Abuse Resistant Traceability with Minimal Trust for Encrypted Messaging Systems

Zhongming Wang†, Tao Xiang†, Xiaoguo Li†, Guomin Yang‡, Biwen Chen$^{\dagger(\boxtimes)}$, Ze Jiang†,
Jiacheng Wang§, Chuan Ma†, and Robert H. Deng‡
†Chongqing University, ‡Singapore Management University, §Nanyang Technological University
{zmwang, txiang, csxgli, macrochen, chuan.ma}@cqu.edu.cn, jiangze@stu.cqu.edu.cn,
{gmyang, robertdeng}@smu.edu.sg, jiacheng.wang@ntu.edu.sg

***Abstract*—Encrypted messaging systems provide end-to-end security for users but obstruct content moderation, making it difficult to combat online abuses. Traceability offers a promising solution by enabling platforms to identify the originator/spreader of messages, yet this capability can be abused for mass surveillance of innocent messages. To mitigate this risk, existing approaches restrict traceability to (problematic) messages that are reported by multiple users or are on a predefined blocklist. However, these solutions either overtrust a specific entity (e.g., the party defining the blocklist) or rely on the unrealistic assumption of non-collusion between servers run by a single platform.**

In this paper, we propose an abuse-resistant source tracing scheme that distributes traceability across distinct real-world entities. Specifically, we formally define its syntax and prove its security properties. Our scheme realizes two essential principles: *minimal trust*, which ensures that traceability cannot be abused as long as a single participant involved in tracing is honest, even if all others collude; and *minimal information disclosure*, which prevents participants from acquiring any information (e.g., communication parties' identities) unnecessary for tracing. We implemented our scheme using techniques deployed by Signal, and our evaluation shows it offers comparable performance to state-of-the-art schemes that are vulnerable to abuse.

I. Introduction

End-to-end encryption (E2EE) has been widely deployed in real-world messaging platforms such as WhatsApp and iMessage, serving billions of users daily. A persistent issue within end-to-end encrypted messaging systems (EEMSs) is the prevalence of online abuse, which includes hate, harassment, and misinformation [51]. To mitigate online abuse, platforms typically employ content-based methods (e.g., machine learning [24], [54]) to detect problematic messages. Unfortunately, these methods require access to the detected content (e.g., text, image, video) in plaintext, which is safeguarded by E2EE in EEMSs. Therefore, E2EE poses a significant challenge to combat online abuse within EEMSs [45], [50].

Traceability is a promising paradigm for addressing online abuse in EEMSs [55]. Instead of detecting problematic content in plaintext, traceability enables the platform to identify culprits responsible for disseminating problematic messages. Subsequently, the platform can implement targeted measures, such as suspending services [58], to impede the perpetrators of online abuse. In recent years, governments in various countries have also considered legislation to mandate traceability in EEMSs [38], [42]. However, traceability inevitably compromises the privacy guarantees offered by E2EE [57]. For instance, a malicious platform could abuse its traceability power to monitor innocent messages containing politically sensitive content. In contrast, a plausible traceability solution should only allow the platform to trace genuinely problematic messages. To address this challenge, we must define trace rules that determine which messages are permissible for tracing.

A. Prior Work

Existing tracing schemes primarily address traceability, exploring various tracing mechanisms including message traceback [33], [55], source tracing [11], [28], [41], and impact tracing [56]. These solutions permit the platform to trace any reported message, irrespective of whether it is problematic or innocuous, posing a significant risk of platform abuse. To mitigate this risk, two distinct approaches have been proposed.

Threshold-based approach [6], [37] limits the platform's traceability to messages reported multiple times by different recipients. The intuition behind implementing this approach is that recipients submit a share of tracing metadata with each report; the platform can trace the originator's identity once a sufficient number of shares have been collected. For instance, Liu et al. [37] and Bell et al. [6] implemented fuzzy and exact threshold reporting using a novel counting bloom filter and secret sharing, respectively. However, these schemes require either the platform or recipients to be honest. A recipient colluding with the platform could submit original tracing metadata instead of a share, thus nullifying the threshold limitation. Even worse, a malicious platform could create an arbitrary number of malicious recipients to report encrypted messages. This issue, analogous to the Sybil attack [19], is inherent to any threshold reporting scheme.

Blocklist-based approach [4] limits the platform's traceability only to messages included in a pre-defined blocklist. This requires a trusted regulator to define the blocklist, as the platform itself cannot be trusted with this task. For example, real-world regulators like the National Center for Missing and

$\boxtimes$ Biwen Chen is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2026.240456
www.ndss-symposium.org

Exploited Children (NCMEC) provide hashes of child sexual abuse materials (CSAM) to platforms like Google to detect problematic content [49]. However, a critical concern is that the regulator could also be dishonest. Such a regulator could covertly inject innocent messages into the blocklist, leading to mass surveillance. Moreover, the high sensitivity of the blocklist (e.g., child sexual exploitation) prevents the regulator from demonstrating that the messages within the blocklist are genuinely problematic. Consequently, the assumption of an honest regulator is fragile from a real-world perspective. This fragility is evidenced by trust issues in Apple's attempts to deploy their client-side scanning scheme [25], [31], [34].

In summary, while traceability helps curb online abuse, it inherently risks granting excessive power to the platform or regulator, potentially leading to mass surveillance. Existing solutions rely on strong trust assumptions—either in the platform, recipients, or regulator. This motivates us to seek an abuse-resistant traceability design that requires weaker trust assumptions among participants.

II. Overview

We now provide details of our design considerations, contributions, and techniques.

A. Design Considerations

To achieve abuse-resistant traceability, we first outline two essential properties for practical deployment:

Minimal trust for abuse resistance requires that a tracing scheme cannot be abused even if only one arbitrary participant, aside from the originator, is trusted. Ideally, a tracing scheme should prevent abuse even if all the participants, except the originator, are untrusted — even when they collude. Unfortunately, this goes against the core of traceability, which depends on non-originator participants to identify the originator. Without this capability, an originator could simply obstruct the disclosure of their identity and evade tracing. Therefore, achieving abuse-resistant traceability under such an ideal assumption is infeasible. We thus adopt a minimal trust model, assuming only one trusted non-originator participant and allowing arbitrary collusion among all other participants.

Minimal information disclosure ensures that only necessary information is revealed to servers (e.g., the platform) for tracing. This principle has two key aspects: First, a tracing scheme must minimize the collection of user information. This means that all information irrelevant to tracing problematic messages should be concealed. Specifically, this includes maintaining the confidentiality of all messages before tracing and ensuring the anonymity of communication parties and reporters. Second, the scheme must also minimize the exposure of detection information. For instance, illegal content databases like NCMEC's CSAM hash database should expose as little as possible. This is crucial because any unnecessary exposure of illegal content risks re-traumatizing survivors [16], [53]. These two aspects contribute to the scheme's abuse resistance, thereby improving overall user privacy and reducing data leakage risks.

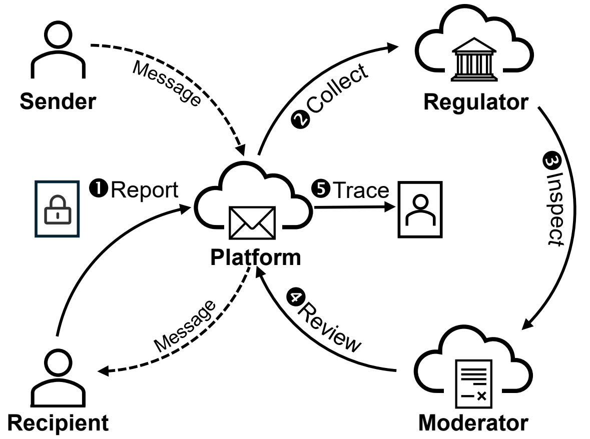

Fig. 1: Our system model. The solid lines represent the reporting-then-tracing process.

B. Our Contributions

In this paper, we aim to address the following question:

"Can we design an abuse-resistant tracing scheme with minimal trust and minimal information disclosure?"

We introduce a new system model grounded in the principle of minimal trust and propose a source tracing scheme that ensures minimal information disclosure. The system addresses the risk of abuse by assigning distinct roles and responsibilities to each participant, thereby establishing a system of mutual checks and balances.

New model for traceability with minimal trust. Our system model draws from observations of real-world moderation systems, which invariably include participants beyond the platform. For instance, a regulator (like NCMEC) manages a blocklist of illegal content, while a moderator (similar to the Oversight Board [9]) reviews the platform's (e.g., Meta's) content moderation decisions. Users interact solely with the platform, with the moderator and regulator operating transparently in the background.

Based on these observations, we distribute tracing responsibilities across four distinct participants (see Fig. 1):

- *Users*: ① Report unwanted messages they receive.
- *Platform*: ② Collect reports from recipients in reporting; and ⑤ Trace the originator of problematic messages in tracing.
- *Regulator*: ③ Inspect reports collected by the platform to filter out invalid ones (i.e., reports on innocent messages).
- *Moderator*: ④ Review moderation decisions made by the regulator and transmit only valid ones to the platform.

Meanwhile, our system adopts a threat model grounded in minimal trust: it is abuse-resistant if any single participant (platform, moderator, regulator, or recipients) is trusted, even if all others collude. This represents a weaker trust assumption than in prior single- and multi-server schemes. We contrast this model with existing approaches to demonstrate its advantages:

- *Single-server schemes* (e.g., [41], [55]) centralize trust in the platform. While such models can satisfy minimal trust, they

grant the platform unilateral power to trace any message and creating a high risk of abuse.
- *Multi-server schemes* attempt to distribute trust but introduce other weaknesses. Those relying on a single designated trusted server [4], [21] merely shift the single point of trust. Other schemes employ secure multi-party computation (MPC) with symmetric servers [6], [36] have two weaknesses: 1) The servers are often run by a single entity, collapsing the distinct trust domains and voiding MPC guarantees [32], and 2) the symmetric architecture requires all servers to view the reported messages for moderation, harming privacy.

Our model avoids these pitfalls by assigning asymmetric roles to participants, which ensures both minimal trust and minimal information disclosure. The regulator, moderator, and platform each have distinct, non-overlapping responsibilities and access privileges, preventing any single entity from unilaterally controlling the tracing process.

New scheme for abuse-resistant source tracing. Within our minimal-trust model, we design a new sourcing tracing scheme that prevents abuse while minimizing information disclosure. Our scheme achieves this by distributing responsibilities among its participants, who collectively enforce specific rules to determine message traceability.

In detail, our regulator uses a private blocklist to filter reports, a design that minimizes detection information disclosure. The blocklist's content remains exclusively with the regulator. However, this non-public list carries a risk: a malicious regulator could potentially add innocent messages to the blocklist for tracing. To counter this, our moderator independently reviews the regulator's decisions, only forwarding valid cases to the platform. Furthermore, honest recipients prevent abuse, even if all servers are malicious, by submitting just one share per report. This stops servers from collecting enough shares to trace innocent messages.

As a result, our system permits tracing only for messages that simultaneously meet three rules: 1) reported by multiple recipients; 2) defined by the regulator in a blocklist; and 3) reviewed by the moderator. This approach combines the principles of threshold reporting, blocklisting, and content reviewing. Consequently, we define "innocent messages" as any messages that do not satisfy this complete set of rules; "abuse" is any information leakage about these innocent messages.

With minimal trust, our scheme can naturally withstand traceability abuse even when an adversary controls all participants but one. The single trusted participant can independently thwart abuse in the following ways:

- An *honest platform* prevents the adversary from obtaining the tracing result, even if the adversary can bypass the rules.
- An *honest regulator* maintains a blocklist devoid of innocent messages, thereby preventing the adversary from tracing a message outside the blocklist.
- An *honest moderator* faithfully reviews the received reported messages, filtering out invalid reports on innocent messages from the adversary.

TABLE I: Comparison with prior work.

Scheme	Trace Rule	Trust Assumption	Anon.	Basic Security
TMR19 [55]	Unrestricted	Honest platform	○	●
PEB21 [41]	Unrestricted	Honest platform	○	●
IAV22 [28]	Unrestricted	Honest moderator†	●	●
BGJP23 [4]	Pre-emptive blocklist	Honest regulator	●	●
LRTY22 [37]	Fuzzy threshold	Non-colluding platform and recipients‡	◐	●
BE24 [6]	Exact threshold		●	●
Ours	Exact threshold Post-facto blocklist Content review	Minimal trust	●	●

Basic Security: Confidentiality, unforgeability, and accountability.
Anon. (Anonymity): ◐ represents one-sided anonymity (i.e., at the level of Signal's sealed sender [30]).
Trust Assumption: †: The moderator is a decoupled tracing authority, fundamentally differing from our moderator; ‡: BE24 uses two servers and additionally assumes they do not collude.

- *Honest recipients* submit only a share of the tracing metadata as a report. Thus, the adversary cannot trace a message without obtaining sufficient reports.

Furthermore, our scheme enforces minimal user information disclosure, preventing unnecessary data leakage across the board. This principle applies to innocent messages, where both their content and originator remain permanently concealed. Even for problematic messages, we apply this principle stringently. For example, after multiple reports, the platform only reconstructs an encrypted problematic message for the regulator. The regulator can then only decrypt and view its content if the encrypted message belongs to the blocklist. Additionally, the identities of communication parties and reporters are permanently concealed from the platform across all messages. This makes our scheme compatible with EEMSs that feature anonymity [30].

More improvements beyond abuse resistance. Beyond its core abuse resistance, our scheme offers several advancements over prior work, which we summarize in Table I.

- *Source tracing:* Our source tracing scheme is a generic construction, offering cryptographic agility [27]. Moreover, it leverages existing techniques, such as key-verification anonymous credential (as used in Signal's private group system [13]), which makes it easy to deploy.
- *Blocklisting:* Our blocklisting implementation checks all reported messages against the most current blocklist (i.e., post-facto), unlike the previous scheme's [4] pre-emptive approach, which checks at message creation. This timing mismatch limits their ability to trace most problematic messages, as such messages are typically blocklisted only after they have begun to spread.
- *Threshold reporting:* Our threshold reporting implementation achieves an exact reporting threshold between users and the platform. This contrasts with prior work that either offered a fuzzy threshold that risks false positives (tracing the originator of insufficiently reported messages) [37], or required two servers that are difficult to deploy due to their impractical trust assumptions [6], [36].

C. Our Techniques

Our construction proceeds in two stages: first, we implement a source tracing scheme within anonymous EEMSs; second, we assign servers different authorities to enforce distinct trace rules for abuse resistance.

Source tracing with anonymity. The foundation of our scheme is tracing metadata tmd that accompanies each message, enabling traceability within EEMSs.

- *Sending.* A sender attaches an tmd to every message.
- *Receiving.* Upon receipt, the recipient verifies the validity of tmd and rejects any messages with malformed metadata.
- *Reporting.* A recipient reports an unwanted message by submitting it to the servers along with its tmd.
- *Tracing.* The servers first verify the validity of tmd. If tmd is valid, they recover the originator's identity from it.

