistent secret-shared user message set $\mathcal{S}(\mathcal{X})$.
- $\mathsf{SSAAB.Mix}()$: Upon receiving the shuffling signal,
  - $\mathcal{P}$ jointly evaluate the secure inner product on every BPC row $\mathcal{S}(\mathcal{M}_{(i)})$ with the message set $\mathcal{S}(\mathcal{X})$, then set results as the shuffled message set, denoted as $\mathcal{S}'(\mathcal{X}')$.
- $\mathsf{SSAAB.Rec}()$: Upon receiving the reconstruction signal,
  - C_i retrieves $\mathcal{S}'(\mathcal{X}')$ from $\mathcal{P}$, and recovers $\mathcal{X}'$.
- $\mathsf{SSAAB.Tra}()$: Upon receiving the tracking signal,
  - $\mathcal{C}$ report the abuse message, denoted as x'_a.
  - $\mathcal{P}$ and H jointly audit the content of x'_a under moderation policy D, and if warranted, $\mathcal{P}$ trigger the tracking process for the abuse message.
  - $\mathcal{P}$ jointly recover the row $\mathcal{M}_{(a)}$, identify the misbehaving user and perform appropriate management actions.

The SSAAB functionality can 1) unlink $\mathcal{C}$ from $\mathcal{X}$ (and $\mathcal{X}'$) via secret-shared shuffling; and 2) relink specific, inappropriate message $x'_a\in\mathcal{X}'$ to its true sender C_i via secret-shared tracing. Its correctness and security depend on the underlying MPC specification and the inputs' well-formedness. A protocol Π is considered secure if, for any $\mathcal{A}$, there exists an ideal simulator Sim such that their views are computationally indistinguishable, denoted as $\mathsf{REAL}_{\Pi,\mathcal{A}}\approx_c\mathsf{IDEAL}_{\mathcal{F},\mathsf{Sim}}$.

More formally, the probability that $\mathcal{A}$ can guess whether a mixed message x' was sent by a specific C_i should satisfy,

$$\mathsf{Pr}[\mathcal{A}(x',C_i)=1]\approx_c\mathsf{Pr}[\mathcal{A}(x',C_j)=1],\ \text{for}\ i\neq j.$$

The selective accountability goal mandates that $\mathcal{P}$ must deterministically identify the sender of an inappropriate message x' in accordance with policy set $\mathcal{D}$, while ensuring the anonymity of benign users is rigorously preserved. denoted as

$$\mathsf{Pr}[\mathcal{D}(x'_a,\mathcal{M}_{(a)})=p]\approx_c p,\ \text{for}\ p\in\{0,1\}.$$

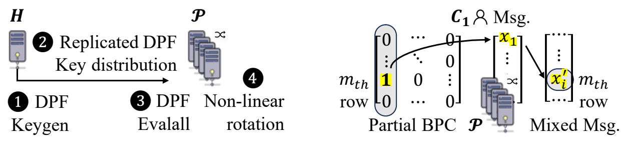
To capture robustness guarantees, for any honest user $C_i\notin\mathcal{C}_c$ and his/her message x_i, the failure probability of delivering its correctly processed output despite adversarial disruptions should be negligible, denoted as below,

$$\mathsf{Pr}[\mathcal{A}(x_i,\mathcal{M}_i)=\perp]\approx_c 0,\ \text{for}\ C_i\notin\mathcal{C}_c.$$

Let n_i be the number of servers managed by party P_i, and b_i the computational capacity per server. As n_i and b_i increase, the service throughput T scales polynomially:

$$T\propto\mathsf{poly}(\min(n_i),\min(b_i)),\ \text{for}\ i\in\{1,2,3,4\}.$$

For more detailed system analysis, see §VIII and Appendix C.



(a) Aided BPC generation (b) BPC-based shuffling

Fig. 2. *Gyges*'s BPC generation and BPC-based secret-shared shuffling.

V. Fast and Lean Shuffling and Tracing

We first introduce our 4PS/4PT in the semi-honest setting and extend them for robustness (§VI) and scalability (§VII).

A. Correlated Boolean Permutation Randomness

Definition 2 (BPC): Let $\mathcal{M} \in \mathbb{Z}_2^{N\times N}$ be a randomly, uniformly sampled Boolean permutation matrix of N-by-N size. We define Boolean permutation correlation (BPC) as a tuple of random Boolean matrix shares $(\langle\mathcal{M}\rangle_1, \langle\mathcal{M}\rangle_2, \langle\mathcal{M}\rangle_3, \langle\mathcal{M}\rangle_4)$ correlated via *element-wise exclusive or*, such that:

$$\mathcal{M} = \bigoplus_{i=1}^{i=4} \langle\mathcal{M}\rangle_i,$$

each row and column of $\mathcal{M}$ has exactly one entry of 1, with all others being 0. Formally, for all $k \in \{1, ..., N\}$, $\bigoplus_{j=1}^{j=N} \mathcal{M}_{j,k} = 1$ and for all $j \in \{1, ..., N\}$, $\bigoplus_{k=1}^{k=N} \mathcal{M}_{j,k} = 1$.

Rationales behind BPC. Akin to the Beaver triples in secure multiplications [63], BPC is specifically designed to facilitate secure shuffling tasks. Notably, BPC depends only on message number N and is independent of message size ℓ. This results in compressed offline overhead, making it highly efficient for mixing extremely large messages (e.g., 1 – 2 KB per chunk in cryptocurrency transactions [64]). Additionally, BPC can maintain somewhat inherent sparsity, comprising only 0s and 1s, which we further leverage to optimize online computational performance (as detailed in §V).

As illustrated in Fig. 2, we let $\mathcal{M}_k$ denote the k-th column of a BPC matrix. The index of the value 1 in $\mathcal{M}_k$ indicates the target virtual address to which message x_k is shuffled. Similarly, $\mathcal{M}_{(k)}$ refers to the k-th row, where the index of the value 1 represents the virtual address from which message x'_k was shuffled. For clarity, we refer to these as *shuffle correlation* and *trace correlation*, respectively.

Well-formed BPC generation via DPF. Inspired by Clarion's use of a semi-honest assisting server that facilitates the offline protocol [11], we employ a helper party H to generate and distribute required "well-formed" base BPC shares. This natural yet effective approach significantly reduces the preprocessing cost and eliminates potential message collisions [11, 12, 20], which is crucial for ensuring the accountability goal.

Specifically, each column in BPC is a binary shared one-hot vector $\langle e^{(k)}\rangle$. To generate this, H can encode k into 4-party 1-bit DPF keys and distribute these keys to $\mathcal{P}$ in a $(4, 2)$-RSS format, resulting in $O(\sqrt{N})$-bit communication per column. $P_j \in \mathcal{P}$ then evaluates these key tuples over the full domain to obtain base BPC shares, with a computational complexity of $O(N)$. This arithmetic-to-one-hot sharing conversion effectively compresses the communication costs.

Protocol $\Pi_{\mathsf{BPC}}[H, \mathcal{P}]$: BPC Generation

Input: Security parameter λ, message batch size N.
Output: $\langle\mathcal{M}\rangle$-shared permutation matrix.

Protocol: Given a security parameter λ and batch size N, the protocol proceeds as follows,

1) **Base BPC generation:**
   - H generates a random N-by-N Boolean permutation matrix $\mathcal{M}' \leftarrow 1^\lambda$.
   - For each $i \in \{1, ..., N\}$, H encodes i-th column of $\mathcal{M}'$ into a DPF key tuple k_{ij} where $j \in \{1, 2, 3, 4\}$, and distributes each key k_{ij} to $\mathcal{P}$, following the $(4, 2)$-RSS format.
   - All $P_j \in \mathcal{P}$ locally evaluate all received k_{ij} over the full domain, obtain $\langle M'\rangle_k$ for $i \in \{1, ..., N\}$ and $j, k \in \{1, 2, 3, 4\}$, where $j \neq k$.
2) **Non-linear BPC rotation:**
   - $\mathcal{P}$ sample a common permutation $\pi \leftarrow 1^\lambda$.
   - All $P_j \in \mathcal{P}$ compute local shares $\pi(\langle\mathcal{M}'\rangle_k)$ on $j, k \in \{1, 2, 3, 4\}$ where $j \neq k$, and set it as $\langle\mathcal{M}_k\rangle$.

Fig. 3. Offline BPC generation and non-linear rotation protocol.

Non-leaky BPC construction via non-linear rotation. The BPC instance discussed above is, unfortunately, not private to H. To mitigate this, we allow $\mathcal{P}$ to rotate each column with different offsets (i.e., non-linear rotation), thus deriving a new private BPC instance. This non-linear rotation is performed by having $\mathcal{P}$ sample a random permutation π and apply it to non-interactively switch local column identifiers, rather than switching the real column shares themselves. This approach ensures concretely efficient execution. As depicted in Fig. 3, when Π_{BPC} terminates, each P_j holds well-formed $\langle\mathcal{M}\rangle$.

Our BPC generation protocol relies on a helper party H for DPF key tuple sampling and distribution. While Doerner and Shelat [65] propose a distributed DPF key generation protocol that eliminates the need for H, it introduces significant preprocessing overhead and scalability challenges. Another line of work [66–68] explores the use of trusted hardware for efficient correlation generation, albeit at the cost of added complexity and assumptions, which could also support our design. In this work, we retain H, following a rationale similar to that of Clarion [11], to prioritize system availability.

The correctness and security of Π_{BPC} can both be reduced to those of the underlying DPF evaluation process (see [14]).

B. Secret-Shared Messages Collection

Once message x_i is ready, C_i distributes it to $\mathcal{P}$. Each P_j learns three out of four shares. This sharing redundancy grants robustness but brings message inflation: it requires 12ℓ bits in total to represent per ℓ-bit message. Unlike conventional MPC use cases, where repeated computations over the same shared data entry can amortize such sharing costs, messages in anonymous communication are typically one-time and rarely reused, limiting amortization opportunities.

Pseudorandom sharing. To compress such user-server communication, we employ classic pseudorandom secret sharing (PRSS) technique [27] (see Fig. 14 in Appendix A), allowing most shares to be generated by servers, reducing the per-message communication from the original 12ℓ to 3ℓ.

Chunking for long messages. Standard CPU and GPU platforms can efficiently handle arithmetic modulo 2^l, especially for certain, small $\ell \in \{32, 64\}$. However, real-world systems, such as X/Twitter and Zcash, often deal with larger message sizes (e.g., up to 280 B per tweet [69] and 2 KB per Zcash transaction [64]). To efficiently bridge this gap and support messages with arbitrary lengths, we incorporate a chunking mechanism that breaks down long messages into smaller, more computation-friendly pieces. Computations are then uniformly applied to all chunks of a message, ensuring efficiency and flexibility across varying message sizes.

C. Sender Anonymity via Silent Shuffling

Once both the BPC and shared message set are prepared, the relay servers can jointly execute secret-shared shuffling.

Detour: shuffling via multiplication over $\mathbb{F}_p$ or $\mathbb{Z}_{2^\ell}$. Rearranging messages using a private permutation matrix can be formulated as a secure dot product between a matrix row $\mathcal{M}(i)$ and the message vector $\mathcal{X}$ (see Fig. 2(b)). To accomplish this, the elements of $\mathcal{M}(i)$, initially in $\mathbb{Z}_2$, need to be lifted to the same domain as the messages, i.e., $\mathbb{Z}_{2^\ell}$.

Additionally, in $(4,2)$-RSS, each single multiplication gate can be expanded and evaluated as follows:

$$xy = (\sum_{i=1}^{i=4}[x]_i)(\sum_{j=1}^{j=4}[y]_j) = \overbrace{\sum_{k\in\{1,2,3,4\}}[x]_k[y]_k}^{\text{subterms known to 3 servers}}$$
$$+ \underbrace{\sum_{u,v\in\{1,2,3,4\}}^{u\neq v}([x]_u[y]_v + [x]_v[y]_u)}_{\text{cross subterms known to 2 servers}}.$$

Throughout this process, all subterms τ can be locally computed, either by two or three servers. τ known to three servers naturally adhere to $(4,2)$-RSS, while τ known to only two servers require one more resharing round to comply with $(4,2)$-RSS [25]. For example, P_3 and P_4 compute $\tau = [x]_1[y]_2 + [y]_1[x]_2$ and reshare the result by setting $[\tau]_1$ as a random r, known to $P_{\{2,3,4\}}$, and sending $[\tau]_2 = \tau - r$ to $P_{\{1,3,4\}}$, thereby converting $(4,3)$-RSS subterms back to $(4,2)$-RSS ones to facilitate follow-up computations. Recent advances have further demonstrated that secure dot products can be performed within a single online interaction round (independent of N) by combining all linear subterms with the same index into a larger subterm [20, 25].

Sparsity-aware optimization. Instead of lifting BPC elements from $\mathbb{Z}_2$ to $\mathbb{Z}_{2^\ell}$ and performing computations on all entries, we focus exclusively on the non-zero entries. Specifically, each

Protocol $\Pi_{\mathsf{4PS}}[\mathcal{P}]$: Message Shuffling

Input: $[\mathcal{X}]$-shared messages and $\langle\mathcal{M}\rangle$-shared BPC.
Output: $[\![\mathcal{X}]\!]$-shared shuffled messages $\mathcal{X}'$.

Protocol: The protocol works as follows,

1) **LUT construction:** For each row $i \in \{1, 2, ..., N\}$, P_j builds non-zero index LUT $\mathcal{T}_{ij}$ for partial BPC share $\langle\mathcal{M}_{(i)}\rangle_k$ where $j, k \in \{1, 2, 3, 4\}$ and $j \neq k$.
2) **Sparsity-aware inner product:** For $l \in \mathcal{T}_{ij}$,
   - P_j locally computes the term $\sum_l([x_l]_u)$ where $u \in \{1, 2, 3, 4\}$ and $u \neq j$ as its local subterm known to three parties.
   - P_j locally computes the term $\sum_l([x_l]_v + [x_l]_w)$ where $v, w \in \{1, 2, 3, 4\}$, $v \neq w \neq j$, and $v < w$ as its cross-subterm known to two parties.
3) **Silent shuffling:** Repeat the step 2) for each row $i \in \{1, 2, ..., N\}$.

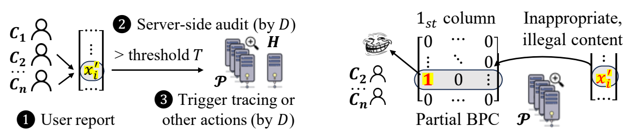
Fig. 4. Silent and sparsity-aware online shuffling protocol.

column of a BPC share is represented using a lookup table (LUT) that stores only the indices of the entries with a value of 1. At the same time, users should encode their ℓ-bit messages in a bitwise Boolean-shared manner rather than the original arithmetic-shared format. For simplicity and consistency, we slightly abuse and retain the semantics of arithmetic sharing (i.e., $[\cdot]$) and addition (i.e., $\sum, +$) to denote de facto bitwise Boolean sharing schemes and exclusive OR operations in later sections and Fig. 4. The small LUT can be precomputed during the offline phase, allowing its construction cost to be amortized across chunking scenarios. For instance, when splitting ℓ-bit messages into d chunks, the shuffling of all chunks $\mathcal{X}[i]$ (where $i \in \{1, 2, \ldots, d\}$) can leverage the same BPC, incurring no additional offline overhead.

Silent execution and message reconstruction. In the vanilla 4-party multiplication protocol proposed by Dalskov *et al.* [25], at least one interaction round is required to reshare $[\![\cdot]\!]$-shared intermediate results into the same format as $[\cdot]$-shared inputs, i.e., converting $(4,3)$-RSS to $(4,2)$-RSS. This resharing step is essential in generic MPC scenarios, as the output shares are often used as inputs for subsequent computations. In our anonymous broadcast setting, however, we observe that the output of Π_{4PS} only needs to be reconstructed, eliminating the necessity for an additional round to ensure sharing format consistency. For example, any three servers can jointly reconstruct $[\![\mathcal{X}']\!]$ directly, without first converting it into $[\mathcal{X}']$ before performing reconstruction.

By doing so, we eventually derive a shuffling protocol that requires no online interaction – termed the *silent shuffling*. This customized approach reduces the online communication to an optimal *zero*, significantly enhancing the system efficiency.

The correctness and security of Π_{4PS} in Fig. 4 are grounded in the underlying non-interactive multiplication protocol, we also detail them in §VIII and Appendix C.



C_1
C_2
C_n
x_i'
❷ Server-side audit (by D)
> threshold T
H
$\mathcal{P}$
❸ Trigger tracing or other actions (by D)
❶ User report
1_{st} column
Inappropriate, illegal content
Partial BPC

(a) Abuse report and audit (b) BPC-based selective tracing

Fig. 5. *Gyges* employs both 1) threshold-based user report, and 2) server audit mechanisms to identify and flag inappropriate messages. The servers then trace and manage misbehaving users in accordance with policy D.

Protocol $\Pi_{\mathsf{4PT}}[\mathcal{C}, \mathcal{P}, H]$: Message Tracing

Input: $[\mathcal{M}]$-shared of BPC, a shuffled message x'_a, moderation policy set $\mathcal{D}$, and report threshold η.
Output: The identifier k of x'_a's sender or abort.

Protocol: The protocol proceeds as follows,

1) **Users reporting and servers auditing:** For a message x'_a that has been reported by users exceeding a predefined threshold η,
   - $\mathcal{P}$ and H audit x'_a's content according to $\mathcal{D}$ and set an abuse flag flag_a under majority consensus.
   - If $\mathsf{flag}_a = 1$, P_j sends the share $\langle \mathcal{M}_{(a)} \rangle_k$ to P_k where $j, k \in \{1, 2, 3, 4\}$ and $j < k$. Else, abort.
2) **Selective tracing:** $\mathcal{P}$ reconstruct $\mathcal{M}_{(a)}$ and return the index of 1 entry under a majority consensus.
3) **Misbehavior management:** $\mathcal{P}$ jointly handle misbehaving clients according to $\mathcal{D}$.

Fig. 6. Information-theoretic tracing and misbehavior management protocol.

D. Accountability via Information-Theoretic Tracing

In this section, we address the dual problem of the secure shuffling protocol that achieves sender anonymity – a secure tracing protocol that enables accountability.

Detour: traceability using public-key cryptography. A tracing protocol aims to offer a mechanism to relink a user C_i with an output message x'_j accused of abuse, while preserving the anonymity guarantees for all other participants. Traditionally, this goal has been achieved using heavy public-key cryptographic techniques like zero-knowledge proofs – either by enabling clients to prove their non-malicious participation [70] or having servers jointly return the queries about the verification of users' behavior [28]. These approaches, while feasible, are computationally very expensive in practice.

Secret-shared tracing. Building upon secret-shared shuffling designs, we introduce a companion secret-shared tracing protocol that eliminates the need for public-key cryptography by leveraging the BPC gadget introduced in §V-A. This design significantly reduces the computational overhead typically associated with traceability in prior systems. The core idea is that each row in the BPC matrix uniquely corresponds to the sender of a message. Specifically, tracing is accomplished by partially reconstructing the relevant matrix row and identifying the 1-index within the reconstructed row.

For example, as in Fig. 5, if a message x'_i from the shuffled set $\mathcal{X}'$ is flagged for tracing, the index of 1 entry in the $\mathcal{M}_{(i)}$ directly reveal the identity of its sender. This approach ensures computational efficiency and simplicity while preserving sender anonymity for all other unflagged messages, aligning with the selective traceability objective of *Gyges*. To ensure the correctness and security of the tracing process, we note that each row in the BPC matrix must be well-formed, containing exactly one 1-entry. This property is inherently guaranteed under the semi-honest assumption of the helper party H and is maintained across the honest-majority assumption of the relay parties $\mathcal{P}$. A detailed analysis of security and system properties is provided in §VIII and Appendix C.

Pre-tracing content moderation and post-tracing management. The traceability mechanism must be carefully integrated with complementary pre- and post-tracing designs to prevent the potential misuse of accountability capability.

In this work, we adopt two widely used moderation strategies [17, 32–34] to safeguard honest users against false accusations: 1) threshold-based user reporting and 2) server-side auditing. A traceback request is triggered only after a predefined reporting threshold η is reached, and the tracing process proceeds only when the servers achieve consensus based on the moderation policy set $\mathcal{D}$ (see Fig. 5(a)). This design ensures that merely controlling multiple users or corrupting a single server is insufficient to exploit the accountability mechanism. Given the public nature of messages in (anonymous) broadcast applications, both user reporting and server auditing are practical and straightforward to implement.