Additionally, throughout the process of messaging, tmd is hidden in E2EE and remains unreadable to the platform, thus preserving the anonymity of the underlying EEMS.

Building on the workflow described, this tracing metadata is designed with four components to simultaneously ensure *traceability*, *accountability*, and *confidentiality*:

- *Encrypted originator identity* (ct_u): The originator's identity is encrypted using server's public keys. This provides traceability, allowing servers to recover the originator's identity in tracing, while ensuring confidentiality from all other parties.
- *Anonymous token* $(token)$: A publicly verifiable token, signed by the platform, that attests to the originator's validity without revealing their actual identity. This provides accountability, preventing malicious users from impersonating others while preserving anonymity among recipients.
- *Encrypted tag* (ct_{tag}): A tag binding the originator's identity to the message content. It ensures accountability, preventing malicious users from framing others by fabricating messages they never sent.
- *Zero-knowledge consistency proof* (π_{send}): A proof showing that 1) the identity within $(ct_u, token, ct_{tag})$ is identical and valid; 2) the message in ct_{tag} matches the verifier's received message. This makes the entire metadata verifiable by any participant, ensuring no malicious sender can evade tracing, which guarantees accountability and confidentiality. Note that this proof can be efficiently instantiated using the generic linear Sigma protocol [10].

Next, to achieve abuse resistance, our scheme protects the originator's identity with a layered ciphertext ct_u. Its decryption requires sequential approval from the regulator, moderator, and platform, where each step is conditioned on the enforcement of a distinct rule. The entire process is gated by the platform collecting enough reports from recipients, which is followed by the regulator's blocklist check and the moderator's content review.

Threshold reporting. Our scheme adapts the threshold aggregation reporting (TAR) protocol [15] for threshold reporting. Given a shared secret (e.g., a reported message) between users, TAR allows individual users to produce a secret share. The platform can only reconstruct this secret when collecting shares that exceed a predefined threshold. Unfortunately, the original TAR protocol is vulnerable to malicious reporters who can submit duplicate reports for the same message to circumvent the threshold. We address this by introducing a report deduplication mechanism. Each reporter generates a deterministic deduplicate tag that uniquely binds their identity to the reported message. The platform can then discard duplicate reports by detecting repeated tags, which ensures each user contributes only once to a message's report count.

Blocklisting. We use set pre-constrained encryption (SPCE) in [4] for blocklisting, but with a critical architectural shift. Unlike that work, which binds a message to a blocklist at creation time, we defer this binding to reporting time. This simple change ensures every report is moderated against the most current blocklist. This deferred-binding approach was previously infeasible. In prior models, a malicious recipient and platform could collude to bypass the binding check and maliciously trace an innocent message. This attack works because the platform is the only server to verify the recipient's report. Our model solves this by making the regulator an active validator of blocklisting, where all reports require validation by the regulator before tracing. This neutralizes the threat of collusion between recipients and the platform, as a trusted regulator can immediately reject false reports.

Content reviewing. A final challenge is to prevent a malicious regulator from abusing its power by adding innocent messages to the blocklist to enable their tracing. A straightforward method is to engage a group of third parties to audit the blocklist at the system setup [44]. But, this method discloses the messages within the blocklist (e.g., CSAMs) to these auditors, violating minimal information disclosure. In particular, except for tracked messages, participants other than the regulator should learn nothing about the blocklist.

Our solution is to review the outcome of the regulator's inspection, not the blocklist itself. The moderator reviews only the messages that the regulator has already flagged as being on the blocklist and ready for tracing. This approach is effective for two reasons: 1) Abuse only occurs when an innocent message is actually traced, not just when it is added to the blocklist. Our review happens at the last possible moment before tracing. 2) It minimizes information disclosure, as the moderator only sees messages that have met all other trace rules, not the entire sensitive blocklist. This strategy prevents the regulator from abusing its power while protecting the confidentiality of the blocklist.

III. Models and Goals

A. Participants

There are four types of participants involved in our system model. Each participant behaves as follows.

- *Users* $\mathcal{U}$: Users communicate with each other via the underlying EEMS, acting simultaneously as sender $\mathcal{U}_S$ and recipient $\mathcal{U}_R$. Additionally, recipients can report any unwanted message to the platform.
- *Platform* $\mathcal{P}$: The platform authenticates user's identities and transmits encrypted messages from senders to recipients. It

also processes (i.e., verifies, collects, and aggregates) user reports, transmitting the aggregated result to the moderator. Upon receiving reviewed tracing metadata from the moderator, it discloses the originator of reported messages for further targeted interventions.

- *Regulator $\mathcal{R}$*: The regulator generates and maintains a regularly (e.g., weekly) updated blocklist, ruling what messages are considered problematic. It inspects aggregated reports from the platform and forwards valid ones to the moderator for further review.
- *Moderator $\mathcal{M}$*: The moderator is responsible for reviewing content moderation decisions from the regulator. For valid decisions, it transmits the tracing metadata to the platform to trigger tracing. Conversely, it ceases to forward the metadata to the platform for invalid decisions.

B. Threat Model

All participants in our system can be either semi-honest or malicious, with their behaviors detailed below:

- *Users $\mathcal{U}$*: Semi-honest users correctly execute our scheme but attempt to deduce the originator of received messages from the corresponding tracing metadata. Malicious users would strive to evade tracing, falsely attribute a problematic message's originator to an innocent user, or intentionally report innocent messages to maliciously trigger a trace.
- *Platform $\mathcal{P}$*: A malicious platform seeks to compromise users' privacy, targeting the content and originator's identity of innocent messages, as well as the identities of the communicating parties and reporters. Additionally, it may bypass the verification of reports to circumvent trace rules.
- *Regulator $\mathcal{R}$*: A semi-honest regulator generates and enforces the blocklist but seeks to disclose the reported message's originator. Conversely, a malicious regulator may inject innocent messages into the blocklist.
- *Moderator $\mathcal{M}$*: A semi-honest moderator dutifully verifies content moderation decisions from the regulator but attempts to infer the reported message's originator. Conversely, a malicious moderator may neglect to revise received content moderation decisions.

Our threat model allows all malicious participants to collude, as long as the minimal trust principle introduced in Section II-B holds. This principle is dynamically applied across our security games (Section VI), where the scope of collusion is tailored to the participants relevant to each specific property. For instance, in analyzing abuse resistance—a property involving recipients, the platform, the regulator, and the moderator—the threat model assumes an adversary that can corrupt any three of these four parties.

C. Design Goals

Our goal is to implement a source tracing scheme that prevents the potential abuse of traceability. Consequently, the design goals of our scheme encompass three key aspects:

First, our scheme should satisfy all the requirements of a conventional source tracing scheme.

- *Message confidentiality*: Participants other than the users who received the message directly learn nothing about the content of the messages before reporting.
- *Source confidentiality*: Participants cannot determine the message's originator from the tracing metadata.
- *Trace accountability*: A malicious user cannot send a message that cannot be traced in the future, and a group of malicious users cannot manipulate the tracing result.
- *Messaging anonymity*: The identities of communication parties are concealed from the servers[1] in messaging.

Second, threshold reporting requires that a user's report for a specific message be unique and anonymous.

- *Report uniqueness*: A user cannot report the same forwarded message to the platform twice.
- *Reporting anonymity*: The platform cannot infer users' identities from their reports.

Third, blocklisting mandates that the blocklist used for filtering reports be certified by the regulator and remain confidential to all participants except the regulator itself.

- *Blocklist confidentiality*: Participants other than the regulator learn nothing about the content of the blocklist from the blocklist, except for reported messages.
- *Blocklist authenticity*: Only the regulator can add or update elements in the blocklist.

Remark 1 (Source replacement). A user may copy and paste a received message to re-send it, rather than forward it. Consequently, the user would be traced as the originator, replacing the actual originator. Since the underlying EEMS can only identify the first user who sent a message as its originator, rather than the one who created it in the real world, this is an inherent limitation of any tracing scheme [41], [55]. Therefore, we consider it a non-goal of this work.

IV. Definitions

This section defines the syntax of our scheme and its core building blocks.

A. Preliminaries

Notations. If S is a set, then $|S|$ represents its cardinality and $a \leftarrow\!\!\$\ S$ uniformly samples an element from S. For two sets S_1 and S_2, we use $S_1 \overset{\cup}{\leftarrow} S_2$ to denote $S_1 \leftarrow S_1 \cup S_2$. Additionally, $[n]$ represents the set of integers $\{1, 2, \ldots, n\}$ and the dot notation is employed to access the element of a tuple. For example, we use $token.cm$ to retrieve the field cm of tuple $token = (cm, \sigma_{sw}, rd)$.

Algorithms. If Alg is an interactive algorithm that receives $input$ from $\mathcal{S}$ and produces $output$ for $\mathcal{T}$, we denote is as: $\mathsf{Alg}\langle \mathcal{S}(input), \cdots \rangle \to (\mathcal{T}(output), \cdots)$ or $\perp$, where $\mathcal{S}, \mathcal{T} \in \{\mathcal{R}, \mathcal{M}, \mathcal{P}, \mathcal{U}\}$. The symbol $\perp$ indicates the failure of algorithm execution, and $\cdots$ indicates that it receives additional inputs from and produces outputs for various participants.

[1] For simplicity, we use 'servers' to refer to $\mathcal{R}$, $\mathcal{M}$, and $\mathcal{P}$ henceforth.

B. Syntax

At a high level, our scheme consists of five phases:

Setup phase. The servers generate their key-pairs and initialize system parameters.

- $\mathsf{KGen}(1^\lambda) \to (\mathcal{R}(pk_R, sk_R), \mathcal{M}(pk_M, sk_M), \mathcal{P}(pk_P, sk_P))$: The key generation protocol lets the servers initialize their key-pairs individually.
- $\mathsf{LGen}(\mathcal{B}, sk_R) \to eb$: The blocklist generation protocol enables the regulator to encode a blocklist $\mathcal{B}$ to eb.

In summary, the system parameters are the threshold δ (set by servers) and the encoded blocklist eb. We assume $(pk_R, pk_M, pk_P, \delta, eb)$ is an implicit input for all other algorithms.

Registration phase. The users register to the platform and obtain publicly verifiable anonymous tokens for 1) originating fresh messages and 2) reporting problematic messages.

- $\mathsf{UReg}\langle\mathcal{U}(id), \mathcal{P}(sk_P)\rangle \to \mathcal{U}(uk, cred)$ *or* $\perp$: The user registration algorithm, executed only once for each new user by $\mathcal{P}$, generates a secret key uk and a corresponding long-term anonymous credential $cred$ for the user.
- $\mathsf{TkGen}\langle\mathcal{U}(uk, cred), \mathcal{P}(sk_P)\rangle \to \mathcal{U}(token)$ *or* $\perp$: The token generation protocol, executed interactively between $\mathcal{P}$ and $\mathcal{U}$, allows $\mathcal{U}$ to derive a publicly verifiable ephemeral anonymous token from its long-term credential.

Message phase. Users communicate with each other via the platform. Meanwhile, users generate tracing metadata for messages they originate, allowing tracing in the future.

- $\mathsf{Send}(m, uk, token) \to tmd$: The send algorithm allows a user to generate tracing metadata using its secret key and an anonymous token when originating a message.
- $\mathsf{Receive}(m, tmd) \to \{0,1\}$: The receive algorithm, run by the recipient, checks the validity of received tracing metadata and rejects the malformed ones. This ensures that every reported message can be traced to its originator.

Report phase. The recipients report unwanted messages, and then the platform processes these reports for tracing.

- $\mathsf{Report}(m, tmd, uk, cred, eb) \to report$: The report algorithm, run by the recipient, allows the user to generate a report of a received unwanted message.
- $\mathsf{VfReport}(report, sk_P) \to report'$ *or* $\perp$: The report verification algorithm is run by $\mathcal{P}$ to check the validity of a received report. If verification succeeds, it outputs a refined report; otherwise, it outputs $\perp$.
- $\mathsf{Collect}(\{report'_i\}_{i\in[n]}) \to ct_{tmd,3}$ *or* $\perp$: The collect algorithm enables the platform to collect and aggregate refined reports to encrypted tracing metadata.

Trace phase. The regulator and moderator filter out invalid reports, allowing the platform to trace only valid reports.

- $\mathsf{Inspect}(ct_{tmd,3}, sk_R) \to (ct_{tmd,2}, m)$ *or* $\perp$: The inspect algorithm, executed by $\mathcal{R}$ to handle user reports, verifies the compliance of these reports with a trace rule. If the rule is satisfied, $\mathcal{R}$ obtains the message m and updates the encrypted tracing metadata $ct_{tmd,3}$, thereby signaling $\mathcal{M}$ for further reviews.
- $\mathsf{Review}(ct_{tmd,2}, m, sk_M) \to (ct_{tmd,1}, m)$ *or* $\perp$: The review algorithm, operated by $\mathcal{M}$, evaluates the content moderation decision on the reported message. If $\mathcal{M}$ deems the message problematic, it updates the encrypted tracing metadata $ct_{tmd,2}$, signaling $\mathcal{P}$ to commence tracing.
- $\mathsf{Trace}(ct_{tmd,1}, m, sk_P) \to id$ *or* $\perp$: The trace algorithm, run by $\mathcal{P}$, validates the encrypted tracing metadata $ct_{tmd,1}$. If validation succeeds, it reveals the originator's identity of the reported message m.

C. Building Blocks

We next define the building blocks of our design and briefly discuss the core properties required.

Keyed-verification anonymous credential (KVAC). A KVAC protocol [12] allows the server $\mathcal{S}$ to (blindly) issue credentials to clients; with the credential, the client $\mathcal{C}$ can anonymously authenticate themselves to $\mathcal{S}$.

- $\mathsf{KGen}(1^\lambda) \to (pp, k_S)$: The key generation algorithm enables $\mathcal{S}$ to create public parameters pp and secret key k_S.
- $\mathsf{Issue}\langle\mathcal{C}(id), \mathcal{S}(k_S)\rangle \to \mathcal{C}(cred, \pi_{iss})$: The credential issuance algorithm produces a credential $cred$ with proof π_{iss} for $\mathcal{C}$ with identity id. In addition, for blind issuance, please refer to BlindIssue in Appendix A.
- $\mathsf{VfIssue}(id, cred, \pi_{iss}) \to \{0,1\}$: The issuance verification algorithm allows $\mathcal{C}$ to verify the proof of issuance π_{iss}, outputting '1' if the verification is successful.
- $\mathsf{Show}(id, cred) \to (cm, \pi_{show}, rd)$: The present algorithm enables $\mathcal{C}$ to generate a presentation (cm, π_{show}, rd) to authenticate its identity to $\mathcal{S}$ anonymously, where cm is a commitment to id and rd is randomness used in cm.
- $\mathsf{VfShow}(k_S, cm, \pi_{show}) \to \{0,1\}$: The show verification algorithm allows $\mathcal{S}$ to verify a credential presentation's validity, outputting '1' for success or '0' for failure.