Importantly, the tracing functionality will only be invoked for very inappropriate, illegal content (e.g., terrorist propaganda) as defined in $\mathcal{D}$. For less severe infractions, milder measures like message filtering would suffice. For identified misbehaving users, standard responses typically include: 1) blocking future participation [54, 71], or 2) revoking anonymity [72]. While the development of specific pre-tracing policies and post-tracing management strategies lies beyond the scope of this work, they are critical and warrant further investigation.

VI. Private Robustness

We now explore robust shuffling and tracing in the presence of malicious adversaries attempting to disrupt the system. Private robustness [25], or its broader abstraction, MPC with friends and foes [61], ensures output delivery without the need to collaboratively identify or depend on a trusted party capable of learning all secrets and performing computations in the clear on behalf of others. Both privacy and robustness are of great importance for anonymous communication systems.

Rationale behind $[\cdot]$-sharing. The $[\cdot]$-sharing or $(4, 2)$-RSS offers the minimal redundancy required for our private robustness. Each input share has three copies, and given that, at most, one server may be malicious, we can apply a majority rule to identify the correct share. The protocol output follows the $(4, 3)$-RSS structure, where each output share has at least two

Protocol $\Pi_{\mathsf{WCS}}[\mathcal{C},\mathcal{P},H]$: Consistent Sharing

Input: A share $[x_i]_j$ known to $C_i \in \mathcal{C}$.
Output: Weak consistent $[x_i]_j$ known to three servers P_k, where $j,k \in \{1,2,3,4\}$ and $j \neq k$.

Protocol: The protocol proceeds as follows,

1) **Share commitment:**
   - C_i sends a commit $\delta = \mathsf{Hash}([x_i]_j)$ to H and sends $[x_i]_{jk}$ to P_k, where $[x_i]_{jk}$ denotes the copy of $[x_i]_j$ held by P_k.
   - $\mathcal{P}$ and H set a complain counter $\mathsf{flag}_c = 0$.
2) **Weak share consistency:** For all $j,k \in \{1,2,3,4\}$ and $j \neq k$; P_k computes $\delta_k = \mathsf{Hash}([x_i]_{jk})$. If $\delta_k \neq \delta$, P_k complains and increases flag_c by 1.
   - If $\mathsf{flag}_c \geq 2$, identify C_i as corrupted and set x_i as dummy default value.
   - If $\mathsf{flag}_c = 1$, set $[x_i]_{jk} = [x_i]_{jl}$ requested from a non-complaining server P_l.

Fig. 7. Weak consistent user message sharing protocol.

Protocol $\Pi_{\mathsf{RMC}}[\mathcal{C},\mathcal{P}]$: Robust Recovering

Input: $[\![\mathcal{X}']\!]$-shared shuffled messages.
Output: $\mathcal{X}'$ known to $\mathcal{C}$.

Protocol: The protocol works by letting a client C_i pulling all $[\![\mathcal{X}']\!]$ from $\mathcal{P}$,

1) **Consistent reconstruction:** If all shares are consistent, recover $\mathcal{X}'$.
2) **Inconsistent reconstruction:**
   - If subterm shares with three copies are inconsistent, recover $\mathcal{X}'$ according to majority rule.
   - If cross-subterm shares with two copies are inconsistent, recover both $\mathcal{X}'$ and $\mathcal{X}''$ according to two share copies, respectively.

Fig. 8. Privately robust output message reconstruction protocol.

copies. This feature essentially facilitates the robust shuffling and the robust reconstruction. In comparison, using $(3,2)$-RSS inputs in our silent execution paradigm would result in a non-replicated $(3,3)$ additive output share, while larger (n,t)-RSS schemes would introduce significant overhead [43].

From robust computation towards robust reconstruction. Notably, our silent protocol execution is non-interactive, and this indeed equally defers any potential online local deviation to the reconstruction phase. As detailed in §V-C, the reconstruction operates upon $(4,2)$-RSS and/or $(4,3)$-RSS message shares. They both offer somewhat error correction capabilities under the Q_2 adversary structure [73]. This eliminates the need – and associated costs – of identifying a non-malicious party, a common requirement by traditional robustness [24], yet also guarantees the correct output delivery for benign users.

Protocol $\Pi_{\mathsf{Blame}}[\mathcal{P},\mathcal{C}]$: Party Blaming

Input: $[\mathcal{X}]$, $[\![\mathcal{X}]\!]$ or $[\mathcal{X}']$, $[\![\mathcal{X}']\!]$-shared messages.
Output: Possible corrupted party set $\mathcal{P}_c$, honest party set $\mathcal{P}_h$, and corrupted client set $\mathcal{C}_c$.

Protocol: The protocol proceeds as below,

1) **Conflicting group initialization:** Set $\mathcal{P}_c = \mathcal{P}$ and $\mathcal{P}_h = \emptyset$, set $\mathcal{C}_c = \emptyset$.
2) **Corrupted entity set updating:**
   - Once an a party P_i complains and blames another party P_j, set a conflicting group $\mathcal{G} = \{P_i, P_j\}$, set $\mathcal{P}_c = \mathcal{P}_c \cap \mathcal{G}$ and honest party set $\mathcal{P}_h = \mathcal{P} - \mathcal{P}_c$.
   - Once a party P_i blames a client C_k and $P_i \in \mathcal{P}_h$, set $\mathcal{C}_c = \mathcal{C}_c \cup \{C_k\}$.

Fig. 9. Blaming game and conflicting group reduction protocol.

A. Enforcing Input Well-Formedness

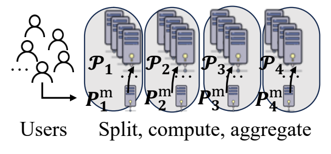
To safeguard the system from malicious users, it is essential to ensure that input message shares adhere to the $(4,2)$-RSS format. For example, if a corrupted client C_c sends a malformed share $[x_i]_4$ to $P_{\{1,2,3\}}$, a corrupted server could further exploit this inconsistency to manipulate the shares, potentially compromising the integrity of all other messages.

Weak secret sharing. To address this issue without resorting to expensive verifiable secret sharing (VSS) [74] or Byzantine reliable broadcast (BRC) [75] primitives like prior works, we assume a public mailbox address managed by the helper party H and tailor on the weak secret sharing (WSS) model by Rabin and Ben-Or [76], which additionally grants honest servers the capability to modify (potentially inconsistent) shares and ensure minimal consistency at relatively low costs.

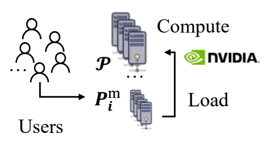
User commitment. When a user sends his/her message shares to servers $\mathcal{P}$, he/she also posts a collision-resistant hash (or a sharing-friendly Carter-Wegman MAC) to a public mailbox (maintained by H) as the commitment of share. Servers verify the integrity of received shares via these commitments, raising a complaint if inconsistencies are detected, as shown in Fig. 7. Our approach rests on two critical observations: 1) honest servers will never issue false complaints against honest users; and 2) message integrity need not be guaranteed for malicious users sending malformed shares. Rather than pinpointing the malicious entity and excluding them, WSS allows us to achieve a consensus by adjusting inconsistent shares to a consistent, dummy value, ensuring no malicious client can disrupt the protocol and no malicious server can impede honest client participation. For BPC, its well-formedness is secured by H and maintained across $\mathcal{P}$ under the majority rule.

B. Robust Reconstruction and Blame Game

We then extend our design to incorporate robust reconstruction capabilities for $[\![\mathcal{X}]\!]$-shared shuffled messages, ensuring G.O.D even in the presence of malicious server disruptions. The key insight is to avoid the explicit identification of exact



(a) Scaling by adding more servers



(b) Scaling via GPU devices

Fig. 10. System scaling of basic 4PS/4PT protocols in both horizontal and vertical directions for better system throughput and performance.

cross subterms prior to reconstruction. Instead, Π_{RMC} detailed in Fig. 8 will generate two potential shuffled message batches, denoted as $\mathcal{X}'$ and $\mathcal{X}''$, whenever inconsistencies are detected. Among these two batches, one – typically the one containing meaningful content – is guaranteed to be valid and correct. This streamlined mechanism efficiently addresses inconsistencies arising from malicious deviations while maintaining full anonymity guarantees for honest participants.

Blaming and conflict group updating. As shown in Fig. 9, any inconsistencies detected during earlier steps lead to the formation of a conflict group. Two critical observations will guide this process: 1) at least one entity in a conflict group is malicious, and 2) honest parties never blame honest users. These properties enable the iterative reduction of group size via set intersections, ultimately isolating the malicious party. This approach is analogous to the *player elimination* technique [77] employed in Shamir sharing but requires neither interpolation nor recomputation. Detailed rules for this process are shown in Appendix B (see Fig. 15).

Computing with a known cheater. Recent work by Bruggemann *et al.* [78] highlights MPC systems can retain a known cheater for better performance. Based on this insight, we adopt a similar technique: once conflicting group $\mathcal{P}_c$ is reduced to a single entity, the system can then directly replace malformed messages with consistent dummy values. This transformation effectively neutralizes the impact of the malicious contribution, treating the dummy message as somewhat cover traffic against external network observers rather than directly excluding it.

VII. System Scaling

Aligned with numerous prior studies [8, 9, 20, 23, 31, 50], we anticipate that our secret-shared accountable anonymous broadcast techniques can scale effectively, thereby enhancing mixing service throughput to better accommodate a broader user base with stronger anonymity.

Viability of MPC-style scaling. Conventionally, MPC deployers may often consider scaling their basic small-party solutions to n-party variants for better deployment flexibility. By using generalized (n, t)-RSS techniques, as outlined by Baccarini *et al.* [43] and Dalskov *et al.* [56], our four-party shuffling and tracing schemes could theoretically expand to the n-party setting. However, this also incurs proportional increases in total share size and computational overhead, offering no tangible server throughput improvements. Instead, we adopt two alternative scaling strategies to enhance protocol execution, ensuring that the number of messages processed per party grows polynomially with the addition of servers (horizontal scaling) or enhancements in per-server computational capacity (vertical scaling), as illustrated in Fig. 11.

Protocol $\Pi_{\mathsf{Scale}}[\mathcal{P}]$: Protocol Scaling

Input: The same as protocol Π_{4PS} and Π_{4PT}.
Output: The same as protocol Π_{4PS} and Π_{4PT}.

Protocol: Let each party $P_i \in \mathcal{P}$ hold a management server and multiple computing servers, denoted as P_i^{m} and P_{ih}. The GPU server held by party P_i is denoted as P_i^{g}. Then the protocol proceeds as follows,

1) **Horizontal scaling:** P_i^{m} coordinates and distributes local computation to different P_{ih} servers.
2) **Vertical scaling:** P_i^{m} coordinates and loads local computation to P_i^{g} server.
3) **Results aggregation:** All computing devices (i.e., P_{ih} and P_i^{g}) return locally computed results to P_i^{m} for aggregation and reconstruction.

Fig. 11. Scaling execution of core four-party shuffling and tracing protocols.

Horizontal scaling. Inspired by recent high-throughput mixnet designs such as Atom [23], XRD [9], and Stadium [79], which employ parallelized small mix chains, we take a similar approach in our relay group. Specifically, one relay party manages multiple computing servers, with tasks distributed and performed across these smaller, parallelized units. Our silent protocols are well-suited to horizontal scaling, as the online phases are dominated by computational workload rather than communication or interaction. Adding more servers per party reduces the computational load per server while preserving the appealing small-party structure that is widely regarded as more practical for real-world MPC deployment [24, 80]. Notably, all connections between P_i^{m} and Pih operate exclusively over intra-network channels, incurring less internal overhead.

Vertical scaling. Building on techniques from Blinder [20] and Talek [50], our protocol also benefits from recent advancements in GPU-accelerated cryptography for MPC [81, 82]. As shown in Fig. 10(b), the silent, parallel-friendly nature of our protocols allows for efficient GPU deployment, significantly speeding up protocol execution. To accommodate modern GPU architectures, we further chunk large messages (e.g., $\ell = 1$ KB) into smaller pieces (e.g., $\ell' = 64$ bits), as described in §V-B, enabling more efficient GPU processing.

Somewhat network-level robustness. One primary limitation of our basic four-party protocols is the inbuilt rigid structure, which makes it less resilient to network-level attacks and poor in service flexibility. While important, this issue is generally regarded as orthogonal in theoretical MPC studies. Horizontal scaling within the small-party framework offers a practical solution. By harnessing multiple servers per party, horizontal scaling enhances somewhat robustness against network-level DoS attacks. Specifically, the system remains operational as long as at least one server per party stays online, obviating the

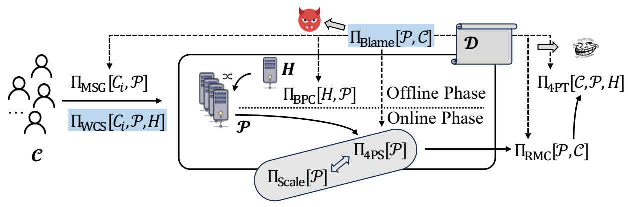

$\Pi_{\text{Blame}}[\mathcal{P},\mathcal{C}]$
$\mathcal{D}$
$\Pi_{\text{MSG}}[C_i,\mathcal{P}]$
$\Pi_{\text{WCS}}[C_i,\mathcal{P},H]$
$\mathcal{C}$
H
$\mathcal{P}$
$\Pi_{\text{BPC}}[H,\mathcal{P}]$
Offline Phase
Online Phase
$\Pi_{\text{4PT}}[\mathcal{C},\mathcal{P},H]$
$\Pi_{\text{4PS}}[\mathcal{P}]$
$\Pi_{\text{Scale}}[\mathcal{P}]$
$\Pi_{\text{RMC}}[\mathcal{P},\mathcal{C}]$

Fig. 12. Overview of *Gyges*'s full protocol stack. A solid line means that the previous sub-protocol's output is the subsequent sub-protocol's input, and a dotted line means that Π_{Blame} participates in all sub-protocols it points to. Shadowed protocols are only invoked in the malicious setting.

need for all servers to be continuously active. This capability is particularly advantageous in scenarios involving server churn, such as temporary server downtime for maintenance, ensuring greater system reliability without compromising efficiency.

VIII. Application and Analysis

Integrating all components, we present *Gyges*, a lightweight, secret-shared accountable anonymous broadcast system. The complete protocol stack is illustrated in Fig. 12. In *Gyges*, users are responsible for submitting message shares, retrieving shuffled messages, and reporting abusive content when necessary. Relay parties collaborate to play several logical roles: 1) secure shuffler, 2) content distributor, 3) content moderator, 4) abuse tracker, 5) misbehaving client handler, and 6) scalable resource allocator. These roles must be carefully assigned to separate trust domains to safeguard sender anonymity and prevent misuse of accountability functionality.

Communication metadata protection, intersection attacks, and Sybil attacks. *Gyges* inherently protects communication metadata within a given broadcast round by hiding the pattern of messages and shuffling them in synchronous batches. To further reduce censorship risks, users are encouraged to participate in multiple broadcast rounds, either by sending meaningful messages or dummy cover traffic. This strategy helps prevent adversaries from executing intersection attacks [31, 83]. Regarding Sybil attacks [84], where the adversary can control numerous nodes to shrink the anonymity set, this remains a significant challenge for anonymous communication systems. To counteract this, we recommend integrating other orthogonal defenses, such as CAPTCHAs [85] or reputation-based access control mechanisms [86], to limit adversaries' ability to flood the network with fake identities effectively.

Realistic applications: anonymous microblogging and message exchanging. One primary application of *Gyges* is anonymous microblogging platforms, similar to Yik Yak [16], where users can publish messages anonymously while concealing their identities. Additionally, *Gyges* supports two-way anonymous message exchange applications.[3] Consider two clients, C_i and C_j, communicating through synchronous message exchanging. This process can be abstracted as writing to and reading from their own virtual addresses. Suppose C_i and C_j have pre-shared a symmetric encryption key k (e.g., $\mathsf{Enc}_k(x_i)$ and $\mathsf{Enc}_k(x_j)$) and their virtual addresses through an out-of-band dialing protocol, as in prior works [13, 21]. After shuffling, C_i and C_j can retrieve and decrypt messages from their respective addresses in $\mathcal{X}'$, completing the exchange.

[3]Messaging services typically involve two phases: 1) dialing to establish the communication pair and 2) message sending and fetching to exchange messages. In this work, we mainly focus on the latter process.

Security analysis and proof sketch. We say *Gyges* is secure if no one can learn private $\mathcal{X}$ and $\mathcal{M}$ during protocol executions (see §IV). We now clarify how *Gyges* achieves its main goals.

Theorem 1: Gyges correctly and securely realizes a privately robust, accountable anonymous broadcast functionality in the presence of an active PPT adversary $\mathcal{A}$ defined in §IV-B.

Proof Sketch: The correctness of *Gyges*'s core protocols has been outlined in prior sections. Their security and output guarantees can be examined below (detailed in Appendix C).

- *Case 1:* $\mathcal{C}_c = \emptyset$ *and* $\mathcal{P}_c = \emptyset$. Here, all relay servers only receive information-theoretic shares instead of the input message batch and BPC matrix themselves, as discussed in §V. Thus, the privacy of *Gyges*'s core shuffling and tracing protocols can always be preserved.
- *Case 2:* $\mathcal{C}_c \neq \emptyset$ *or* $\mathcal{P}_c \neq \emptyset$*, or both.* The well-formedness of inputs in the presence of malicious clients $\mathcal{C}_c$ and/or server P_c can be enforced as in §VI (via weak consistent sharing and robust reconstruction). Following this, the analysis reduces to *Case 1*, and the message integrity for any honest client can always be preserved.

In both aforementioned cases, *Gyges* can ensure the input privacy and the G.O.D of shuffled messages for honest users. Given that $\mathcal{M}$ is private to all participants (including H). The probability that $\mathcal{A}$ can successfully guess whether a shuffled message x'_j originated from C_i indistinguishably approximate $\frac{1}{|\mathcal{X}'|}$. Meanwhile, *Gyges* enables relay servers to deterministically track down a (or a set of) misbehaving user(s) by opening certain $\mathcal{M}_{(a)}$ under established moderation policy $\mathcal{D}$.

IX. Evaluation

System parameters. We built a prototype in Python/C++ and evaluated on machines running Ubuntu 20.04 LTS, equipped with Intel i7-9700K processors, 32 GB of RAM, and NVIDIA RTX 3080 GPUs. For the vertical scaling, we integrated the Piranha framework [82], particularly its P-FantasticFour layer and 64-bit integer kernel. To simulate the horizontal scaling, additional servers were modeled by proportionally reducing workloads across the server per party. Synthetic strings were used as test data, consistent with prior designs in this domain, which can represent realistic scenarios such as microblogging on X/Twitter and transactions in Zcash. The performance of our core techniques was benchmarked against other systems operating in the comparable small-party setting [11, 21, 28]. Specifically, our evaluation spanned a wide range of system parameters, including message batch size N (from 10^4 to 10^6), message length ℓ (from 8 B to 1 KB), the number of computing servers per party (from 1 to 4), and the experiment results are mainly obtained under two different network settings: 1) a LAN environment with a round-trip time (RTT) of 0.4 ms and bandwidth of 1 Gbps; and 2) a WAN environment with an RTT of 120 ms and bandwidth of 100 - 200 Mbps.

Preprocessing performance. To generate a fresh, valid BPC, relay servers should evaluate N independent four-party one-bit DPF key tuples, which results in total offline communication complexity of $O(N^{1.5})$. As demonstrated in Table II, *Gyges* generally reports lower communication overhead compared to prior systems Clarion [11] and RPM [21]. A notable advantage of *Gyges* lies in its Boolean representation of target correlations, which ensures offline communication scales linearly with the batch N only and remains independent of the message length ℓ. This independence makes *Gyges* particularly well-suited for scenarios involving very large messages, overcoming a critical limitation of prior approaches where communication scales proportionally with both parameters. For instance, RPM [21] (variant-I) adopts a similar random permutation matrix representation in Shamir's sharing. However, it requires significant overhead due to extensive matrix multiplications, leading to $O(N^2\ell)$ communication and $O(N^3)$ computational complexity per shuffle in its offline phase. These inefficiencies become particularly pronounced in large-scale deployments.