A KVAC protocol should satisfy: 1) *Unforgeability:* A malicious client cannot generate an accepting proof for an identity they never received a corresponding credential. 2) *Anonymity:* A credential presentation from Show reveals nothing about the client's identity. Notably, KVAC's security implies the binding and hiding properties of the commitment cm.

Multi-key public key encryption (mkPKE). A mkPKE protocol enables a message to be encrypted with multiple public keys, while decryption requires all corresponding secret keys. We provide a definition involving three keys:

- $\mathsf{KGen}(1^\lambda) \to (pk, sk)$: The key generation algorithm takes in a security parameter and outputs a key-pair.
- $\mathsf{Enc}(pk_3, pk_2, pk_1, m) \to ct_3$: The encryption algorithm takes in three public keys and a message, producing a third-level ciphertext.
- $\mathsf{Dec}_3(sk_3, ct_3) \to ct_2$: The decryption algorithm takes in the third-level ciphertext and the secret key sk_3, and outputs a second-level ciphertext.
- $\mathsf{Dec}_2(sk_2, ct_2) \to ct_1$: The decryption algorithm takes in the second-level ciphertext and the secret key sk_3, and outputs a first-level ciphertext.

- $\mathsf{Dec}_1(sk_1, ct_1) \to m$: The decryption algorithm takes in the first-level ciphertext and the secret key sk_1, and outputs a message.

Similar to a typical PKE's *confidentiality*, mkPKE ensures that ciphertexts at all levels (ct_3, ct_2, ct_1) are semantically secure, thereby revealing nothing about the message.

Threshold aggregation reporting (TAR). A TAR [15] protocol enables a server to collect shared secrets from clients only when a pre-defined threshold of contributions is met.

- $\mathsf{Share}(\delta, S) \to (ek, lb, sh)$: The share algorithm, executed by a client, accepts a threshold δ and a secret S, producing a message label lb, a private key ek, and a share sh.
- $\mathsf{Aggregate}(\delta, \{sh_i\}_{i\in[n]}) \to ek$: The aggregate algorithm, executed by the server, takes a set of shares and recover the key ek when the number of shares exceeds δ.

A TAR protocol ensures *privacy* that a server with fewer than the threshold number of shares gains no information about the encryption key.

Set pre-constrained encryption (SPCE). SPCE [4] ensures that only ciphertexts containing messages from a pre-defined blocklist are decryptable, even with a malicious key generator. It consists of a server $\mathcal{S}$ and clients $\mathcal{C}$.

- $\mathsf{Setup}(1^\lambda, \mathcal{B}) \to (sp, eb, pk_S, sk_S)$: The setup algorithm is executed by $\mathcal{S}$. It generates system parameters sp, initializes a key-pair (pk_S, sk_S), and encodes the blocklist $\mathcal{B}$ to eb. Finally, it publishes (sp, eb, pk_S) to all the clients.
- $\mathsf{Enc}(pk_S, m, p, eb) \to ct \ or \ \perp$: The encryption algorithm is run by $\mathcal{C}$, starting with the encoded blocklist eb's verification. Subsequently, $\mathcal{C}$ encrypts the payload p using a message m and eb, producing a ciphertext ct.
- $\mathsf{Dec}(sk_S, ct) \to p \ or \ \perp$: The decryption algorithm allows $\mathcal{S}$ to decrypt a ciphertext ct only when the corresponding message belongs to the blocklist (i.e., $m \in \mathcal{B}$).

A SPCE scheme satisfies: 1) *Server privacy:* Clients gain no knowledge of the blocklist's content from the corresponding encoded blocklist eb. 2) *Client privacy:* The server learns nothing about the payload from a ciphertext if the message is not in the blocklist.

V. Design

This section presents the construction of our abuse-resistant source tracing scheme, along with the instantiations of its building blocks.

A. Full Construction

Our full construction is formalized in Fig. 2 and Fig. 3, with its primary building blocks defined in Section IV. Additionally, we specify the other primitives involved as follows:

- $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme that satisfies random key robustness (RKR) [1], [22]. RKR guarantees that a ciphertext can only be decrypted with the correct key; otherwise, Dec outputs $\perp$ indicating a failure.
- $\mathsf{MAC} = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{Vf})$ is a deterministic message authentication code scheme, where the generated tag also satisfies weak pseudorandomness [18].
- $\mathsf{DS} = (\mathsf{Sign}, \mathsf{Vf})$ is a digital signature scheme.
- $\mathsf{Com} = (\mathsf{Commit}, \mathsf{Vf})$ is a commitment scheme.
- $\mathsf{PoK} = (\mathsf{Prv}, \mathsf{Vf})$ is a non-interactive zero-knowledge proof of knowledge (NIZKPoK) scheme. We represent the proof generation and verification as: $\pi \leftarrow \mathsf{Prv}(\mathbf{y}, \mathbf{x}); \{0,1\} \leftarrow \mathsf{Vf}(\mathbf{y}, \pi)$, where $\mathbf{y}$, $\mathbf{x}$, and π are a statement, witness, and proof, respectively.

Below, we describe the algorithms in different phases.

Setup phase enables the servers to initialize the system.

In KGen, servers independently execute key generation algorithms of the underlying building blocks. Specifically, all servers generate a key-pair for mkPKE. Additionally, the platform generates separate key-pairs for KVAC and DS. For simplicity, we use $(pk_{\mathcal{R}/\mathcal{M}/\mathcal{P}}, sk_{\mathcal{R}/\mathcal{M}/\mathcal{P}})$ to denote the servers' public and secret keys, even when used for different building blocks like KVAC or DS.

Subsequently, in LGen, the regulator initializes an SPCE key-pair and generates an encoded blocklist eb of problematic messages using $\mathsf{SPCE.Setup}(1^\lambda, \mathcal{B})$. This blocklist $\mathcal{B}$ is a set of problematic messages defined by the regulator. The server privacy of SPCE guarantees that eb leaks nothing about $\mathcal{B}$ to other participants, thus preserving blocklist confidentiality.

In addition, the servers configure a reporting threshold δ. Finally, they publish their public keys (pk_R, pk_M, pk_P), the encoded blocklist eb, and the threshold δ to all users.

Registration phase consists of a one-time user registration process and a multi-time token generation process.

Using UReg, a user id requests an anonymous identity from the platform upon joining the system. The platform responds to the user with a random key uk and a corresponding credential $cred$ using KVAC.Issue. Besides, the platform records the tuple (id, uk) for future tracing.

Next, using TkGen, the user converts the received *key verification* anonymous credential $(cred)$ into a *publicly verifiable* anonymous token $(token)$ over an anonymous channel (e.g., privacy relay [3] and oblivious HTTP [52]). Specifically, the user submits a credential presentation derived from $cred$ to the platform. The platform returns a signature on the presentation after successfully verifying it. Ultimately, the user obtains an anonymous token (cm, σ, rd), where cm is a commitment on uk. This anonymous token overcomes KVAC's limitation on the credential verifier, allowing any participant to verify a user's identity validity, rather than only the platform. Additionally, since these tokens are independent of messages, users can request these tokens during off-peak times without affecting messaging latency. The intuition behind TkGen is analogous to the pre-processing in Hecate [28].

Message phase includes sending and receiving a message.

Whenever a message is sent, the sender transmits a tuple (m, tmd) to the recipient. Depending on the message type, there are two sources of the tracing metadata tmd. Specifically, the sender executes Send to generate a new tmd for a fresh message, while reusing the received tmd for a forwarded message. The sender generates tmd as follows:

- *Encrypted originator identity* (ct_u): An encryption of uk

```
$\mathsf{UReg}\langle\mathcal{U}(id),\mathcal{P}(sk_P)\rangle$ $[\mathcal{P}\leftrightarrow\mathcal{U}]$
/ $\mathcal{P}$ generates $\mathcal{U}$'s key and credential.
$uk \leftarrow\$\ \mathsf{MAC.KGen}(1^\lambda)$
$\mathcal{L}_{id} \overset{\cup}{\leftarrow} \{(id, uk)\}$
$(cred, \pi_{iss}) \leftarrow\$\ \mathsf{KVAC.Issue}(uk, sk_P)$
/ $\mathcal{U}$ verifies the issued credential.
if $\mathsf{KVAC.VfIssue}(uk, cred, \pi_{iss}) = 0$ :
    return $\perp$
return $(uk, cred)$
```

```
$\mathsf{TkGen}\langle\mathcal{U}(uk, cred),\mathcal{P}(sk_P)\rangle$ $[\mathcal{U}\leftrightarrow\mathcal{P}]$
/ $\mathcal{U}$ generates credential presentation.
$(cm, \pi_{sw}, rd) \leftarrow\$\ \mathsf{KVAC.Show}(uk, cred)$
/ $\mathcal{P}$ verifies and signs valid presentation.
if $\mathsf{KVAC.VfShow}(sk_P, cm, \pi_{sw}) = 0$ :
    return $\perp$
$\sigma \leftarrow \mathsf{DS.Sign}(sk_P, cm)$
$token = (cm, \sigma, rd)$
return $token$
```

```
Relation $\phi_{send}$
Instance: $\mathbf{y} = (m, pk_R, pk_M, pk_P,$
                $cm, ct_{u,3}, ct_{tag})$
Witness: $\mathbf{x} = (uk, rd)$
$\phi_{send}(\mathbf{y}, \mathbf{x}) = 1$, iff.:
    $\mathsf{mkPKE.Enc}(pk_R, pk_M, pk_P, uk) = ct_{u,3}$
    $\wedge\ \mathsf{mkPKE.Enc}(pk_R, pk_M, pk_P, tag) = ct_{tag}$
    $\wedge\ \mathsf{MAC.Vf}(uk, m, tag) = 1$
    $\wedge\ \mathsf{Com.Vf}(uk, cm, rd) = 1$
```

```
$\mathsf{Send}(m, uk, token)$ $[\mathcal{U}_S\rightarrow\mathcal{U}_R]$
$tag \leftarrow \mathsf{MAC.Tag}(uk, m)$
$ct_{u,3} \leftarrow \mathsf{mkPKE.Enc}(pk_R, pk_M, pk_P, uk)$
$ct_{tag} \leftarrow \mathsf{mkPKE.Enc}(pk_R, pk_M, pk_P, tag)$
$\pi_{send} \leftarrow \mathsf{PoK.Prv}((m, pk_R, pk_M, pk_P, cm, ct_{u,3}, ct_{tag}), (uk, rd))$
$token' \leftarrow (token.cm, token.\sigma)$
$tmd = (token', ct_{u,3}, ct_{tag}, \pi_{send})$
return $tmd$
```

```
$\mathsf{Receive}(m, tmd)$ $[\mathcal{U}_R\rightarrow\mathcal{U}_R]$
if $\mathsf{DS.Vf}(\sigma, pk_P, cm) = 0$ :
    return $0$
if $\mathsf{PoK.Vf}((m, pk_R, pk_M, pk_P, cm, ct_{u,3}, ct_{tag}), \pi_{send}) = 0$ :
    return $0$
return $1$
```

Fig. 2: Algorithms in the registration and message phase. The process of initializing server keys and encoding the blocklist is described in the setup phase text. The notation $[\mathcal{A} \rightarrow \mathcal{B}]$ indicates a non-interactive algorithm executed by $\mathcal{A}$, which returns the output to $\mathcal{B}$. In contrast, $[\mathcal{A} \leftrightarrow \mathcal{B}]$ denotes an interactive algorithm between $\mathcal{A}$ and $\mathcal{B}$.

using multi-key PKE could be decrypted only if all three servers are involved. Note that, as users' key uk is bound to their identity id in UReg, we treat uk as users' identity after user registration.

- *Anonymous token* ($token$): A user obtains multiple tokens in the registration phase and uses a different token for each generated tracing metadata.
- *Encrypted tag* (ct_{tag}): A multi-key PKE ciphertext of a MAC tag that combines uk and m.
- *Zero-knowledge consistency proof* (π_{send}): Prove that the above components are well-formed, i.e., 1) uk remains consistent across these components, and 2) the message sent via E2EE matches the one committed in ct_{tag}.

Upon receiving a message with tracing metadata (m, tmd), a recipient verifies it using Receive. This process involves two key aspects: 1) Signature verification: The signature σ authenticates the originator's key uk within tmd. 2) Proof verification: The proof π_{send} ensures that tmd is well-formed, as defined by its relation ϕ_{send} in Fig. 2.

Report phase involves submitting reports by recipients and processing these reports by the platform.

To report an unwanted message using Report, a recipient generates a report comprising three functionalities:

- *Achieving traceability* (lb, sh', ct'_{tmd}): Encrypt the tuple (m, tmd) using SPCE, and subsequently divide the ciphertext into shares utilizing TAR.
- *Authenticating identity* (cm, π_{sw}): Run KVAC.Show to generate a credential presentation from the recipient's credential obtained in UReg.
- *Ensuring report uniqueness* (dup, π_{rpt}): Generate a duplicate tag dup that deterministic binds the message label lb to the recipient's key uk . Two identical tags will appear when a recipient reports the same message twice.

Upon receiving a report, the platform use VfReport to verify two aspects: 1) The validity of the reporter's identity. 2) The well-formedness of the duplicate tag. If these verifications succeed, the platform stores a refined report, removing the verification components $(cm, \pi_{sw}, \pi_{rpt})$ from the original.

Once the platform collects enough reports with the same label lb, it executes Collect to aggregate them. This algorithm begins with deduplication, where reports sharing the same tag dup are identified and removed. Next, the key ek is aggregated from the remaining unique reports. Finally, the platform uses ek to decrypt the SPCE ciphertext $ct_{tmd,3}$ from ct'_{tmd}, and sends it to the regulator for inspection.

Trace phase consists of three steps executed sequentially by the regulator, moderator, and platform.

First, the regulator $\mathcal{R}$ run Inspect to check whether the reported message m is on the blocklist $\mathcal{B}$. This check is uses SPCE.Dec. If $m \notin \mathcal{B}$, SPCE.Dec outputs $\perp$, ensuring that $\mathcal{R}$ learns nothing about innocent messages outside $\mathcal{B}$. On the other hand, when $m \in \mathcal{B}$, $\mathcal{R}$ recovers (m, tmd) from the received ciphertext $ct_{tmd,3}$. Next, $\mathcal{R}$ uses $sk_{\mathcal{R}}$ to decrypt the encrypted originator identity $ct_{u,3}$ within tmd to $ct_{u,2}$. The proof $\pi_{d,2}$ guarantees this decryption's correctness, preventing a malicious $\mathcal{R}$ from swapping $ct_{u,2}$ with an innocent user's encrypted identity and violating trace rules. In the end, $\mathcal{R}$ forwards the tuple $(m, tmd, ct_{u,2}, \pi_{d,2})$ to the moderator. Second, the moderator $\mathcal{M}$ executes Review to evaluate the reported message from $\mathcal{R}$. For problematic messages, $\mathcal{M}$

```
$\mathsf{Report}(m, tmd, uk, cred, eb)$ $[\mathcal{U}_R \to \mathcal{P}]$
$(ek, lb, sh) \leftarrow \mathsf{TAR.Share}(\delta, tmd)$
$ct_{tmd,3} \leftarrow \mathsf{SPCE.Enc}(pk_R, m, (m, tmd), eb)$
$ct'_{tmd} \leftarrow \mathsf{SKE.Enc}(ek, ct_{tmd,3})$
$dup \leftarrow \mathsf{MAC.Tag}(uk, lb)$
$(cm, \pi_{sw}, rd) \leftarrow \mathsf{KVAC.Show}(cred, uk)$
$\pi_{rpt} \leftarrow \mathsf{PoK.Prv}((lb, dup, cm), (uk, rd))$
/ The relation $\phi_{rpt} = 1$ , iff.:
/ $\mathsf{Com.Vf}(uk, cm, rd) = 1 \land \mathsf{MAC.Vf}(dup, uk, lb) = 1$
$report = (lb, sh, ct'_{tmd}, dup, \pi_{rpt}, cm, \pi_{sw})$
return $report$
```

```
$\mathsf{VfReport}(report, sk_P)$ $[\mathcal{P} \to \mathcal{P}]$
if $\mathsf{KVAC.VfShow}(sk_P, cm, \pi_{sw}) = 0$ :
  return $\perp$
if $\mathsf{PoK.Vf}((lb, dup, cm), \pi_{rpt}) = 0$ :
  return $\perp$
$report' \leftarrow (lb, sh, dup, ct'_{tmd})$
return $report'$
```

```
$\mathsf{Collect}(\{report'_i\}_{i \in [n]})$ $[\mathcal{P} \to \mathcal{R}]$
$\mathcal{L}_{rpt}, \mathcal{L}_{dup} \leftarrow \emptyset$
for $(sh_i, dup_i) \in \{report'_i\}_{i \in [n]}$ :
  if $dup_i \notin \mathcal{L}_{dup}$ :
    $\mathcal{L}_{dup} \xleftarrow{\cup} \{dup_i\}$
    $\mathcal{L}_{rpt} \xleftarrow{\cup} \{sh_i\}$
$ek \leftarrow \mathsf{TAR.Aggregate}(\delta, \mathcal{L}_{rpt})$
if $ek = \perp$ :   return $\perp$
$ct_{tmd,3} \leftarrow \mathsf{SKE.Dec}(ek, ct'_{tmd})$
return $ct_{tmd,3}$
```

```
$\mathsf{Inspect}(ct_{tmd,3}, sk_R)$ $[\mathcal{R} \to \mathcal{M}]$
$(m, tmd) \leftarrow \mathsf{SPCE.Dec}(sk_R, ct_{tmd,3})$
if $\mathsf{Receive}(m, tmd) = 0$ :
  return $\perp$
$ct_{u,2} \leftarrow \mathsf{mkPKE.Dec_3}(sk_R, ct_{u,3})$
$\pi_{d,2} \leftarrow \mathsf{PoK.Prv}((ct_{u,3}, ct_{u,2}, pk_R), sk_R)$
/ $\phi = 1$ , iff.: $ct_{u,2} = \mathsf{mkPKE.Dec_3}(sk_R, ct_{u,3})$
$ct_{tmd,2} = (tmd, ct_{u,2}, \pi_{d,2})$
return $(ct_{tmd,2}, m)$
```

```
$\mathsf{Review}^{\mathsf{Judge}}(ct_{tmd,2}, m, sk_M)$ $[\mathcal{M} \to \mathcal{P}]$
if $\mathsf{Receive}(m, tmd) = 0$
    $\lor\ \mathsf{PoK.Vf}((ct_{u,3}, ct_{u,2}, pk_R), \pi_{d,2}) = 0$
  return $\perp$
if $\mathsf{Judge}(m) = 0$ : return $\perp$
$ct_{u,1} \leftarrow \mathsf{mkPKE.Dec_2}(sk_M, ct_{u,2})$
$\pi_{d,1} \leftarrow \mathsf{PoK.Prv}((ct_{u,2}, ct_{u,1}, pk_M), sk_M)$
/ $\phi = 1$ , iff.: $ct_{u,1} = \mathsf{mkPKE.Dec_2}(sk_M, ct_{u,2})$
$ct_{tmd,1} = (tmd, ct_{u,2}, \pi_{d,2}, ct_{u,1}, \pi_{d,1})$
return $(ct_{tmd,1}, m)$
```

```
$\mathsf{Trace}(ct_{tmd,1}, m, sk_P)$ $[\mathcal{P} \to \mathcal{P}]$
if $\mathsf{Receive}(m, tmd) = 0$
    $\lor\ \mathsf{PoK.Vf}((ct_{u,3}, ct_{u,2}, pk_R), \pi_{d,2}) = 0$
    $\lor\ \mathsf{PoK.Vf}((ct_{u,2}, ct_{u,1}, pk_M), \pi_{d,1}) = 0$
  return $\perp$
$uk \leftarrow \mathsf{mkPKE.Dec_1}(sk_P, ct_{u,1})$
$id \leftarrow \mathcal{L}_{id}[uk]$
return $id$
```

Fig. 3: Algorithms in the report and trace phase. Judge is an oracle that reviews tracked messages' content, returning '1' to indicate a problematic message and '0' for an innocent message.

further decrypts the encrypted identity $ct_{u,2}$ and forwards the report to the platform for tracing; otherwise, $\mathcal{M}$ discontinues the tracing process. The core of Review is the Judge oracle, determining whether the reported message should be traced. In practice, Judge can involve human review [24] or machine-learning-based classification [58]. Additionally, we emphasize that $\mathcal{M}$'s revision preserves minimal information disclosure, as $\mathcal{M}$ only observes messages that meet both recipients' and the regulator's trace rules.

Finally, the platform $\mathcal{P}$ executes Trace to recover the originator's identity. This recovery can succeed only if the encrypted identity $ct_{u,3}$ has been processed by both $\mathcal{R}$ and $\mathcal{M}$, a guarantee provided by the semantic security of mkPKE.

Remark 2 (Blocklist update). Since the encoded blocklist eb is stored in users' storage, the regulator cannot update the blocklist locally. Consequently, to update eb, the regulator incorporates newly appended elements into a fresh blocklist and re-initializes it to generate a new encoded blocklist, which is then broadcast to all users. This updating strategy follows the principle of updatable PSI in [17]. A potential concern with this approach is the unbounded growth of encoded blocklists. Fortunately, in practice, the regulator can renew eb during users' client updates, which allows it to compact multiple blocklists into a single one and remove obsolete elements.

B. Instantiation of Building Blocks

PoK, KVAC, and SPCE. To ensure parameter consistency within tracing metadata, elements generated by different building blocks must conform to the PoK protocol. For efficiency, we instantiate PoK using the generic linear Sigma protocol [10, Chapter 19.5.3] with Fiat-Shamir transformation [23]. Consequently, we instantiate all building blocks under discrete logarithm problems, ensuring that the relationships between ciphertexts can be described through linear relations among group elements. Specifically, our instantiation of KVAC and SPCE follow [12, Section 4.2] and [4, Figure 2], respectively. We recall these constructions in Appendix A.

```
$\mathsf{KGen}(1^\lambda)$
$sk \xleftarrow{\$} \mathbb{Z}_q, pk \leftarrow g^{sk}$
return $(pk, sk)$
```

```
$\mathsf{Enc}(pk_3, pk_2, pk_1, m)$
$r \xleftarrow{\$} \mathbb{Z}_q, E_1 \leftarrow g^r$
$E_2 \leftarrow pk_3^r \cdot pk_2^r \cdot pk_1^r \cdot m$
return $ct_3 = (E_1, E_2)$
```

```
$\mathsf{Dec_3}(sk_3, ct_3)$
$E'_2 \leftarrow E_2 / E_1^{sk_3}$
$ct_2 = (E_1, E'_2)$
return $ct_2$
```

```
$\mathsf{Dec_2}(sk_2, ct_2)$
$E''_2 \leftarrow E'_2 / E_1^{sk_2}$
$ct_1 = (E_1, E''_2)$
return $ct_1$
```

```
$\mathsf{Dec_1}(sk_1, ct_1)$
$m \leftarrow E''_2 / E_1^{sk_1}$
return $m$
```

Fig. 4: Instantiation of multi-key PKE.

Multi-key PKE. We instantiate a mkPKE scheme by slightly modifying the ElGamal encryption [20], presented in Fig. 4. Although we only consider three keys in the construction, it can be easily extended to multiple keys. The semantic security of the leveled ciphertexts (e.g., ct_2, ct_1) can be directly reduced

```
$\mathsf{TAR.Share}(\delta, S)$
$r \leftarrow H_1(S||1)$
$ek \leftarrow H_1(S||2)$
$lb \leftarrow H_1(S||3)$
$sh \leftarrow\!\$\ \mathsf{SS.Share}(ek, \delta; r)$
$\mathbf{return}\ (ek, lb, sh)$

$\mathsf{TAR.Aggregate}(\delta, \{sh_i\}_{i\in[n]})$
$\mathbf{if}\ n < \delta : \mathbf{return}\ \perp$
$ek \leftarrow \mathsf{SS.Recover}(\{sh_i\}_{i\in[n]})$
$\mathbf{return}\ ek$
```

```
$\mathsf{MAC.KGen}(1^\lambda)$
$\mathbf{return}\ k \leftarrow\!\$\ \mathbb{Z}_q$

$\mathsf{MAC.Tag}(k, m)$
$\mathbf{return}\ tag \leftarrow H_2(m)^k$

$\mathsf{MAC.Vf}(k, m, tag)$
$tag' \leftarrow H_2(m)^k$
$\mathbf{if}\ tag' \neq tag :$
  $\mathbf{return}\ 0$
$\mathbf{return}\ 1$
```

Fig. 5: Instantiation of TAR and MAC. SS = (Share, Recover) is a secret sharing protocol with reproducibility [7]. The hash function $H_1 : \{0,1\}^* \to \mathbb{Z}_p$ is modeled as random oracle; $H_2 : \{0,1\}^* \to \mathbb{G}$ is cryptographic secure.

to the semantic security of standard ElGamal encryption.

Threshold aggregation reporting. We instantiate a TAR protocol in Fig. 5, which follows the secret sharing-based construction in [15]. In Share, a client first deterministically derives a randomness r, a symmetric key ek, and a label lb for identifying the secret. The client then uses the underlying secret sharing protocol, SS.Share, to generate a random δ-out-of-n share sh of ek. Upon receiving sufficient shares from the clients, the server aggregates them to reconstruct the key ek.

To avoid share collisions, the prime field for generating secret shares sh must be sufficiently large. For Shamir's secret sharing, the deterministically-derived randomness r ensures that all clients holding the same secret generate the same set of polynomial coefficients. Each client then creates its unique share by evaluating this polynomial at a randomly sampled x-coordinate within the field. Our implementation uses Curve25519, which operates over a 255-bit prime field. This makes the probability of a share collision—wherein two clients independently sample the same x-coordinate—negligible.

Message authentication code. We instantiate the MAC in Fig. 5, which is realized by a weak pseudorandom function (wPRF) [13], [40]. The wPRF output $H(m)^k$ satisfies uniqueness, weak pseudorandomness, and unforgeability.

C. Design Choices

Deniability vs. Unframeability. Deniability allows a user to deny sending or receiving a specific message to a third party, thus protecting users' privacy under mass surveillance. In contrast, unframeability ensures that no participant, not even a malicious platform, can originate a message under another user's identity. Our design can achieve either deniability or unframeability, depending on who generates the users' secret keys. When the platform generates the users' keys, it can forge tracing metadata corresponding to any message and user's identity, thereby achieving platform deniability. Conversely, when using blind issuance of KVAC to issue credentials to the users, they can generate their secret keys. In this case, only the user themselves can generate tracing metadata under their key. The choice between deniability and unframeability can be tailored to real-world scenarios.

Key-verification vs. Publicly verifiable. To accommodate the Sigma protocol, we instantiate the anonymous credential (AC) using KVAC. An alternative approach is using a publicly verifiable AC protocol, such as pairing-based credentials [5]. In this case, all cryptographic tools should be built using structure-preserving cryptography [2] compatible with the pairing-based proof system [26]. Although this alternative would incur higher costs than the instantiation without pairing, it offers two benefits: First, it eliminates the need for TkGen, which is introduced to transform a KVAC into a publicly verifiable AC. Second, it allows recipients to re-randomize received tracing metadata, thereby achieving unlinkability.

Linkability vs. Unlinkability. Linkability allows a user who receives two identical forwarded messages to determine whether they originated from the same user. In contrast, unlinkability severs the connection between messages from the same originator. As discussed above, our construction can support unlinkability by selecting appropriate underlying building blocks, albeit at the cost of some efficiency. We leave a more efficient unlinkable construction as future work.

Report count confidentiality vs. Efficiency. In our design, the platform can observe the report counts associated with each deterministically generated label from the tracing metadata. Crucially, the platform can neither link a given label to the message content nor to the originators' or reporters' identities. Even in a collusion scenario where a recipient helps the platform link a label to their received message, our reporting anonymity ensures the identities of all other reporters remain protected. Furthermore, this leakage does not undermine our abuse resistance; the originator's identity remains secured by the three-layer mkPKE ciphertext. Additionally, for systems requiring stricter privacy, this leakage could be suppressed by using a leakage-free TAR protocol [35], albeit at a higher cost.

VI. Security Analysis

In this section, we formally define and prove the security of our design. Due to the page limits, some security definitions and all formal proofs are deferred to Appendix B.

Before detailing the security goals, in Fig. 6, we introduce two oracles that will be utilized in multiple security games. Considering the possibility of servers colluding with users, we categorize the users into two sets $\mathcal{U}_m$ and $\mathcal{U}_h$ that represent users who collude and do not collude with the adversary, respectively. First, $\mathcal{O}_{\mathsf{cred}}$ allows the adversary to register any user's identity in the system but only obtain the key and credential of colluding users $\mathcal{U}_m$. Second, $\mathcal{O}_{\mathsf{token}}$ enables the adversary to derive anonymous tokens from their credentials.

A. Confidentiality

The confidentiality ensures that the content and the originator's identity of a message remain hidden before tracing.