Importantly, both the shuffle correlation (i.e., BPC column) and the trace correlation (i.e., BPC row) are captured simultaneously within the preprocessing cost, as shown in Table II. No additional overhead is required to enable traceability.

Silent shuffling performance. In contrast to prior anonymous broadcast systems, such as RPM [21] and Clarion [11], the silent execution of *Gyges* significantly reduces online communication and interaction complexity. Specifically, Clarion requires two rounds for secret-shared shuffling and an additional four rounds for verification in malicious settings, resulting in a total of six rounds per shuffle. RPM's variant-I, although more efficient in round complexity, incurs substantially higher computational costs and communication overhead in both the offline and online phases.

As summarized in Table III, *Gyges* outperforms Clarion by $3.4\times$ and the three RPM variants by $9.6\times$, $3.7\times$, and $3.6\times$ in shuffle protocol execution time. This performance advantage is particularly pronounced for relatively smaller batch sizes (e.g., $N \leq 10^4$), where *Gyges*'s reduced communication overhead plays a pivotal role, making it especially suited for bandwidth-sensitive scenarios. The minimal interaction nature of the silent protocol enables faster execution compared to all other competitors, effectively leveraging its streamlined communication model. For larger batch sizes at the $N \geq 10^6$ scale, however, *Gyges* encounters challenges due to memory limitations and increased computational complexity, which lead to higher latency in the online shuffling phase. These challenges can be addressed through horizontal and vertical scaling strategies, as detailed in Table V and Table VI, enabling *Gyges* to maintain its performance edge even in large-scale deployments.

Secret-shared tracing performance. Compared to traditional verifiable shuffle-and-trace solutions relying on public-key cryptography, such as state-of-the-art traceable mixnets [28], *Gyges*'s secret-shared tracing mechanism offers substantial improvements in both simplicity and efficiency. As highlighted in Table IV, our lightweight tracing protocol delivers several orders of magnitude performance improvement in comparable settings. The efficiency of tracing remains nearly independent of the total message batch size N and message length ℓ, as the search operates directly on the Boolean one-hot vector.

In typical scenarios, a tracing request can be completed in less than one second. Notably, the security assumptions underlying the tracing for accountability are aligned with those of shuffling for anonymity, both relying on a distributed trust model across multiple non-colluding entities.

Private robustness overhead. The results in Table III already account for the amortized costs of private robustness against malicious adversaries. Specifically, *Gyges*'s robustness overhead arises from two components: 1) weak consistent sharing, and 2) private robust reconstruction. These costs can be further amortized over the execution of the blame game and the re-

TABLE II
COMPARISON OF OFFLINE PREPROCESSING PERFORMANCE.

Ref.	N	ℓ	Clarion [11]	RPM-I [21][†]	*Gyges*[‡]
Comm.[MB]	10^3	8 B	0.198	16	0.045
		32 B	0.258	64	
		160 B	0.575	320	
	10^4	8 B	1.983	1600	1.431
		32 B	2.578	6400	
		160 B	5.751	32000	

[†] RPM [21] introduces three different variants. This table picks RPM-I for comparison, as its computation paradigm is the closest to *Gyges*. RPM-III reports a relatively better offline communication cost of 240 MB for shuffling 10^4 messages of 8 B length at the cost of more online overhead. *Gyges* still outperforms it regarding communication overhead and computation latency.

[‡] *Gyges* and Clarion [11] both adopt a semi-honest helper party H to prepare the predecessor for constructing their shuffle correlations. Note it is slightly different from the traditional dealer model, as H doesn't learn the final correlation directly. In our case and Clarion, such a party also helps enforce the well-formedness of shuffle correlation for better system availability.

TABLE III
COMPARISON OF ONLINE SHUFFLING PROCESS PERFORMANCE.

Ref.[†, ‡, ♮]		N	Clarion [11]	RPM-I, II, III [21]			*Gyges*
Shuffle	Comm. [MB]	10^3	0.228	0.485	0.097	-	0
		10^4	2.289	0.742	0.961	2.763	0
		10^5	22.888	-	-	27.632	0
		10^6	228.881	-	-	-	0
	Time [s]	10^3	0.075	0.051	0.022	-	0.013
		10^4	0.718	1.485	0.581	0.556	0.155
		10^5	7.629	-	-	13.681	1.774
		10^6	95.089	-	-	-	27.945
Trace	Time [s]	10^5	-	-	-	-	0.035

[†] This table shows the results in LAN settings where $\ell = 8$ B.

[‡] As estimated by Table II, RPM-I/II requires over 320 GB of offline communication to mix 10^5 8 B messages, making it less practical than others. Here, we mark its online performance as unavailable accordingly, denoted as "–". Also, since no prior works provide traceability mechanisms, we mark them as unavailable as well.

[♮] As a fair comparison, this table only reflects the costs of shuffling protocol execution (excluding the message sharing and reconstruction phases). In Fig 13, we further report the end-to-end performance in the WAN setting.

TABLE IV
COMPARISON OF ONLINE TRACING PROCESS PERFORMANCE.

Ref.[†]			Traceable mixnets [28]	*Gyges*
Shuffle	$n = 4$ $N = 10^4$	Time [s]	343	0.155
Trace		Comm. [MB]	4.1	0.005
		Time [s]	681.6	0.134

[†] For a fair comparison, here we report the tracing latency, which measures the time from initiating a tracing request to identifying the abusive senders in a WAN setting. This metric excludes the costs associated with pre- and post-tracing actions, such as content moderation enforcement and management of misbehaving users.

TABLE V
EFFICIENCY OF *Gyges* UNDER DIFFERENT DOMAIN-SPECIFIC COMPUTATIONAL OPTIMIZATIONS.

Ref.†, ‡	Env.	w/o Sparsity	w/ Sparsity
Time [s]	*Gyges*-CPU	70.144	27.945
	Gyges-GPU	19.813	-

† GPU architectures do not natively support operations on elements with differing bit lengths (e.g., between the 1-bit and 64-bit operands). As a result, we simply lift the BPC element to 64-bit, thus foregoing the sparsity-aware optimizations so far.
‡ To support an arbitrary large size of messages in the GPU variant, we combine the message chunking mechanism (as discussed in §V-B) with silent shuffling techniques.

TABLE VI
THROUGHPUT OF *Gyges*'S DIFFERENT SCALING VARIANTS.

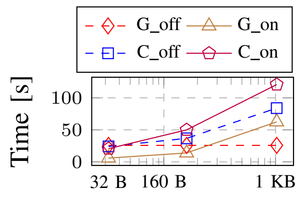
Ref.†	Env.	# servers per party		
		1	2	3
Throughput $[10^7$ entry/min]	*Gyges*-CPU (w/ sparsity)	1.266	2.531	3.797
	Gyges-GPU (w/o sparsity)	1.546	3.092	4.639

† The results are under the LAN setting for mixing 10^5 8 B messages. Inter-party communication and interaction remain optimal.

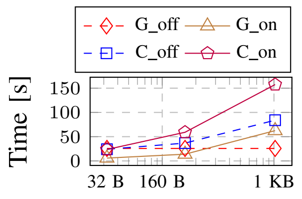
duction of conflicting groups for long-term broadcast services (see Appendix B). In *Gyges*, misbehaving clients are classified into two categories: 1) those sending malformed shares (i.e., malicious users), and 2) those submitting inappropriate content (i.e., abusive users). The former is captured by our private robustness technique. For the latter, tracing evaluation results are provided in Table IV.

Domain-specific optimizations and scaling designs. *Gyges*'s sparsity-aware optimization improves the protocol execution by eliminating the physical execution of multiplication and any computations involving 0. This proves particularly effective in the case that message sizes are very large, such as $\ell = 1$ KB. As shown in Table V, when shuffling 10^5 messages of 8 B length, *Gyges* with (w/) outperforms without (w/o) sparsity-aware optimization by $2.4\times$ on average regarding the online time. Moreover, *Gyges* leverages vertical scaling through GPU acceleration, achieving up to a $3.5\times$ performance improvement over the CPU-based implementation. Additionally, as shown in Table VI, horizontal scaling design allows *Gyges* to improve throughput by increasing the number of servers per party, further boosting its practical performance.

According to Table VI, we estimate that *Gyges* can mix between 4.22×10^5 and 1.54×10^6 messages of 30×8 B size per minute, while a real-world platform like Twitter/X generates approximately 3.5×10^5 tweets (up to 240 B in length) per minute in total [69]. Although these results were obtained in a LAN environment, we note the interaction/communication complexity in *Gyges* is optimal and does not pose a bottleneck. In this sense, it is fair to assume *Gyges* has the potential to scale effectively and serve real-world applications.



(a) Anonymous microblogging



(b) Anonymous message exchanging

Fig. 13. Comparison of execution time for 1) one-way microblogging; and 2) two-way message exchanging services over *Gyges* and Clarion over the WAN setting. $N = 10^5$, $\ell \in \{32$ B, 160 B, 1 KB$\}$. G_off/G_on and C_off/C_on mean the offline and online execution time of *Gyges* and Clarion, respectively.

End-to-end applications. In all cases, users $\mathcal{C}$ are responsible for encoding and submitting their $[\mathcal{X}]$-shared messages to relay parties $\mathcal{P}$. The meaningful messages sent by real users will be anonymized, while the dummy messages sent by volunteers will play as somewhat cover traffic.

As depicted in Fig. 13, in the typical WAN setting, *Gyges* can anonymously broadcast 10^5 microblog messages of 1 KB length each in an end-to-end latency of 88.19 seconds, which outperforms Clarion [11] by $2.32\times$. Notably, the offline costs for *Gyges* remain constant regardless of message size ℓ, providing significant efficiency benefits, especially when ℓ is very large. In the (two-way) message-exchanging service, users can obtain target addresses and notify their peers via out-of-band communication beforehand, enabling efficient message retrieval without linear scans, which results in only slightly higher costs than the (one-way) broadcast scenario.

Discussions. Due to the minimal online communication (and interaction) complexity, ℓ-independent offline costs, sparsity-aware optimizations, and parallel-friendly features, *Gyges* can perform well when 1) ℓ is large, 2) network latency is high, 3) inter-server bandwidth is limited. Although optimizations in *Gyges* are initially tailored for anonymous broadcasts, the communication-efficient BPC and silent shuffling may be of other independent interest in MPC settings with no honest majority [11], thus eliminating inherent inefficiencies rooted in RSS schemes. Also, for other situations where the consistency of input and output sharing semantics is not necessary, one may optimize protocols by saving one round of interaction.