Source confidentiality. Source confidentiality dictates that tracing metadata reveals nothing about its originator's identity. To capture this goal, we define the game $\mathrm{SConf}_{ST}^{\mathcal{A},b}$, shown

```
$\mathcal{O}_{\mathsf{cred}}(id, isMal)$
if $id \in (\mathcal{U}_h \cup \mathcal{U}_m)$ : return $\perp$
$(uk, cred) \leftarrow \mathsf{UReg}(id, sk_P)$
$\mathcal{L}_k[id] \leftarrow (uk, cred)$
if $isMal = 1 : \mathcal{U}_m \overset{\cup}{\leftarrow} \{id\}$
    return $(id, uk, cred)$
else : $\mathcal{U}_h \overset{\cup}{\leftarrow} \{id\}$
    return $id$

$\mathcal{O}_{\mathsf{token}}(uk, cred)$
$tk \leftarrow \mathsf{TkGen}(uk, cred, sk_P)$
if $tk = \perp$ : return $\perp$
$\mathcal{L}_{tk} \overset{\cup}{\leftarrow} \{(cred, tk)\}$
return $tk$
```

Fig. 6: The oracles for simulating the user registration phase. The platform's secret key sk_P, users' secret key $\mathcal{L}_k$ and tokens $\mathcal{L}_{tk}$ are maintained by the challenger.

in Fig. 7. In this game, an adversary acts as a malicious recipient who can collude with any two servers. The adversary is allowed to query $\mathcal{O}_{\mathsf{corrupt}}$ to obtain the secret keys of two out of three servers. Furthermore, if the adversary chooses to corrupt the platform, they also obtain $\mathcal{L}_k$, which stores all users' identities and credentials. The adversary's task is to distinguish tracing metadata generated by different originators when querying $\mathcal{O}_{\mathsf{send},b}$.

Definition 1 (Source Confidentiality). *We say a source tracing scheme ST satisfies source confidentiality if, for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in winning $SConf^{\mathcal{A},b}_{ST}$ is negligible, where the advantage is defined as follows:*

$$\mathsf{Adv}^{SConf}_{ST}(\mathcal{A}) = \left|\Pr\left[SConf^{\mathcal{A},1}_{ST} = 1\right] - \Pr\left[SConf^{\mathcal{A},0}_{ST} = 1\right]\right|.$$

Theorem 1. *Assuming the PoK protocol, mkPKE scheme, and KVAC protocol satisfy zero-knowledge property, semantic security, and anonymity, respectively, then the construction in Section V-A satisfies source confidentiality.*

Intuitively, all components in the tracing metadata (tmd) are indistinguishable from random bit strings, as they satisfy either the indistinguishability or zero-knowledge property. First, the anonymous token ($token$) comprises a commitment and the platform's signature. This signature is generated from an anonymous credential presentation that reveals nothing about the originator's identity. Thus, the token reveals nothing to the adversary, even if $\mathcal{A}$ colludes with the platform. Second, the ciphertexts $(ct_{u,3}, ct_{tag})$ are encrypted using the servers' public keys. Since $\mathcal{A}$ can only acquire two of the three servers' secret keys, mkPKE's semantic security guarantees that these ciphertexts are indistinguishable from random bit strings. Finally, the zero-knowledge property of the proof (π_{send}) ensures no information is leaked. For a formal proof, please refer to Appendix B-A.

Message confidentiality. Message confidentiality ensures that the tracing metadata or message reports disclose nothing about the content of innocent messages to participants other than the direct recipient. In this context, we do not consider collusion between the recipient and servers, as a malicious recipient can always relay the message to a colluding server.

```
$\mathrm{SConf}^{\mathcal{A},b}_{ST}(1^\lambda)$
$\mathcal{L}_k, \mathcal{L}_{tk}, \mathcal{U}_h, \mathcal{U}_m \leftarrow \emptyset$
$(sk_R, sk_M,$
        $sk_P, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
/ Choose a pair of key indices.
$\{j, k\} \leftarrow \mathcal{A}(pk)$
if $\{j, k\} \not\subseteq \{R, M, P\}$ :
    return $\perp$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^*_b}(pk)$
return $b'$

$\mathcal{O}_{\mathsf{send},b}(id_0, id_1, m)$
if $\exists id \in \{id_0, id_1\}$,
    $id \notin \mathcal{U}_h$ : return $\perp$
$(uk_b, cred_b) \leftarrow \mathcal{L}_k[id_b]$
$tk_b \leftarrow \mathsf{TkGen}(uk_b, cred_b, sk_P)$
$tmd_b \leftarrow \mathsf{Send}(m, uk_b, tk_b)$
return $tmd_b$

$\mathcal{O}_{\mathsf{corrupt}}(i)$
if $i \notin \{j, k\}$ : return $\perp$
if $i = P$ : return $(sk_P, \mathcal{L}_k)$
else : return $sk_i$
```

Fig. 7: Security game of source confidentiality. Here, pk represents the public keys of servers (pk_R, pk_M, pk_P). $\mathcal{O}^*_b$ denotes $\mathcal{O}_{\mathsf{cred}}$, $\mathcal{O}_{\mathsf{token}}$, $\mathcal{O}_{\mathsf{send},b}$, and $\mathcal{O}_{\mathsf{corrupt}}$.

The message confidentiality of our design is straightforward. First, before reporting, both the message content and tracing metadata are transmitted via the underlying E2EE. This ensures that only the communication parties can access the plaintext; all other participants observe only end-to-end encrypted ciphertexts. Second, during the report phase, the privacy provided by TAR ensures the platform learns nothing of the message content from user reports. Furthermore, even if the platform collects enough shares, it only reconstructs a SPCE ciphertext of (m, tmd), which inherently preserves message confidentiality due to its client privacy.

Blocklist confidentiality. Blocklist confidentiality ensures that the encoded blocklist reveals nothing about its content to anyone other than the regulator. In our design, this property directly stems from SPCE's set-hiding property [4], guaranteeing its master public key (our encoded blocklist) reveals no information about the blocklist content $\mathcal{B}$. This property is defined as server security in Apple PSI [39, Section 5.1].

B. Accountability

The accountability prevents malicious users from manipulating tracing results or submitting duplicate reports, while ensuring only the regulator can modify the encoded blocklist.

Trace accountability. Trace accountability ensures that an adversary cannot evade tracing or manipulate tracing results. This property encompasses the goals of sender binding and trace unforgeability from prior work [41], guaranteeing that a message can be traced back to its actual originator.

We formally define this property in a game $\mathrm{Act}^{\mathcal{A}}_{ST}$, depicted in Fig. 8. In this game, the adversary's task is to create a valid tuple of messages and tracing metadata that either cannot be traced ($id^* = \perp$) or will be traced to the wrong originator ($id^* \in \mathcal{U}_h \wedge (id^*, m^*) \notin \mathcal{L}_q$). To prevent trivial wins, we require that the misidentified user be honest, as the adversary can always generate unrecorded tracing metadata using the secret keys of malicious users $\mathcal{U}_m$. Furthermore, we allow the adversary to obtain any user's tracing metadata for any message. In particular, for malicious users $\mathcal{U}_m$, the adversary

```
$\mathrm{Act}^{\mathcal{A}}_{ST}(1^\lambda)$
$\mathcal{L}_q, \mathcal{L}_k, \mathcal{L}_{tk}, \mathcal{U}_h, \mathcal{U}_m \leftarrow \emptyset$
$(sk_R, sk_M, sk_P, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(ct^*_{tmd,1}, m^*) \leftarrow \mathcal{A}^{\mathcal{O}^*}(sk_R, sk_M, pk)$
$id^* \leftarrow \mathsf{Trace}(ct^*_{tmd,1}, m^*, sk_P)$
$\textbf{if } id^* = \perp : \textbf{return } 1$
$\textbf{if } id^* \in \mathcal{U}_h \wedge (id^*, m^*) \notin \mathcal{L}_q :$
  $\textbf{return } 1$
$\textbf{return } 0$
```

```
$\mathcal{O}_{\mathsf{honSend}}(id, m)$
$\textbf{if } id \notin \mathcal{U}_h : \textbf{return } \perp$
$(uk, tk) \leftarrow \mathcal{L}_k[id]$
$tmd \leftarrow \mathsf{Send}(m, uk, tk)$
$\mathcal{L}_q \overset{\cup}{\leftarrow} \{(id, m)\}$
$\textbf{return } tmd$
```

Fig. 8: Security game of trace accountability, where $\mathcal{O}^*$ denotes $\mathcal{O}_{\mathsf{cred}}$, $\mathcal{O}_{\mathsf{token}}$, and $\mathcal{O}_{\mathsf{honSend}}$.

can obtain their secret key and credentials, enabling it to generate tracing metadata by executing the Send algorithm. For honest users $\mathcal{U}_h$, the adversary can obtain their tracing metadata by querying the oracle $\mathcal{O}_{\mathsf{honSend}}$.

Without loss of generality, $\mathrm{Act}^{\mathcal{A}}_{ST}$ focuses on the tracing metadata's accountability and omits the intermediate steps (i.e., Collect, Inspect, and Review) for enforcing trace rules. Since these steps only affect the reconstruction of the tracing metadata, even if an adversary can manipulate these processes, it still needs to forge tracing metadata to break trace unforgeability. Therefore, this omission does not affect our analysis.

Definition 2 (Trace accountability). *We say a source tracing scheme ST satisfies trace accountability if for any PPT adversary $\mathcal{A}$, $\mathcal{A}$ has a negligible advantage in winning the game $Act^{\mathcal{A}}_{ST}$, where the advantage is defined as follows:*

$$\mathsf{Adv}^{Act}_{ST}(\mathcal{A}) = \left|\Pr\left[Act^{\mathcal{A}}_{ST} = 1\right]\right| .$$

Theorem 2. *Assuming the PoK protocol satisfies soundness; the KVAC protocol, DS scheme, and MAC scheme satisfy unforgeability, then the construction in Section V-A satisfies trace accountability.*

Informally, we can reduce trace accountability to the unforgeability of the components within the tracing metadata. In detail, the anonymous token incorporates a commitment and a platform-generated signature. The signature ensures that the adversary cannot forge a valid token not issued by the platform. Furthermore, the MAC binds the originator's identity to the message. Moreover, the consistent proof guarantees the originator's identity remains consistent across both the anonymous token and the MAC. Upon receiving or tracing a message, an honest recipient or the platform will verify the validity of the signature and consistent proof. These verifications ensure that the tracing metadata is well-formed, thereby mitigating the risk of a malicious sender using malformed tracing metadata to evade tracing or falsely implicate honest users. We defer the formal proof to Appendix B-B.

Report uniqueness. Report uniqueness ensures that a user cannot generate duplicate reports of the same message, even if colluding with the moderator and regulator. For our construction, this property primarily relies on the unique deduplication tag generated by a *deterministic* MAC. Specifically, the deduplication tag is generated from the reported message and the reporter's key. Consequently, two identical tags would appear if a user reports the same message twice. To prevent a malicious reporter from submitting a malformed tag, we require the reporter to submit an anonymous credential and consistent proof, verifying their identity to the moderator. A formal definition and proof can be found in Appendix B-C.

Blocklist authenticity. Blocklist authenticity safeguards the integrity of the encoded blocklist against a malicious moderator or platform. It ensures that the problematic messages embedded within the encoded blocklist are maintained by the regulator. In our construction, this property can be reduced to the unforgeability of the signature, which is signed using the regulator's secret key.

C. Anonymity

Our tracing scheme's anonymity is independent of the underlying EEMS; it's designed not to undermine the underlying EEMS's native anonymity. Specifically, our anonymity definitions ensure the algorithms' output (e.g., tracing metadata or reports) leaks no information about user identities. Consequently, the final integrated system inherits the anonymity level of the underlying EEMS.

Messaging anonymity. Message anonymity ensures that the identities of both the sender and the recipient remain concealed from all other participants, encompassing both sender and recipient anonymity. First, source confidentiality safeguards the originator's identity, ensuring anonymity when the sender is the message originator. Second, the sender and recipient's identities are not required in any part of our algorithms, making their confidentiality inherent. Thus, our construction preserves message anonymity.

Reporting anonymity. Reporting anonymity mandates that a message report does not disclose the reporter's identity. For our construction, we can reduce this property to the underlying building blocks' confidentiality. Due to KVAC's anonymity and PoK's zero-knowledge property, the credential presentation (cm, π_{sw}) and proof of report π_{rpt} leak nothing about the reporter's identity. The only potential source of identity leakage is the deduplication tag used to ensure report uniqueness. Given the MAC's weak pseudorandomness [18], this tag remains secure when the input tracing metadata has sufficient entropy. Crucially, we prove that the tracing metadata appears random to the platform in source confidentiality. As a result, the platform cannot ascertain the reporter's identity from any report. Moreover, this property holds even if a user submits the same message twice, as the platform can only link the two reports without inferring the reporter's identity. A formal definition and proof can be found in Appendix B-D.

VII. Implementation and Evaluation

To evaluate the performance of our design, we implement it using Rust [29] and conduct experiments on two PCs. For server-side benchmarking, we utilized a PC with an Intel Xeon 8375C CPU @ 2.90GHz and 192 GB RAM. For user-side

TABLE II: Bandwidth and latency comparison.

Schemes	Bandwidth			Latency			
	Send	Receive	Report	Send	Process	Receive	Overall
Peale et al.† [41]	256B	320B	160B	42.6 - 39.0 μs	19.4 μs	81.7 - 160.1 μs	143.7 - 218.5 μs
Issa et al. [28]	380B	484B	380B	32.5 μs	22.5 μs	161.4 μs	216.4 μs
Bartusek et al. [4]	2304B	2304B	2304B	2528.5 μs	-	10973.5 μs	13502 μs
Ours	614B	614B	1708B	954.2 μs	-	300.1 μs	1254 μs

Peale et al. (the linkable scheme) [41] and Issa et al. [28] represent the most efficient schemes for non-anonymous and anonymous EEMS, respectively. Bartusek et al. [4] is the only tracing scheme incorporating blocklisting.

$\dagger$: The runtime of PEB21 varies between an originated message (left) and a forwarded (right) message.

TABLE III: Runtime of our algorithms.

LGen($\lvert\mathcal{B}\rvert = \{10^5, 10^6, 10^7\}$)	UReg	TkGen	Report	VfReport	Collect($\delta = \{20, 40, 60\}$)	Inspect	Review	Trace
{1.92, 21.4, 284.9} s	697.2 μs	757.8 μs	1491 μs	415.3 μs	{5.88, 24.6, 53.1} ms	759.1 μs	621.0 μs	567.5 μs

benchmarking, we employed a PC with an Intel Core i5-9600K CPU @ 4.60GHz and 32 GB RAM.

A. Implementation Details

Our implementation utilizes the platform-agnostic APIs provided by libsignal [47], with all group operations performed using Curve25519 [14]. Moreover, the instantiations of KVAC and PoK are directly invoked through the libsignal API, as these tools are integral to Signal's private group system [13]. Additionally, we instantiate the SKE, DS, and PKE using AES-GCM-256, ECDSA, and Hybrid PKE [43], respectively. Furthermore, the TAR scheme is built on Shamir secret sharing. The hash function that maps an arbitrary-length bit string to a fixed-length bit string is instantiated by SHA-512.

For the experimental setting, we set the user's secret key to 32 bytes, equivalent to the size of a group element. The $|\mathcal{B}|$ elements in the blocklist are assigned to $1.2|\mathcal{B}|$ items in the Cuckoo hashing table. The message size is set to 1 kB in all the experiments to align with prior work [41].

B. Performance Evaluation

Here, we evaluate all our algorithms' performance and present the following results.

Comparison with prior work. We begin by comparing our scheme with the current state-of-the-art (SOTA) in source tracing. Notably, we exclude threshold reporting schemes from the comparison due to their primary overhead occurring outside the message phase. For instance, the primary overhead of Liu et al. [37] is maintaining a collaborative counting bloom filter on the platform; Bell et al. [6] is built on black-box invocations for source tracing; thus, its performance in messaging depends on the underlying tracing schemes.

Table II presents our comparison results in bandwidth and latency overhead. Overall, due to the employment of trace rules, our scheme requires slightly more overhead compared to the SOTA schemes; nevertheless, it remains sufficient for practical deployment. When sending a 1 kB message from a user to another user, our scheme introduces additional bandwidth and latency overheads of less than 1.7 kB and 1.3 ms, respectively. Furthermore, our system incurs no platform-side operations when sending a message. Considering the daily volume of messages on WhatsApp (i.e., 100 billion [48]), this can significantly reduce the platform-side complexity and overhead of deployment.

Runtime. We then present the runtime of our algorithms in Table III. All algorithms have runtimes of less than 1.5 ms, except for Collect and LGen. The only algorithm potentially impacting messaging latency is TkGen. While it requires users to request anonymous tokens before messaging, this can be done during off-peak times, limiting its impact. Furthermore, the algorithms involved in the report phase (Report, VfReport, Collect) and the trace phase (Inspect, Review, Trace) are executed infrequently and are therefore not latency-sensitive.

We also measure Collect and LGen across various thresholds and blocklist sizes, respectively. Our measurements indicate that the efficiency of these two algorithms is sufficient for practical deployment, even in the worst cases. For instance, the regulator can encode a blocklist containing 10 million messages within 5 minutes.

Storage. We next present the storage overheads of our scheme. On the user side, a client needs to store a secret key (32B), an anonymous credential (64B), and multiple anonymous tokens (134B). For a user sending 10,000 messages daily, these tokens require only 1.3 MB of storage. Additionally, clients should store the encoded blocklist for blocklisting, where a blocklist containing 10 million messages requires 320 MB of storage. Importantly, since the blocklist is re-initialized during OS software updates (typically occurring once every two months), downloading these keys does not impact the user experience. On the server side, the moderator collects refined reports (1420B) after verification. For one million such reports, this incurs an overhead of only 1.35 GB. It is reasonable to assume that the system processes reports within a time window (e.g., one month), thereby preventing indefinite storage growth.

End-to-end prototype. We finally test the end-to-end messag-

TABLE IV: End-to-end runtime (ms).

Runtime	Signal	Signal & Ours	Difference
Sending	3.03	4.84	1.81
Receiving	0.84	1.13	0.29
Total	3.87	5.97	2.10
E2E latency†	77.91	80.37	2.46

†: The end-to-end latency encompasses network latency and algorithm execution times.

ing overhead by integrating the Send and Receive components into the signal-cli [46] and libsignal [47]. All of our evaluations rely on Signal's Sealed Sender feature [30] for sender anonymity. As discussed in Section VI-C, the use of this one-sided anonymous EEMS does not compromise our protocol's security guarantees.

Table IV presents a comparison of messaging latency between the original Signal and Signal with our scheme. For this evaluation, we deployed the client on a virtual machine (2-core CPU, 8GB RAM, 32GB storage) hosted on GitHub Codespaces. The clients communicated directly via the official Signal server. The results indicate that the inclusion of our scheme introduces only a minor overhead (2.46 ms) to the latency between users.

VIII. Conclusion

This paper introduced the goal of achieving abuse-resistant traceability with minimal trust and information disclosure for EEMSs. Aiming at this goal, we introduced a novel system model wherein each participant is a real-world entity belonging to different trust domains. Under the model, we propose a source tracing scheme in which each participant enforces distinct trace rules for reported messages. Implementation and evaluation show that our scheme requires only 1.3 ms of latency and 1.7 kB of bandwidth per message, making it practical for real-world deployment.

Acknowledgments

The authors thank anonymous reviewers for their constructive reviews. This work was supported by the National Natural Science Foundation of China under Grants 62572079, 62472056, 62502056, and 62572083, the New Chongqing YC Project under Grant CSTB2025YCJH-KYXM0068.

Ethics Considerations

This work does not present direct ethical concerns, as it does not include any personal data. The evaluations conducted are simulations of clients and servers and do not involve interaction with real users. However, the real-world deployment of this work presents an ethical challenge. While our goal is to curb abuse, any traceability mechanism inherently compromises user privacy by creating a capability to reveal an originator's identity. This capability, even when narrowly targeted, creates a privacy risk for all users. Consequently, we argue that a technical solution alone is insufficient for responsible deployment. Any implementation must be governed by robust non-technical safeguards, including clear legal frameworks, independent oversight, and rigorous adherence to human rights principles.

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In *TCC*, 2010.

[2] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *JoC*, 2016.

[3] Apple. About icloud private relay. https://support.apple.com/102602, Aug. 2023.

[4] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. End-to-end secure messaging with traceability only for illegal content. In *EUROCRYPT*, 2023.

[5] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO*, 2009.

[6] Connor Bell and Saba Eskandarian. Anonymous complaint aggregation for secure messaging. In *PETS*, 2024.

[7] Mihir Bellare, Wei Dai, and Phillip Rogaway. Reimagining secret sharing: Creating a safer and more versatile primitive by adding authenticity, correcting errors, and reducing randomness requirements. *PETS*, 2020.

[8] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kunal Talwar, and Karl Tarbe. The apple psi system. https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf, July 2021.

[9] Oversight Board. How we do our work. https://www.oversightboard.com/our-work/, 2024.

[10] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

[11] Pedro Branco, Matthew Green, Aditya Hegde, Abhishek Jain, and Gabriel Kaptchuk. How to trace viral content in end-to-end encrypted messaging. Cryptology ePrint Archive, Paper 2025/1052, 2025.

[12] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In *ACM CCS*, 2014.

[13] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal private group system and anonymous credentials supporting efficient verifiable encryption. In *ACM CCS*, 2020.

[14] Dalek-crypto. curve25519-dalek. https://github.com/dalek-cryptography/curve25519-dalek, 2020.

[15] Alex Davidson, Peter Snyder, EB Quirk, Joseph Genereux, Benjamin Livshits, and Hamed Haddadi. Star: Secret sharing for private threshold aggregation reporting. In *ACM CCS*, 2022.

[16] Patricia Davis. Helping child survivors: The fight to remove sex abuse images. https://www.missingkids.org/blog/2024/helping-child-survivors-fight-to-remove-sex-abuse-images/, 2024.

[17] Samuel Dittmer, Yuval Ishai, Steve Lu, Rafail Ostrovsky, Mohamed Elsabagh, Nikolaos Kiourtis, Brian Schulte, and Angelos Stavrou. Streaming and unbalanced psi from function secret sharing. In *SCN*, 2022.

[18] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message authentication, revisited. In *EUROCRYPT*, 2012.

[19] John R Douceur. The sybil attack. In *IPTPS*, 2002.

[20] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. 1985.

[21] Saba Eskandarian. Abuse Reporting for Metadata-Hiding Communication Based on Secret Sharing. In *USENIX Security*, 2024.

[22] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *ToSC*, 2017.

[23] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1987.

[24] Google. Content safety api. https://protectingchildren.google/tools-for-partners/, 2024.

[25] Matthew Green and Alex Stamos. Apple wants to protect children. but it's creating serious privacy risks. https://www.nytimes.com/2021/08/11/opinion/apple-iphones-privacy.html, Aug. 2021.

[26] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.

[27] Jasmine Henry. What is crypto-agility? https://www.cryptomathic.com/blog/what-is-crypto-agility, Aug. 10 2018.

[28] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. Hecate: Abuse reporting in secure messengers with sealed sender. In *USENIX Security*, 2022.

[29] Ze Jiang. trace-ruler. https://github.com/Ming-bc/trace-ruler.

[30] jlund. Technology preview: Sealed sender for signal. https://signal.org/blog/sealed-sender/, Oct. 2018.

[31] Gabriel Kaptchuk and Ran Canetti. The broken promise of apple's announced forbidden-photo reporting system - and how to fix it. https://www.bu.edu/riscs/2021/08/10/apple-csam/, Aug. 2021.

[32] Darya Kaviani, Sijun Tan, Pravein Govindan Kannan, and Raluca Ada Popa. Flock: A Framework for Deploying On-Demand Distributed Trust. In *OSDI*, 2024.

[33] Erin Kenney, Qiang Tang, and Chase Wu. Anonymous traceback for end-to-end encryption. In *ESORICS*, 2022.

[34] Richard Lawler. Apple drops controversial plans for child sexual abuse imagery scanning. https://www.theverge.com/2022/12/7/23498588/apple-csam-icloud-photos-scanning-encryption, Dec. 2022.

[35] Hanjun Li, Sela Navot, and Stefano Tessaro. POPSTAR: Lightweight Threshold Reporting with Reduced Leakage. In *USENIX Security*, 2024.

[36] Rui Lian, Yulong Ming, Chengjun Cai, Yifeng Zheng, Cong Wang, and Xiaohua Jia. Nemesis: Combating Abusive Information in Encrypted Messaging with Private Reporting. In *ESORICS*, 2024.

[37] Linsheng Liu, Daniel S Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). In *NDSS*, 2022.

[38] Alexis Madrigal. India's lynching epidemic and the problem with blaming tech. https://www.theatlantic.com/technology/archive/2018/09/whatsapp/571276/, 2018.

[39] Mihir Bellare. A concrete-security analysis of the apple psi protocol. https://www.apple.com/child-safety/pdf/Alternative_Security_Proof_of_Apple_PSI_System_Mihir_Bellare.pdf, July 2021.

[40] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In *EUROCRYPT*, 1999.

[41] Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. In *ACM CCS*, 2021.

[42] Katitza Rodriguez and seth schoen. Faq: Why brazil's plan to mandate traceability in private messaging apps will break user's expectation of privacy and security. https://www.eff.org/deeplinks/2020/08/faq-why-brazils-plan-mandate-traceability-private-messaging-apps-will-break-users, Aug. 2020.

[43] Michael Rosenberg. rust-hpke. https://github.com/rozbb/rust-hpke, 2020.

[44] Sarah Scheffler, Anunay Kulshrestha, and Jonathan Mayer. Public verification for private hash matching. In *IEEE S&P*, 2023.

[45] Sarah Scheffler and Jonathan Mayer. Sok: Content moderation for end-to-end encryption. In *PETS*, 2023.

[46] Sebastian Scheibner. signal-cli. https://github.com/AsamK/signal-cli.

[47] Signal. libsignal. https://github.com/signalapp/libsignal/tree/main.

[48] Manish Singh. Whatsapp is now delivering roughly 100 billion messages a day. https://techcrunch.com/2020/10/29/whatsapp-is-now-delivering-roughly-100-billion-messages-a-day/, Oct. 2020.

[49] Google Australia Team. How we detect, remove and report child sexual abuse material. https://blog.google/intl/en-au/company-news/outreach-initiatives/how-we-detect-remove-and-report-child-sexual-abuse-material/, Feb. 2023.

[50] Dhanaraj Thakur, Guest Post, Mallory Knodel, Emma Llansó, Greg Nojeim, and Caitlin Vogus. Outside looking in: Approaches to content moderation in end-to-end encrypted systems. https://cdt.org/insights/outside-looking-in-approaches-to-content-moderation-in-end-to-end-encrypted-systems/, Aug. 2021.

[51] Kurt Thomas, Devdatta Akhawe, Michael Bailey, Dan Boneh, Elie Bursztein, Sunny Consolvo, Nicola Dell, Zakir Durumeric, Patrick Gage Kelley, Deepak Kumar, Damon McCoy, Sarah Meiklejohn, Thomas Ristenpart, and Gianluca Stringhini. SoK: Hate, harassment, and the changing landscape of online abuse. In *IEEE S&P*, 2021.

[52] Martin Thomson and Christopher A. Wood. Oblivious HTTP. Internet-Draft draft-ietf-ohai-ohttp-10, Internet Engineering Task Force, August 2023. Work in Progress.

[53] Thorn. How hashing and matching can help prevent revictimization. https://www.thorn.org/blog/hashing-detect-child-sex-abuse-imagery/, 2023.

[54] Twitter. Our approach to the 2022 us midterms. https://blog.x.com/en_us/topics/company/2022/-our-approach-to-the-2022-us-midterms, 2022.

[55] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In *ACM CCS*, 2019.

[56] Zhongming Wang, Tao Xiang, Xiaoguo Li, Biwen Chen, Guomin Yang, Chuan Ma, and Robert H. Deng. Impact Tracing: Identifying the Culprit of Misinformation in Encrypted Messaging Systems. In *NDSS*, 2025.

[57] WhatsApp. FAQ: What is traceability and why does whatsapp oppose it? https://faq.whatsapp.com/1206094619954598/?helpref=uf_share, May. 2021.

[58] WhatsApp. Whatsapp messaging guidelines. https://www.whatsapp.com/legal/messaging-guidelines, Feb. 2024.

APPENDIX A
PRELIMINARIES

Proof of Knowledge (PoK). We use the Sigma protocol for arbitrary linear relations [10, Chapter 19.5.3] to instantiate the proofs. Specifically, for a linear relation

$$\phi(\mathbf{y}, \mathbf{x}) = \left\{ y_1 = \prod_{j=1}^{n} g_{1j}^{x_j} \wedge \ldots \wedge y_m = \prod_{j=1}^{n} g_{mj}^{x_j} \right\},$$

- $\mathsf{Prv}(\mathsf{y}, \mathsf{x})$: A prover generates a proof as follows:
  1) Compute commitments $C_i \leftarrow \prod_{j=1}^{n} g_{ij}^{r_j}$, where $r_j \leftarrow\$ \mathbb{Z}_q$, $j \in [n]$, and $i \in [m]$.
  2) Compute a challenge as $c = H(\{C_i\}_{i \in [m]})$.
  3) Compute responses $s_j \leftarrow r_j + x_j c$, where $j \in [n]$.
  4) Send the proof $\pi = (\{C_i\}_{i \in [m]}, \{s_j\}_{j \in [n]})$ to a verifier.
- $\mathsf{Vf}(\mathsf{y}, \pi)$: The verifier accepts the proof π if the equation $\prod_{j=1}^{n} g_{ij}^{s_j} = C_j \cdot y_i^c$ holds for $i \in [m]$.