X. LIMITATIONS AND ETHICAL CONSIDERATIONS

We now discuss limitations regarding system assumptions, our countermeasures and trade-offs, and ethical considerations surrounding accountability in *Gyges*. This highlights potential areas for refinement and provides a broader context for understanding responsible use of our design.

Honest-majority setup. In-protocol robustness is crucial for (anonymous) communication systems, ensuring that the service can withstand disruptions and guarantee message delivery. To achieve this beyond security with abort [11], we adopt an honest-majority assumption – as justified by Cleve [87], any robust MPC protocol (including *Gyges*) must operate under this assumption. Building on this, we further customize new, lightweight private robustness techniques, ensuring message delivery without sacrificing sender anonymity.

Small-party setup. Compared to systems relying on Shamir sharing [20–22], *Gyges* follows a more rigid adversarial structure. However, MPC deployers show that small-party setups are often more practical and easier to correctly instantiate within distributed trust domains [80]. As a countermeasure, we explore the possibility of scaling *Gyges* in a traditional MPC sense via generic RSS techniques (see §VII) and examine other scaling notions for better throughput. While MPC protocols, especially in small-party settings, are typically less focused on mitigating essential but orthogonal network-level attacks, *Gyges*'s horizontal scaling approach offers some flexibility in this regard, enabling server churn and enhancing robustness against certain network-level disruptions.

Computational bottleneck and countermeasures. As shown in Table I, *Gyges* reports a higher computational complexity. However, we note our protocol execution requires information-theoretical additions only. By Amdahl's law and Gustafson's law [88], such parallel-friendly protocols allow scaling *Gyges* up and out easily and can concretely accelerate their executions. In addition, our silent protocol technique minimizes the inter-server interaction, making the system scale well in terms of N and ℓ from the communication perspective.

Ethical considerations. The enduring tension between user privacy and network governance has long-defined debates over the Internet ethics. An illustrative case is Yik Yak [16], a once-popular anonymous online community that ultimately faced significant backlash due to pervasive cyberbullying and abuse. This underscores how unfettered anonymity can undermine the very values it seeks to protect. In this context, we envision *Gyges* as a promising tool for such anonymous online platforms, offering a nuanced balance between anonymity and accountability while maintaining robust availability.

Tracing can identify the source of harmful message content but risks misuse for censorship or privacy infringement. To mitigate this, well-defined pre-tracing policies and post-tracing management are essential to safeguard against the abuse of accountability mechanisms. A conventional human moderation pipeline can serve as a foundation, with advancements in large language models [89, 90] offering opportunities to automate and enhance moderation efforts. Notably, in *Gyges*, a traceback request will be initiated only after a predefined user-reporting threshold is met, and the process will proceed only upon careful consensus among the servers. This restricts the tracing function to very inappropriate or illegal cases, such as terrorist propaganda or child exploitation, while less critical violations are addressed with milder actions like filtering.

XI. Conclusion

This work introduces *Gyges*, a lightweight secret-shared accountable anonymous broadcast scheme that tackles the long-standing challenge of reconciling anonymity, accountability, performance, robustness, and scalability in secure communication systems. Through innovative silent protocol techniques, *Gyges* achieves optimal communication efficiency, optimizes computational overhead, ensures private robustness, and scales effectively to enhance system throughput. Moreover, it strikes a careful balance between preserving sender anonymity and enabling accountability for misbehaving users within the MPC framework. Looking ahead, several exciting research directions arise: 1) reducing offline costs by eliminating the need for an assisting party and 2) further optimizing computational complexity to improve overall performance.

Acknowledgments

The authors sincerely thank the reviewers for their invaluable feedback. This work was supported in part by Hong Kong Research Grants Council (RGC) under Grants 11217620, 11218521, 11218322, R6021-20F, R1012-21, RFS2122-1S04, C2004-21G, C1029-22G, C6015-23G, N_CityU139/21; Hong Kong Innovation and Technology Commission (ITC) under Scheme MHP/135/23; and National Natural Science Foundation of China (NSFC) under Grants U20B2049, U21B2018, 62302344. This work was also substantially supported by the InnoHK initiative, The Government of the HKSAR, and the Laboratory for AI-Powered Financial Technologies (AIFT).

References

[1] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[2] ——, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[3] S. Bayer and J. Groth, "Efficient zero-knowledge argument for correctness of a shuffle," in *Proc. of EUROCRYPT*, 2012.

[4] T. Haines, R. Goré, and B. Sharma, "Did you mix me? Formally verifying verifiable mix nets in electronic voting," in *Proc. of IEEE S&P*, 2021.

[5] D. Wikström, "A sender verifiable mix-net and a new proof of a shuffle," in *Proc. of ASIACRYPT*, 2005.

[6] D. Chaum, D. Das, F. Javani, A. Kate, A. Krasnova, J. D. Ruiter, and A. T. Sherman, "cMix: Mixing with minimal real-time asymmetric cryptographic operations," in *Proc. of ACNS*, 2017.

[7] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable anonymous messaging," in *Proc. of ACM CCS*, 2010.

[8] S. Langowski, S. Servan-Schreiber, and S. Devadas, "Trellis: Robust and scalable metadata-private anonymous broadcast," in *Proc. of NDSS*, 2023.

[9] A. Kwon, D. Lu, and S. Devadas, "XRD: Scalable messaging system with cryptographic privacy," in *Proc. of USENIX NSDI*, 2020.

[10] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "MCMix: Anonymous messaging via secure multiparty computation," in *Proc. of USENIX Security*, 2017.

[11] S. Eskandarian and D. Boneh, "Clarion: Anonymous communication from multiparty shuffling protocols," in *Proc. of NDSS*, 2022.

[12] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *Proc. of IEEE S&P*, 2015.

[13] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the cost of metadata-hiding communication with cryptographic privacy," in *Proc. of USENIX Security*, 2021.

[14] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Proc. of CRYPTO*, 2015.

[15] L. Mann, “The baiting crowd in episodes of threatened suicide.” *Journal of Personality and Social Psychology*, vol. 41, no. 4, p. 703, 1981.
[16] M. Saveski, S. Chou, and D. Roy, “Tracking the Yak: an empirical study of Yik Yak,” in *Proc. of ICWSM*, 2016.
[17] L. Liu, D. S. Roche, A. Theriault, and A. Yerukhimovich, “Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS),” in *Proc. of NDSS*, 2022.
[18] A. Schlesinger, E. Chandrasekharan, C. A. Masden, A. S. Bruckman, W. K. Edwards, and R. E. Grinter, “Impacts of anonymity, ephemerality, and hyper-locality on social media,” in *Proc. of ACM CHI*, 2017.
[19] W. Ahmad and I. Liccardi, “Addressing anonymous abuses: Measuring the effects of technical mechanisms on reported user behaviors,” in *Proc. of ACM CHI*, 2020.
[20] I. Abraham, B. Pinkas, and A. Yanai, “Blinder – Scalable, robust anonymous committed broadcast,” in *Proc. of ACM CCS*, 2020.
[21] D. Lu and A. Kate, “RPM: Robust anonymity at scale,” in *Proc. of PETs*, 2023.
[22] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, “HoneybadgerMPC and Asynchromix: Practical asynchronous mpc and its application to anonymous communication,” in *Proc. of ACM CCS*, 2019.
[23] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, “Atom: Horizontally scaling strong anonymity,” in *Proc. of ACM SOSP*, 2017.
[24] N. Koti, V. B. Kukkala, A. Patra, B. R. Gopal, S. Sangal *et al.*, “Ruffle: Rapid 3-party shuffle protocols,” in *Proc. of PETs*, 2023.
[25] A. Dalskov, D. Escudero, and M. Keller, “Fantastic Four: Honest-majority four-party secure computation with malicious security,” in *Proc. of USENIX Security*, 2021.
[26] C. Cho, S. Dittmer, Y. Ishai, S. Lu, and R. Ostrovsky, “Rabbit-mix: Robust algebraic anonymous broadcast,” in *Proc. of USENIX Security*, 2024.
[27] R. Cramer, I. Damgård, and Y. Ishai, “Share conversion, pseudorandom secret-sharing and applications to secure computation,” in *Proc. of TCC*, 2005.
[28] P. Agrawal, A. Nakarmi, M. P. Jhawar, S. Sharma, and S. Banerjee, “Traceable mixnets,” in *Proc. of PETs*, 2024.
[29] W. Chen and R. A. Popa, “Metal: A metadata-hiding file-sharing system,” in *Proc. of NDSS*, 2020.
[30] Z. Newman, S. Servan-Schreiber, and S. Devadas, “Spectrum: High-bandwidth anonymous broadcast,” in *Proc. of USENIX NSDI*, 2022.
[31] 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 *Proc. of USENIX NSDI*, 2023.
[32] S. Eskandarian, “Abuse reporting for metadata-hiding communication based on secret sharing,” in *Proc. of Usenix Security*, 2024.
[33] R. Issa, N. Alhaddad, and M. Varia, “Hecate: Abuse reporting in secure messengers with sealed sender,” in *Proc. of USENIX Security*, 2022.
[34] N. Tyagi, I. Miers, and T. Ristenpart, “Traceback for E2EE messaging,” in *Proc. of ACM CCS*, 2019.
[35] J. Bartusek, S. Garg, A. Jain, and G.-V. Policharla, “End-to-end secure messaging with traceability only for illegal content,” in *Proc. of EUROCRYPT*, 2023.
[36] D. Mardi, S. Tanwar, and J. Howlader, “Multiparty protocol that usually shuffles,” *Security and Privacy*, vol. 4, no. 6, p. e176, 2021.
[37] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. K. Thyagarajan, “Foundations of coin mixing services,” in *Proc. of ACM CCS*, 2022.
[38] A. Cheu, A. Smith, J. Ullman, D. Zeber, and M. Zhilyaev, “Distributed differential privacy via shuffling,” in *Proc. of EUROCRYPT*, 2019.
[39] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” in *Proc. of USENIX Security*, 2004.
[40] jrandom (Pseudonym), “Invisible internet project (I2P) project overview,” 2003, https://geti2p.net/en/.
[41] M. Chase, E. Ghosh, and O. Poburinnaya, “Secret-shared shuffle,” in *Proc. of ASIACRYPT*, 2020.
[42] X. Song, D. Yin, J. Bai, C. Dong, and E.-C. Chang, “Secret-shared shuffle with malicious security,” in *Proc. of NDSS*, 2024.
[43] A. Baccarini, M. Blanton, and C. Yuan, “Multi-party replicated secret sharing over a ring with applications to secure machine learning,” in *Proc. of PETs*, 2023.
[44] A. Kwon, D. Lazar, S. Devadas, and B. Ford, “Riffle: An efficient communication system with strong anonymity,” in *Proc. of PETs*, 2016.
[45] C. A. Melchor, J. Barrier, L. Fousse, and M.-O. Killijian, “XPIR: Private information retrieval for everyone,” in *Proc. of PETs*, 2016.
[46] S. Angel and S. T. Setty, “Unobservable communication over fully untrusted infrastructure.” in *Proc. of USENIX OSDI*, 2016.
[47] A. Vadapalli, K. Storrier, and R. Henry, “Sabre: Sender-anonymous messaging with fast audits,” in *Proc. of IEEE S&P*, 2022.
[48] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
[49] D. Das, S. Meiser, E. Mohammadi, and A. Kate, “Divide and funnel: a scaling technique for mix-networks,” in *Proc. of IEEE CSF*, 2024.
[50] 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 *Proc. of IEEE ACSAC*, 2020.
[51] L. Von Ahn, A. Bortz, N. J. Hopper, and K. O’Neill, “Traceable anonymity,” in *Proc. of PETs*, 2006.
[52] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, “Dissent in numbers: Making strong anonymity scale,” in *Proc. of USENIX OSDI*, 2012.
[53] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, “Proac-