Keyed-Verification Anonymous Credential (KVAC). We utilize the KVAC protocol as introduced in [12, Section 4.2]. For simplicity, we assume that each client is represented by a single identity rather than a set of attributes.

- $\mathsf{KGen}(1^\lambda) \rightarrow (pp, k_S)$: Generate a group $(\mathbb{G}, p, g)$, an element $h \leftarrow \mathbb{G}$, MAC's secret keys $x_0, x_1 \leftarrow\$ \mathbb{Z}_q$, and server parameters k_S as $X_0 = h^{x_0}, X_1 = h^{x_1}$. Commit the secret key x_0 as $C_{x_0} = g^{x_0} h^{\hat{x_0}}$, where $\hat{x_0} \leftarrow\$ \mathbb{Z}_q$. It finally outputs:

$$pp = (\mathbb{G}, p, g, h, X_0, X_1, C_{x_0}), \quad k_S = (x_0, x_1, \hat{x_0}).$$

- $\mathsf{Issue}\langle \mathcal{C}(id), \mathcal{S}(k_S) \rangle \rightarrow \mathcal{C}(cred, \pi_{iss})$: To issue a credential with an identity $id \in \mathbb{Z}_q$, the server computes $u' = u^{x_0 + x_1 id}$, where $u \leftarrow\$ \mathbb{G}$. Then, the server sends the credential $cred = (u, u')$ and a proof π_{iss} as follows:

$$\begin{aligned} \pi_{iss} = &\mathsf{PoK.Prv}\{(x_0, x_1, \hat{x_0}, id) : \\ &u' = u^{x_0 + x_1 id} \wedge C_{x_0} = g^{x_0} h^{\hat{x_0}} \wedge X_1 = h^{x_1}\}. \end{aligned}$$

- $\mathsf{VfIssue}(id, cred, \pi_{iss}) \rightarrow \{0, 1\}$: Upon receiving a credential and corresponding proof, a client verifies the proof with her id and public parameters using $\mathsf{PoK.Vf}$.
- $\mathsf{BlindIssue}\langle \mathcal{C}(E_{id}), \mathcal{S}(k_S) \rangle \rightarrow (cred, \pi_{iss})$: To blindly request a credential, a client generates a request as follows:
  1) Randomly choose a key-pair (d, g^d), where $d \leftarrow\$ \mathbb{Z}_q$.
  2) Encrypt the identity with Elgamal encryption as $E_{id} \leftarrow (g^r, g^{r \cdot d} g^{id})$.
  3) Generate a proof of knowledge π_r of (r, id) that proves the ciphertext is well-formed.

  Upon receiving a blind request (E_{id}, π_r), the platform issues an encrypted credential as follows:
  1) Verify the proof π_r and abort if the verification fails.
  2) Randomly choose $b \leftarrow\$ \mathbb{Z}_p$ and set $u \leftarrow g^b$.

3) Generate a ciphertext $E_{u'}$ of $u' \leftarrow g^{b(x_0+x_1 id)}$ from the encrypted identity E_{id}.
4) Generate a proof π_{blind} of (x_0, x_1, b), and return $(u, E_{u'}, \pi_{blind})$ to the client.

Finally, once receiving the platform's response, the client verifies the proof π_{blind} and decrypts $E_{u'}$ to get u'.

- $\mathsf{Show}(id, cred) \rightarrow (cm, \pi_{show}, rd)$: The prover chooses $r, z \leftarrow \mathbb{Z}_q$, and computes $C_{id} = u^{id}h^z$ and $C_{u'} = u'g^r$. It then output a tuple $cm = (u, C_{id}, C_{u'})$, randomnesses $rd = (r, z)$, and a proof:

$$\pi_{show} = \mathsf{PoK.Prv}\left\{(id, z, r) : C_{id} = u^{id}h^z \wedge V = g^{-r}X_1^z\right\}.$$

- $\mathsf{VfShow}(k_S, cm, \pi_{show}) \rightarrow \{0,1\}$: The verifier parses $cm = (u, C_{id}, C_{u'})$ and computes V as:

$$V = \frac{u^{x_0}C_{id}^{x_1}}{C_{u'}} = \frac{u^{x_0}u^{id \cdot x_1}h^{zx_1}}{u^{x_0+x_1 id}g^r} = \frac{h^{x_1 z}}{g^r} = g^{-r}X_1^z,$$

and verifies the proof π_{show}. If the proof is valid, it outputs '1'; otherwise, it outputs '0'.

Set Pre-Constrained Encryption (SPCE). We instantiate the SPCE using the protocol proposed by Bartusek et al. [4, Figure 2], inspired by the Apple PSI system [8].

- $\mathsf{Setup}(1^\lambda, \mathcal{B}) \rightarrow (sp, eb, pk_S, sk_S)$: The server setup the system as follows:

1) Initialize system parameter as $sp = (n', h_0, h_1) \leftarrow \mathsf{CH.Gen}(1^\lambda, n, \varepsilon)$, where $\mathsf{CH} = (\mathsf{Gen}, \mathsf{Hash})$ is a Cuckoo hashing.
2) Generate a key-pair as $(sk \leftarrow\$ \mathbb{Z}_q, pk \leftarrow g^{sk})$.
3) Encode the blocklist as follows:
    a) Generate a table $T \leftarrow \mathsf{CH.Hash}(n', h_0, h_1, \mathcal{B})$.
    b) For each element $T[i]$ in T: If $T[i] = \perp$, generate a key as $\overline{T}[i] \leftarrow g^{r_i}$; otherwise, generate a key as $\overline{T}[i] \leftarrow H(T[i])^{sk_S}$.
    c) Sign the table as $\sigma_{\overline{T}} \leftarrow \mathsf{DS.Sign}(sk_S, \overline{T})$.
4) Publish the tuple $(sp, pk_S, eb = (\overline{T}, \sigma_{\overline{T}}))$ to all clients.

- $\mathsf{Enc}(pk_S, m, p, eb) \rightarrow ct \text{ } or \perp$: To encrypt a payload p, for $b \in \{0,1\}$, a client produces a ciphertext as follows:

1) Verify the encoded blocklist as $\mathsf{DS.Vf}(\sigma_{\overline{T}}, pk_S, \overline{T})$, and output $\perp$ if the verification fails.
2) Randomly choose $\beta_b, \gamma_b \leftarrow\$ \mathbb{Z}_q$.
3) Compute $Q_b \leftarrow g^{\beta_b} \cdot H(m)^{\gamma_b}$ using the message m and encryption key as $S_b \leftarrow pk_S^{\beta_b} \cdot \overline{T}_{h_b(m)}^{\gamma_b}$.
4) Encrypt the payload as $C_b = \mathsf{SKE.Enc}(S_b, p)$, where the SKE scheme satisfies RKR.
5) Output the ciphertext $ct = (Q_0, C_0, Q_1, C_1)$.

- $\mathsf{Dec}(sk_S, ct) \rightarrow p \text{ } or \perp$: To decrypt a ciphertext ct, the server computes as follows:

1) For $b \in \{0,1\}$, reconstruct the key as $S'_b \leftarrow Q_b^{sk_S}$ and decrypt the ciphertext as $p_b \leftarrow \mathsf{SKE.Dec}(S'_b, C_b)$.
2) If both p_0 and p_1 are $\perp$, output $\perp$; otherwise, output the non-empty one. Note that, due to the SKE's RKR, at most one key will decrypt successfully.

Appendix B
Security Definitions and Proofs

A. Source Confidentiality

Before presenting the proof, we recall Theorem 1 as follows:

Theorem 3. *Let ST be the construction in Section V-A, for any PPT adversary $\mathcal{A}$, it holds that:*

$$\mathsf{Adv}_{ST}^{SConf}(\mathcal{A}) \leq \varepsilon_{PoK}^{ZK} + \varepsilon_{mkPKE}^{SS} + \varepsilon_{KVAC}^{ANON},$$

where ε_{PoK}^{ZK}, ε_{mkPKE}^{SS}, and $\varepsilon_{KVAC}^{ANON}$ denotes a PPT adversary's advantage in breaking the PoK's zero-knowledge property, the mkPKE's semantic security, and the KVAC's anonymity, respectively.

Proof. We prove the above theorem through indistinguishable games, where all the modifications happen in $\mathcal{O}_{\mathsf{send},b}$.

- G0 : We begin with the game that identical to $\text{SConf}_{ST}^{\mathcal{A},b}$ in Fig. 7 when $b = 0$.
- G1 : In Send, we model the hash function in PoK as a random oracle, enabling the challenger to program it.[2] As a result, the proof π_{send} will be generated in simulation mode. This modification decouples the proof from other components in the tracing metadata. The indistinguishability between this game and G0 can be easily reduced to the zero-knowledge property of the PoK scheme. Specifically, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G0}}(\mathcal{A})\right| \leq \varepsilon_{PoK}^{ZK}.$$

- G2 : In Send, we replace the key uk_0 with the key uk_1 of uid_1. This modification affects the ciphertexts $ct_{u,3}$ and ct_{tag} in the tracing metadata tmd. Since the two ciphertexts are encrypted using all the servers' keys and the adversary can only obtain two of the three keys, we can apply the semantic security of mkPKE. Specifically, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A})\right| \leq \varepsilon_{mkPKE}^{SS}.$$

- G3 : In TkGen, we finally replace the anonymous credential $cred_0$ with $cred_1$ of uid_1. Due to the anonymity of KVAC, its Show algorithm produces two indistinguishable credential presentations for two different input credentials. Thus, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G3}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A})\right| \leq \varepsilon_{KVAC}^{Anon}.$$

In the final game G3, the tracing metadata tmd_b is identical to tmd_1 in the game $\text{SConf}_{ST}^{\mathcal{A},b}$ when $b = 1$, which concludes the proof.

B. Trace Accountability

Before presenting the proof, we recall Theorem 2 as follows:

Theorem 4. *Let ST be the construction in Section V-A, for any PPT adversary, it holds that:*

$$\mathsf{Adv}_{ST}^{Act}(\mathcal{A}) \leq \varepsilon_{KVAC}^{Unf} + \varepsilon_{DS}^{Unf} + \varepsilon_{PoK}^{Sound} + \frac{q_h \cdot \varepsilon_{MAC}^{Unf}}{(1-\varepsilon_{PoK}^{Sound})(1-\varepsilon_{PoK}^{ZK})},$$

[2] We assume the PoK works in random oracle model (ROM), but the proof also holds when it is in the common reference string (CRS) model.

where $\varepsilon_{PoK}^{Sound}$ an adversary's advantage in breaking PoK's soundness; ε_{KVAC}^{Unf}, ε_{DS}^{Unf}, and ε_{MAC}^{Unf} denote the advantage in breaking the unforgeability of KVAC, DS, and MAC, respectively; q_h denotes the number of queries to $\mathcal{O}_{\mathsf{cred}}$.

Proof. We prove the above theorem through a series of indistinguishable games.

- G0 : This game is identical to $\mathrm{Act}_{ST}^{\mathcal{A}}$ in Fig. 8.
- G1 : In TkGen of $\mathcal{O}_{\mathsf{token}}$, we replace the anonymous credential's verification (i.e., KVAC.VfShow) with a check to determine whether the credential $cred$ is an output from $\mathcal{O}_{\mathsf{cred}}$. Since the adversary has no access to the KVAC issuer's secret key sk_P, the difference of $\mathcal{A}$'s advantage in G0 and G1 is upper bounded by the KVAC's unforgeability, i.e.,
$$\left|\mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G0}}(\mathcal{A})\right| \leq \varepsilon_{KVAC}^{Unf}.$$
- G2 : In Receive within the Trace algorithm, we replace the anonymous token's verification (i.e., $\mathsf{DS.Vf}(cm, pk_P, \sigma)$) with a check to determine whether the token (cm, σ) is output from $\mathcal{O}_{\mathsf{token}}$. Since the adversary has no access to the signer's secret key sk_P, we have:
$$\left|\mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A})\right| \leq \varepsilon_{DS}^{Unf}.$$

In the resulting game G2, $\mathcal{A}$'s advantage includes the probability that the two winning conditions occur. Let $S1$ and $S2$ be the events that $\mathcal{A}$ wins in the winning condition $(uid^* \notin (\mathcal{U}_m \cup \mathcal{U}_h))$ and $((uid^* \in \mathcal{U}_h) \wedge (uid^*, m^* \in \mathcal{L}_q))$, respectively. It is easy to see that the two events are independent. Thereby, we have:
$$\mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A}) = \Pr[S1] + \Pr[S2],$$

We now argue that, for any PPT adversary, the probability of the two events happening is negligible.

Claim 1. *For any PPT adversary $\mathcal{A}$ in G2, the following equation holds:*
$$\Pr[S1] \leq \varepsilon_{PoK}^{Sound}.$$

We substantiate this claim with a straightforward analysis on the tracing metadata tmd^* within the ciphertext $ct_{tmd,1}^*$. First, the Receive algorithm ensures that $token^*$ in tmd^* must have been previously queried in $\mathcal{O}_{\mathsf{token}}$. Consequently, we can retrieve a key uk^* corresponding to $token^*$ from $\mathcal{L}_{tk}$ and $\mathcal{L}_k$. Second, unless $\mathcal{A}$ can break the soundness of PoK, the proof π_{send} guarantees that the ciphertext $ct_{u,3}$ is a well-formed ciphertext of uk^*, implying that the decryption result must be uk^*. Since the key uk^* is directly retrieved from the list $\mathcal{L}_k$, the corresponding identity uid^* must have been previously queried in $\mathcal{O}_{\mathsf{cred}}$, thereby rendering the event $S1$ impossible.

Claim 2. *For any PPT adversary $\mathcal{A}$ in G3, the following equation holds:*
$$\Pr[S2] \leq \frac{q_h \cdot \varepsilon_{MAC}^{Unf}}{(1-\varepsilon_{PoK}^{Sound})(1-\varepsilon_{PoK}^{ZK})},$$

Proof. We prove this claim by contradiction. Suppose that, for a PPT adversary $\mathcal{A}$, the probability $\Pr[S2]$ is non-negligible, we show that there exists a PPT adversary $\mathcal{B}$ that has a non-negligible advantage in breaking the unforgeability of MAC (i.e., EU-CMA). Given a MAC oracle that responds to tags on request messages, $\mathcal{B}$ simulates G3 as follows:

1) *Setup.* $\mathcal{B}$ initializes the system normally and additionally randomly picks an index $j \in [q_h]$.
2) *Oracle queries.* Throughout the game, $\mathcal{B}$ answers $\mathcal{A}$'s queries to the oracles as follows:
    - $\mathcal{O}_{\mathsf{cred}}$: $\mathcal{A}$ makes this query with a tuple $(uid, isMal)$. If $isMal = 1$, $\mathcal{B}$ correctly executes the algorithms in the oracle. On the i-th query that $isMal = 0$, $\mathcal{B}$ proceeds as follows: If $i \neq j$, $\mathcal{B}$ executes as $isMal = 1$; otherwise, $\mathcal{B}$ sets $uid_j = uid_i$ and returns 0.
    - $\mathcal{O}_{\mathsf{token}}$: $\mathcal{A}$ makes this query with a credential $cred_u$. Because $\mathcal{B}$ owns the platform's secret key, it can execute the algorithms correctly.
    - $\mathcal{O}_{\mathsf{honSend}}$: $\mathcal{A}$ make this query with a tuple (m, uid). If $uid \neq uid_j$, $\mathcal{B}$ correctly executes the algorithms in the oracle; otherwise, $\mathcal{B}$ proceeds the query as follows:
        a) Generate two random bit strings as the anonymous token $token$ and encrypted identity $ct_{u,3}$.
        b) Forward the message m to the MAC oracle, and obtains tag as response.
        c) Generate the encrypted tag ct_{tag} using public keys and the tag.
        d) Generate the proof π_{send} by running the PoK in the simulation mode.
        e) Return $tmd = (token, ct_{u,3}, ct_{tag}, \pi_{send})$ to $\mathcal{A}$.
3) *Forge.* Finally, $\mathcal{A}$ outputs a tuple $(m^*, ct_{tmd,1}^*)$, where $ct_{tmd,1}^* = (tmd^*, ct_{u,2}^*, \pi_{d,2}^*, ct_{u,1}^*, \pi_{d,1}^*)$. The tracing metadata tmd^* satisfies $\mathsf{Receive}(m^*, tmd^*) = 0$ and $(uid^* \in \mathcal{U}_h) \wedge ((uid^*, m^*) \notin \mathcal{L}_q)$.

Due to the forward confidentiality, it is evident that $\mathcal{B}$'s simulation for G2 is perfect unless $\mathcal{A}$ can break the zero-knowledge property of the proof π_{send}. Thus, the simulation fails with a probability equal to ε_{PoK}^{ZK}.

Next, we analyze $\mathcal{B}$'s advantage in breaking the unforgeability of MAC. If $uid^* \neq uid_j$, $\mathcal{B}$ aborts. Otherwise, due to the proof of knowledge property of the PoK, $\mathcal{B}$ can extract the witness uk_j from the proof; then, it forwards $(m^*, \mathsf{MAC.Tag}(uk_j, m^*))$ to its own challenger. Consequently, $\mathcal{B}$ wins with the advantage:
$$\mathsf{Adv}_{MAC}^{Unf}(\mathcal{A}_{MAC}) \geq (1-\varepsilon_{PoK}^{ZK}) \cdot (1-\varepsilon_{PoK}^{Sound}) \cdot \frac{\Pr[S2]}{q_h},$$

which proves Claim 2.

Based on the indistinguishable games and Claim 1-2, we prove the Theorem 2.

C. Report Uniqueness

We formalize this property in the game $\mathrm{Unique}_{ST}^{\mathcal{A}}$ (see Fig. 9). The adversary can access the secret keys of malicious users $\mathcal{U}_m$, the reports of honest users $\mathcal{U}_h$, the moderator's secret key sk_M, and the regulator's secret key sk_R.

```
$\text{Unique}_{ST}^{\mathcal{A}}(1^\lambda)$
$\mathcal{L}_q, \mathcal{L}_k, \mathcal{L}_{tk}, \mathcal{U}_h, \mathcal{U}_m \leftarrow \emptyset$
$(sk_R, sk_M, sk_P, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(lb^*, \{rpt_i^*\}_{i\in[n]})$
        $\leftarrow \mathcal{A}^{\mathcal{O}^*}(sk_R, sk_M, pk)$
$\textbf{for } rpt \in \{rpt_i^*\}_{i\in[n]}:$
    $\textbf{if } rpt.lb \neq lb^* \vee rpt \notin \mathcal{L}_q:$
        $\textbf{return } 0$
    $\textbf{if } \mathsf{VfReport}(rpt, sk_P) = \perp:$
        $\textbf{return } 0$
$\mathcal{L}_{rpt} \leftarrow \mathsf{dedup}(\{rpt_i^*\}_{i\in[n]})$
$\textbf{if } |\mathcal{L}_{rpt}| > |\mathcal{U}_m|$
    $\textbf{return } 1$

$\mathcal{O}_{\mathsf{report}}(m, id, tmd)$
$\textbf{if } id \notin \mathcal{U}_h : \textbf{return } \perp$
$(uk, cred) \leftarrow \mathcal{L}_k[id]$
$rpt \leftarrow \mathsf{Report}(m, tmd,$
        $uk, cred, eb)$
$\mathcal{L}_q \xleftarrow{\cup} \{rpt\}$
$\textbf{return } rpt$

$\mathsf{dedup}(\{rpt_i'\}_{i\in[n]})$
$\mathcal{L}_{rpt}, \mathcal{L}_{dup} \leftarrow \emptyset$
$\textbf{for } dup_i \in \{rpt_i'\}_{i\in[n]}:$
    $\textbf{if } dup_i \notin \mathcal{L}_{dup}:$
        $\mathcal{L}_{rpt} \xleftarrow{\cup} \{rpt_i'\}$
        $\mathcal{L}_{dup} \xleftarrow{\cup} \{dup_i\}$
$\textbf{return } \mathcal{L}_{rpt}$
```

Fig. 9: Security game of report uniqueness, where $\mathcal{O}^*$ denotes the above $\mathcal{O}_{\mathsf{report}}$ and $\mathcal{O}_{\mathsf{cred}}$ in Figure 6.

```
$\text{Anon}_{ST}^{\mathcal{A},b}(1^\lambda)$
$\mathcal{L}_q, \mathcal{L}_k, \mathcal{L}_{tk}, \mathcal{L}_{tmd} \leftarrow \emptyset$
$\mathcal{U}_h, \mathcal{U}_m \leftarrow \emptyset$
$(sk_R, sk_M, sk_P, pk) \leftarrow \mathsf{KGen}(1^\lambda)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_b^*}(sk_P, \mathcal{L}_k, pk)$
$\textbf{return } b'$

$\mathcal{O}_{\mathsf{send}}(id, m)$
$\textbf{if } id \notin \mathcal{U}_h : \textbf{return } \perp$
$(uk, cred) \leftarrow \mathcal{L}_k[id]$
$tk \leftarrow \mathsf{TkGen}(uk, cred, sk_P)$
$tmd \leftarrow \mathsf{Send}(m, uk, tk)$
$\textbf{return } tmd$

$\mathcal{O}_{\mathsf{honRpt},b}(m, tmd, id_0, id_1)$
$\textbf{if } \mathsf{Receive}(m, tmd) = 0:$
    $\textbf{return } \perp$
$\textbf{for } id \in \{id_0, id_1\}:$
    $\textbf{if } ((id, tmd) \in \mathcal{L}_q)$
        $\vee (id \notin \mathcal{U}_h):$
        $\textbf{return } \perp$
    $\mathcal{L}_q \leftarrow \mathcal{L}_q \cup \{(id, tmd)\}$
$(uk_b, cred_b) \leftarrow \mathcal{L}_k[id_b]$
$rpt_b \leftarrow \mathsf{Report}(m, tmd, uk_b,$
        $cred_b, eb)$
$\textbf{return } rpt_b$
```

Fig. 10: Security game of reporting anonymity, where $\mathcal{O}_b^*$ denotes the above $\mathcal{O}_{\mathsf{honRpt},b}$ and $\mathcal{O}_{\mathsf{cred}}$, $\mathcal{O}_{\mathsf{send}}$ in Figure 6.

After querying the oracles, the adversary outputs a list of reports that should be successfully verified and have not been queried previously. The adversary wins if, after deduplication, the number of unique reports exceeds the number of malicious users. This indicates that the adversary has successfully created two duplicated tags that has the same reporter and message.

Definition 3 (Report uniqueness). *We say a source tracing scheme ST satisfies report uniqueness if for any PPT adversary $\mathcal{A}$, $\mathcal{A}$ has a negligible advantage in winning the game $Unique_{ST}^{\mathcal{A}}$, where the advantage is defined as follows:*

$$\mathsf{Adv}_{ST}^{\textit{Unique}}(\mathcal{A}) = \left|\Pr\left[\textit{Unique}_{ST}^{\mathcal{A}} = 1\right]\right|.$$

Theorem 5. *Let ST be the construction in Section V-A, for any PPT adversary, it holds that:*

$$\mathsf{Adv}_{ST}^{\textit{Unique}}(\mathcal{A}) \leq \varepsilon_{KVAC}^{Unf} + \varepsilon_{PoK}^{Sound}.$$

Proof. We conduct the proof through a series of indistinguishable games.

- $\mathsf{G0}$: This game is identical to $\text{Unique}_{ST}^{\mathcal{A}}$.
- $\mathsf{G1}$: In $\mathsf{VfReport}$, we replace the anonymous credential's verification (i.e., $\mathsf{KVAC.VfShow}$) with a check to determine whether the credential is previously queried in $\mathcal{O}_{\mathsf{cred}}$.
$$\left|\mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G0}}(\mathcal{A})\right| \leq \varepsilon_{KVAC}^{Unf}.$$
- $\mathsf{G2}$: Before dedup, we introduce an extractor to extract the witness $\{uk_i^*\}_{i\in[n]}$ from the proofs $\{rpt_i\}_{i\in[n]}$. Subsequently, we retrieve the corresponding identities $\{uid_i^*\}_{i\in[n]}$ of the keys $\{uk_i^*\}_{i\in[n]}$ from the list $\mathcal{L}_k$. The existence of this extractor is ensured by the proof of knowledge property of the proof π_{rpt}.
$$\left|\mathsf{Adv}_{ST}^{\mathsf{G3}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A})\right| \leq \varepsilon_{PoK}^{Sound}.$$

In the final game G2, the adversary has two winning events: 1) Forging a valid report as an honest user $uid^* \in \mathcal{U}_h$; 2) Creating different valid deduplicate tags from the same key belonging to malicious users $uid^* \in \mathcal{U}_m$. Note that, in $\mathsf{VfReport}$, the credential presentation's verification ensures that all witnesses of proofs passing this verification correspond to a registered user $uid^* \in (\mathcal{U}_h \cup \mathcal{U}_m)$. Therefore, these two events cover all possible cases in which the adversary wins.

Next, we argue that for any PPT adversary, the probability of both events occurring is negligible. First, owing to the unforgeability of KVAC, an adversary cannot forge an honest user's credential without querying the oracle $\mathcal{O}_{\mathsf{honSend}}$. Second, due to the unique tag property of MAC, a user's key uk^* can only generate a valid tag for the label lb, thereby precluding the possibility of duplicate tags. Therefore, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G3}}(\mathcal{A})\right| = 0,$$

which concludes the proof.

D. Reporting Anonymity

We formalize this property in the game $\text{Anon}_{ST}^{\mathcal{A},b}$ in Fig. 10. The oracle $\mathcal{O}_{\mathsf{send}}$ enables the adversary to obtain tracing metadata from any user for a chosen message, while the oracle $\mathcal{O}_{\mathsf{honRpt},b}$ allows the adversary to acquire reports on valid tracing metadata from honest users. The adversary's goal is to distinguish the source of a report. To prevent trivial wins, the pairs of tracing metadata and users queried in $\mathcal{O}_{\mathsf{honRpt},b}$ must be honest users who have not been queried before; otherwise, the adversary could exploit the deduplicate tag for report uniqueness to distinguish the reporter.

Definition 4 (Reporting anonymity). *We say a source tracing scheme ST satisfies reporting anonymity if for any PPT adversary $\mathcal{A}$, $\mathcal{A}$ has a negligible advantage in winning the game $Anon_{ST}^{\mathcal{A},b}$, where the advantage is defined as follows:*

$$\mathsf{Adv}_{ST}^{\textit{Anon}}(\mathcal{A}) = \left|\Pr\left[\textit{Anon}_{ST}^{\mathcal{A},1} = 1\right] - \Pr\left[\textit{Anon}_{ST}^{\mathcal{A},0} = 1\right]\right|.$$

Theorem 6. *Let ST be the construction in Section V-A, for any PPT adversary, it holds that:*

$$\mathsf{Adv}_{ST}^{Anon}(\mathcal{A}) \leq \varepsilon_{PoK}^{ZK} + \varepsilon_{KVAC}^{Anon} + \varepsilon_{MAC}^{wPRF},$$

where ε_{MAC}^{wPRF} denotes a PPT adversary's advantage in breaking the weak pseudorandomness of the MAC.

Proof. We conduct the proof through indistinguishable games, where all the modifications happen in $\mathcal{O}_{\mathsf{honRpt},b}$.

- $\mathsf{G0}$: We begin with the game that identical to $\mathrm{Anon}_{ST}^{\mathcal{A},b}$ when $b = 0$.
- $\mathsf{G1}$: In Report, we first replace the hash function with a random oracle, to make the PoK protocol work in simulation mode. This modification decouples the proof π_{send} from the report metadata that includes an anonymous token and the deduplicate tag. Specifically, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G0}}(\mathcal{A})\right| \leq \varepsilon_{PoK}^{ZK}.$$

- $\mathsf{G2}$: In Report, we again replace the secret key uk_0 with uk_1. This change specifically affects the deduplicate tag in the output. Due to the weak pseudorandomness of the MAC, the deduplicate tag remains indistinguishable from a random value. Owing to the source confidentiality, the tracing metadata input to the MAC appears as a random string from the adversary's view. Thus, we have:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A})\right| \leq \varepsilon_{MAC}^{wPRF}.$$

- $\mathsf{G3}$: In Report, we finally replace the anonymous credential $cred_0$ with $cred_1$ of uid_1. Similar to the $\mathsf{G3}$ in the proof of source confidentiality, the anonymity provided by KVAC conceals the user's identity during credential presentation. Therefore, the advantage of an adversary breaks reporting anonymity in $\mathsf{G2}$ and $\mathsf{G3}$ satisfies:

$$\left|\mathsf{Adv}_{ST}^{\mathsf{G2}}(\mathcal{A}) - \mathsf{Adv}_{ST}^{\mathsf{G1}}(\mathcal{A})\right| \leq \varepsilon_{KVAC}^{Anon}.$$

At the end of game hops, the output of $\mathcal{O}_{\mathsf{honRpt},0}$ is the same as $\mathrm{Anon}_{ST}^{\mathcal{A},b}$ when $b = 1$, which concludes the proof.