tively accountable anonymous messaging in verdict," in *Proc. of USENIX Security*, S. T. King, Ed., 2013.
[54] N. Tyagi, J. Len, I. Miers, and T. Ristenpart, "Orca: Blocklisting in sender-anonymous messaging," in *Proc. of USENIX Security*, 2022.
[55] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPDZ2k: Efficient MPC mod 2k for dishonest majority," in *Proc. of CRYPTO*, 2018.
[56] A. Dalskov, D. Escudero, and A. Nof, "Fast fully secure multi-party computation over any ring with two-thirds honest majority," in *Proc. of ACM CCS*, 2022.
[57] D. Escudero, *Multiparty Computation over Z/2k*. University of Aarhus, 2021.
[58] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness," in *Proc. of TCC*, 2013.
[59] C. Kuhn, M. Beck, S. Schiffner, E. Jorswieck, and T. Strufe, "On privacy notions in anonymous communication," in *Proc. of PETs*, 2019.
[60] S. Sasy and I. Goldberg, "Sok: Metadata-protecting communication systems," in *Proc. of PETs*, 2024.
[61] B. Alon, E. Omri, and A. Paskin-Cherniavsky, "MPC with friends and foes," in *Proc. of CRYPTO*, 2020.
[62] R. Cohen, I. Haitner, E. Omri, and L. Rotem, "Characterization of secure multiparty computation without broadcast," in *Proc. of TCC*, 2016.
[63] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. of CRYPTO*, 1991.
[64] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments," in *Proc. of IEEE S&P*, 2014.
[65] J. Doerner and A. Shelat, "Scaling ORAM for secure computation," in *Proc. of ACM CCS*, 2017.
[66] B. Karmakar, N. Koti, A. Patra, S. Patranabis, P. Paul, and D. Ravi, "Asterisk: Super-fast mpc with a friend," in *Proc. of IEEE S&P*, 2024.
[67] W. Dong and C. Wang, "Poster: Towards lightweight TEE-assisted MPC," in *Proc. of ACM CCS*, 2023.
[68] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. of ACM AsiaCCS*, 2018.
[69] J. Shepherd, "Essential twitter (x) statistics," 2024, https://thesocialshepherd.com/blog/twitter-statistics.
[70] P. Grubbs, A. Arun, Y. Zhang, J. Bonneau, and M. Walfish, "Zero-knowledge middleboxes," in *Proc. of USENIX Security*, 2022.
[71] P. P. Tsang, A. Kapadia, C. Cornelius, and S. W. Smith, "Nymble: Blocking misbehaving users in anonymizing networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, pp. 256–269, 2009.
[72] Y. Emek, J. Seidel, and R. Wattenhofer, "Computability in anonymous networks: Revocable vs. irrecovable outputs," in *Proc. of ICALP*, 2014, pp. 183–195.
[73] R. Jadoul, N. P. Smart, and B. Van Leeuwen, "MPC for Q2 access structures," in *Proc. of SAC*, 2021.
[74] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. of IEEE SFCS*, 1987.
[75] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT," in *Proc. of ACM CCS*, 2016.
[76] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proc. of ACM STOC*, 1989.
[77] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity," in *Proc. of TCC*, 2008.
[78] A. Brüggemann, O. Schick, T. Schneider, A. Suresh, and H. Yalame, "Don't eject the impostor: Fast three-party computation with a known cheater," in *Proc. of IEEE S&P*, 2024.
[79] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proc. of ACM SOSP*, 2017.
[80] D. Kaviani and R. A. Popa, "MPC deployments," 2023, https://mpc.cs.berkeley.edu/.
[81] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the gpu," in *Proc. of IEEE S&P*, 2021.
[82] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *Proc. of USENIX Security*, 2022.
[83] S. Gaballah, T. H. L. Nguyen, L. Abdullah, E. Zimmer, and M. Mühlhäuser, "Mitigating intersection attacks in anonymous microblogging," in *Proc. of ARES*, 2023.
[84] E. Crites, A. Kiayias, and A. Sarencheh, "Syra: Sybil-resilient anonymous signatures with applications to decentralized identity," *IACR ePrint Archive*, 2024.
[85] J. Yan and A. S. El Ahmad, "CAPTCHA security," *IEEE Security & Privacy*, vol. 7, no. 4, pp. 22–28, 2009.
[86] L. Tulloch and I. Goldberg, "Lox: Protecting the social graph in bridge distribution," in *Proc. of PETs.*, 2023.
[87] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proc. of ACM STOC*, 1986.
[88] D. Padua, *Encyclopedia of parallel computing*. Springer Science & Business Media, 2011.
[89] OpenAI, "GPT-4 for content moderation," 2023, https://openai.com/index/using-gpt-4-for-content-moderation/.
[90] M. Kolla, S. Salunkhe, E. Chandrasekharan, and K. Saha, "LLM-Mod: Can large language models assist content moderation?" in *Proc. of ACM CHI*, 2024.

APPENDIX

A. Pseudorandom Message Sharing

We now outline the pseudorandom message-sharing protocol Π_{MSG} between clients and servers (see Fig. 14). Most shares can be locally generated by servers, and for each client and message, only three duplicated $[x_i]_4$ need to be sent to P_1, P_2, P_3, along with the corresponding commitment δ to H. Notably, our protocol does not rely on the broadcast channel. As a result, each client must send 3ℓ bits for the shares and $|\delta|$ bits for the commitment per message. However, if a reliable

Protocol $\Pi_{\mathsf{MSG}}[\mathcal{C},\mathcal{P}]$: Message Sharing

Input: Message $x_i \in \mathbb{Z}_{2^\ell}$ known to $C_i \in \mathcal{C}$.
Output: $[x_i]$-shared user message known to $\mathcal{P}$.

Protocol: the protocol runs as follows,

1) **PRF setup:** C_i samples seeds $s_{ij} \leftarrow 1^\lambda$, sends s_{ij} to P_k where $j \in \{1,2,3\}$, $k \in \{1,2,3,4\}$, and $j \neq k$.
2) **PRSS execution:** Both C_i and $P_{j\in\{1,2,3\}}$ compute $[x_i]_j = \mathsf{PRF}_j(s_j)$ locally. C_i computes $[x_i]_4 = x_i - \sum_{i=1}^{i=3}[x_i]_j$ and sends it to $P_{j\in\{1,2,3\}}$.

Fig. 14. Pseudorandom user message sharing protocol.

broadcast channel is available, *Gyges* can still benefit from it – the client-server communication cost can further be reduced to an optimal ℓ per message, on an amortized basis.

B. Blaming Game

The blaming game operates via two core functions: 1) server blaming and 2) reduction. Within a static adversary structure, this process progressively narrows down $\mathcal{P}_c$, ultimately isolating the malicious one. The procedure is executed during two main processes: 1) server-client interactions (as in Fig.15(a) - 15(d)); 2) server-server interactions (as in Fig.15(e) - 15(h)). For instance, if P_i detects and accuses client C_j of sending inconsistent shares, C_j, P_i forms a conflict pair. Similarly, if server P_i accuses server P_k of inconsistent sharing, P_i, P_k is identified as a conflicting group, with at least one entity assumed honest. Over time, the honest parties in $\mathcal{P}$ can reliably identify any malicious participant, whether server P_c or client C_c, based on repeated protocol deviations. The computational and communication overhead introduced by the accusation and reduction processes is amortized over the overall protocol execution, ensuring minimal performance impact.

C. System Analysis

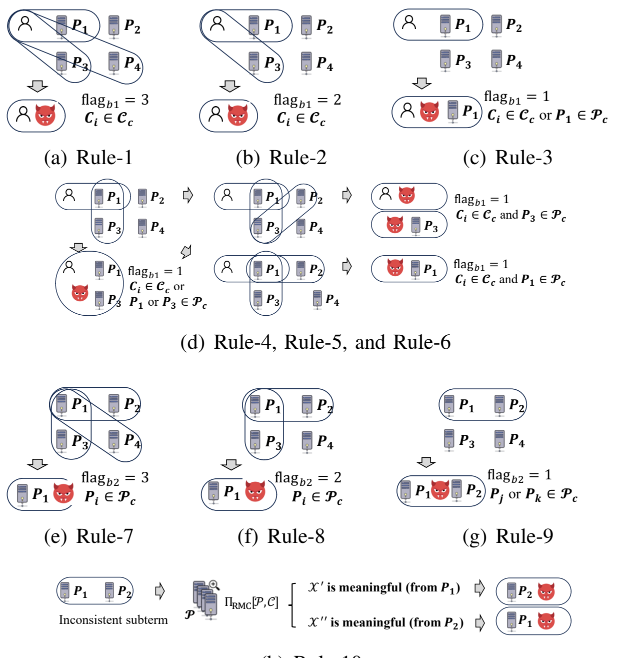
In this section, we extend the arguments in Theorem 1 to justify the security and design goals of our proposed protocols. In the next, we consider all relay parties to be symmetric and take P_1 as an example. The cases for $\{P_2, P_3, P_4\}$ are similar.

❶ **Message sharing phase.** We first consider protocol Π_{MSG} and Π_{WCS} across $\mathcal{C}$, $\mathcal{P}$, and H.

Theorem 2: Assuming the availability of cryptographically secure pseudorandom functions and collision-resistant hash functions, protocol Π_{MSG} and Π_{WCS} securely realize the weak consistent message sharing functionality, even in the presence of a potentially malicious client and/or one corrupted party.

PROOF: In protocols Π_{MSG} and Π_{WCS}, the four participating parties consist of honest and potentially corrupted individuals, with the client C_i either being honest or adversarial. We analyze the security under various cases:

- *Case 1: Both P_1 and C_i are semi-honest.* The protocol can operate securely and privately in this setting, consistent with traditional semi-honest MPC security. Since P_1 receives only information-theoretic shares of the input x and the commitments, it learns nothing about the plaintext message. The views in the ideal world and the real world remain computationally indistinguishable.



flag$_{b1}$ = 3 $C_i \in \mathcal{C}_c$
flag$_{b1}$ = 2 $C_i \in \mathcal{C}_c$
flag$_{b1}$ = 1 $C_i \in \mathcal{C}_c$ or $P_1 \in \mathcal{P}_c$
(a) Rule-1
(b) Rule-2
(c) Rule-3
flag$_{b1}$ = 1 $C_i \in \mathcal{C}_c$ and $P_3 \in \mathcal{P}_c$
flag$_{b1}$ = 1 $C_i \in \mathcal{C}_c$ or P_1 or $P_3 \in \mathcal{P}_c$
flag$_{b1}$ = 1 $C_i \in \mathcal{C}_c$ and $P_1 \in \mathcal{P}_c$
(d) Rule-4, Rule-5, and Rule-6
flag$_{b2}$ = 3 $P_i \in \mathcal{P}_c$
flag$_{b2}$ = 2 $P_i \in \mathcal{P}_c$
flag$_{b2}$ = 1 P_j or $P_k \in \mathcal{P}_c$
(e) Rule-7
(f) Rule-8
(g) Rule-9
Inconsistent subterm
$\Pi_{\mathsf{RMC}}[\mathcal{P},\mathcal{C}]$
X' is meaningful (from P_1)
X'' is meaningful (from P_2)
(h) Rule-10

Fig. 15. *Gyges*'s blaming game execution and conflict reduction rules.

- *Case 2: P_1 is malicious and C_i is semi-honest.* This case aligns with traditional malicious security in MPC. If P_1 falsely accuses C_i of sending inconsistent shares $[x_i]$ that don't match its commitment $H([x_i])$, P_1 can still acquire the correct share from other honest parties, ensuring the well-formedness of the $(4,2)$-RSS sharing scheme. Even in the false accusation, P_1 gains no knowledge of x. If P_1 persists in blaming others, the blame and reduction mechanism will iteratively reduce the conflicting group and reveal the malicious server (see Appendix B).
- *Case 3: P_1 is semi-honest and C_i is malicious.* This is common in the traditional verifiable secret-sharing context. For our cases, P_1 can always correctly blame the inconsistent $[x_i]$ sent by C_i and ensure the well-formedness via Π_{WCS}, during which P_1 learns nothing about x_i, so the joint distribution of $\mathcal{P}$'s view and messaging sharing functionality's outputs should be indistinguishable to the adversary $\mathcal{A}$'s view and the Π_{MSG}.
- *Case 4: P_1 is malicious and C_i is malicious.* Even if P_1 and C_i collude to pass off malformed, inconsistent message shares as valid, if C_i sends inconsistent shares to the remaining honest servers, they will correctly identify C_i as malicious. Otherwise, the remaining honest parties will reach a consensus on a dummy, consistent message to substitute the malformed message via Π_{WCS}, or identify the malicious entity via Π_{Blame}. During this process, P_1 gains no other information about x_i, and $\mathcal{A}$'s view is indistinguishable from the ideal world.

Across all cases, we ensure that an adversary can neither compromise the privacy, nor violate the message integrity of the messages of honest users under the majority rule.

❷ **Offline preprocessing phase.** We then examine protocol Π_{BPC} between $\mathcal{P}$ and H.

Theorem 3: Assuming the availability of cryptographically secure pseudorandom functions and collision-resistant hash functions, Π_{BPC} securely realizes the preprocessing functionality, even in the presence of up to one corrupted party.

PROOF: Similar to the above, Π_{BPC} also divides the four relay parties into honest and corrupted ones, with the helper party H assumed to be semi-honest.

- *Case 1: P_1 is semi-honest.* P_1 only receives information-theoretic shares $\langle \mathcal{M}' \rangle_2, \langle \mathcal{M}' \rangle_3, \langle \mathcal{M}' \rangle_4$ from H and the non-linear offset permutation π from $\mathcal{P}$. Even after locally shifting $\langle \mathcal{M}' \rangle$ to obtain $\langle \mathcal{M} \rangle$, P_1 learns nothing about $\mathcal{M}$. The adversary's view remains indistinguishable from the ideal world.
- *Case 2: P_1 is malicious* Even if P_1 uses a random offset or permutation π to compromise $\mathcal{M}$, the silent execution equally defers the deviation to the reconstruction phase. That is, the server can neither infer BPC privacy nor deviate from the protocol to compromise the integrity of BPC under the honest-majority rule.

❸ **Online shuffling and tracing phase.** We next proceed to examine protocol Π_{4PS}, Π_{RMC}, and Π_{4PT} across $\mathcal{P}$ and $\mathcal{C}$.

Theorem 4: Π_{4PS} and Π_{RMC} securely realize privately robust shuffling functionality and achieve sender anonymity, against one possibly malicious party.

PROOF: Again, we consider a division of the four parties into honest and corrupted ones.

- *Case 1: P_1 is semi-honest.* In this case, P_1 only learns the $(4,2)$-RSS shares of $\langle \mathcal{M} \rangle_2, \langle \mathcal{M} \rangle_3, \langle \mathcal{M} \rangle_4$, and the $(4,3)$-RSS shares of $\mathcal{X}'$. The adversary gains no additional information about the underlying data or shuffle process, preserving the robustness and anonymity guarantees.
- *Case 2: P_1 is malicious.* If P_1 attempts to substitute the input/output shares in prior steps, due to the silent feature of Π_{4PS}, all deviations will be deferred to the reconstruction phase. For inconsistent output shares, Π_{RMC} generate both $\mathcal{X}'$ and $\mathcal{X}''$. One of them is guaranteed to be correct, and the other plays cover traffic.

In all cases, the correct batch of shuffled messages will be reconstructed without compromising sender anonymity.

Theorem 5: Given the public moderation policy $\mathcal{D}$, Π_{4PT} securely realizes traceability while preserving others' anonymity, against malicious users and up to one malicious party.

PROOF We examine the selective traceability properties of the Π_{4PT} protocol as below,

- *Case 1: P_1 is semi-honest.* If x'_a is reported as abuse by $\mathcal{C}$ over the threshold and audited across $\mathcal{P}$, then P_1 will request $\langle \mathcal{M}_{(a)} \rangle_1$ from P_2, P_3, P_4, reconstruct it, and search the index of 1 entry to reveal the abuse sender, without compromising the anonymity of others.
- *Case 2: P_1 is malicious.* If P_1 maliciously requests partial shares from other parties that violate the moderation policy $\mathcal{D}$, the blame mechanism will directly reveal P_1 as the malicious entity (see Fig 15(e)), all while ensuring no anonymity of honest users is compromised.

D. De-anonymization Attacks

In prior sections, we discuss the anonymity goal under the standard MPC security model, where adversary $\mathcal{A}$ has no prior knowledge of $\mathcal{X}$ and $\mathcal{M}$. However, a skillful $\mathcal{A}$ in the real world could attempt to narrow the anonymity set. Its capability via de-anonymization attacks can be examined below,

- *Case 1:* If $\mathcal{A}$ can compromise ρ clients out of a shuffle batch of size N but knows nothing about $\mathcal{M}$ (for simplicity, we assume the first ρ clients are compromised), the anonymity set reduces to $N-\rho$. The probability that $\mathcal{A}$ can correctly guess whether a message x_i was sent by a non-corrupted client C_i would be reduced to:
$$\mathsf{Pr}_{\exists x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{N-\rho},$$
and the probability of correctly identifying all messages x_i sent by the remaining clients becomes:
$$\mathsf{Pr}_{\forall x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{(N-\rho)!}.$$
- *Case 2:* If $\mathcal{A}$ cannot compromise any clients but possesses some partial knowledge η about the permutation matrix $\mathcal{M}$, which helps de-anonymize ϕ clients from the shuffle batch of size N, the anonymity set is reduced to $N-\phi$. In this case, the probability of guessing whether a message x_i was sent by a specific client C_i is:
$$\mathsf{Pr}_{\exists x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{N-\phi},$$
and the probability of de-anonymizing all messages sent by the remaining clients becomes:
$$\mathsf{Pr}_{\forall x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{(N-\phi)!}.$$

Case 3: If $\mathcal{A}$ can compromise ρ clients and also has partial knowledge η about $\mathcal{M}$ (where we assume these two factors are independent), the combined probability of correctly identifying a specific client C_i is:
$$\mathsf{Pr}_{\exists x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{N-\rho} + \frac{1}{N-\phi},$$
and the probability he can guess all x_i in the remaining anonymity set is sent by which user C_i is reduced to:
$$\mathsf{Pr}_{\forall x_i \leftarrow C_i}[\mathcal{A}(x_1^c, ..., x_\rho^c) = 1] = \frac{1}{(N-\rho)!} + \frac{1}{(N-\phi)!}.$$

These cases lead to a key yet common observation: a practical anonymous broadcast system should integrate access control mechanisms, such as reputation-based techniques [86] or CAPTCHA [85], to restrict $\mathcal{A}$'s influence to a limited subset of clients in each broadcast round, thereby maintaining robust system availability and strong sender anonymity. This highlights the need for careful system design, private random permutation selection, and trust domain separation.