Impact Tracing: Identifying the Culprit of Misinformation in Encrypted Messaging Systems

Zhongming Wang†, Tao Xiang†, Xiaoguo Li†(✉), Biwen Chen†, Guomin Yang‡, Chuan Ma†, and Robert H. Deng‡
†College of Computer Science, Chongqing University, China
‡School of Computing and Information Systems, Singapore Management University, Singapore
{zmwang, txiang, csxgli, macrochen, chuan.ma}@cqu.edu.cn, {gmyang, robertdeng}@smu.edu.sg

***Abstract*—Encrypted messaging systems obstruct content moderation, although they provide end-to-end security. As a result, misinformation proliferates in these systems, thereby exacerbating online hate and harassment. The paradigm of "Reporting-then-Tracing" shows great potential in mitigating the spread of misinformation. For instance, *message traceback* (CCS'19) traces all the dissemination paths of a message, while *source tracing* (CCS'21) traces its originator. However, message traceback lacks privacy preservation for non-influential users (e.g., users who only receive the message once), while source tracing maintains privacy but only provides limited traceability.**

In this paper, we initiate the study of *impact tracing*. Intuitively, impact tracing traces influential spreaders central to disseminating misinformation while providing privacy protection for non-influential users. We introduce noises to hide non-influential users and demonstrate that these noises do not hinder the identification of influential spreaders. Then, we formally prove our scheme's security and show it achieves differential privacy protection for non-influential users. Additionally, we define three metrics to evaluate its traceability, correctness, and privacy using real-world datasets. The experimental results show that our scheme identifies the most influential spreaders with accuracy from 82% to 99% as the amount of noise varies. Meanwhile, our scheme requires only a 6-byte platform storage overhead for each message while maintaining a low messaging latency (< 0.25ms).

I. Introduction

End-to-end encrypted messaging systems (EEMSs), such as WhatsApp and iMessage, have been widely deployed in the real world and serve billions of people nowadays. These EEMSs provide a strong guarantee for users' message privacy since they ensure that only the communication parties (end users) can access the content of the messages. Unfortunately, the encrypted messages prevent the platform from moderating content to manage any abuse of the system [22], such as misinformation, thus exacerbating the proliferation of online hate and harassment [48]. As a result, EEMSs raise the tension between protecting users' message privacy and preventing platform abuse [47].

"Reporting-then-Tracing" is a classical paradigm in combating misinformation. It allows a user to report a problematic message, and then the platform can trace the users who disseminate it. *Message franking* [34], [20] is a real-world deployed scheme allowing recipients to report problematic messages and corresponding senders to the platform. Although supporting reporting, message franking reveals only the sender whom the message is directly from. In real social networks, the dissemination of misinformation typically occurs through forwarding (or retweeting) rather than its creation [25], [31], [43]. For instance, misinformation on Twitter can be retweeted by 1000 $\sim$ 100,000 people [52]. Therefore, tracing the dissemination path of a message is essential for combating misinformation. Both Brazil [42] and Indian [33] governments are considering legislation to make EEMSs support traceability. To provide traceability, Tyagi et al. [50] and Peale et al. [39] proposed *message traceback* and *source tracing*, respectively. The two concepts differ in tracing policies (shown in Figure 1), i.e., *which part of the dissemination path is tracked during tracing.*

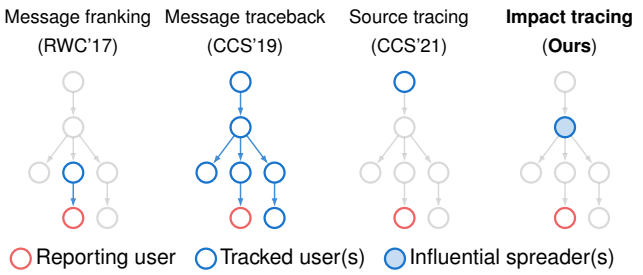


Fig. 1: Industrial and academic efforts in combat misinformation campaigns for EEMSs.

Existing tracing policies lie between two extremes. *Message traceback* reveals all users who disseminate the message, which allows the platform to reconstruct the entire dissemination path. This will infringe upon the privacy of users who merely receive or share a message out of concern, as they do not intentionally propagate misinformation. Source tracing only reveals the message's originator to the platform, which ignores the other users in disseminating the message. Although source tracing provides a better privacy guarantee, it incurs a significant loss in traceability and limits its ability to combat misinformation campaigns. At a high level, traceability is fundamentally incompatible with (message) privacy since the platform must know some information (weakening message privacy) to support traceability. Naturally, an interesting open problem arises.

"Is there a tracing policy that balances traceability and privacy, but also provides practical values to EEMSs?"

✉ Xiaoguo Li is the corresponding author.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240980
www.ndss-symposium.org

This paper initiates a new tracing policy, called ***impact tracing***, which reveals only the influential spreaders of a reported message while preserving the privacy of non-influential users. Impact tracing is motivated by the laws of message dissemination that a small group of users (called *influential spreaders*) contribute significantly to disseminating the messages where most users have a marginal impact [18], [17]. In social networks, influential spreaders act as amplifiers of information dissemination and are distributed on the branches of the dissemination paths. Identifying influential spreaders can reveal the backbone of misinformation dissemination, enabling the platform to conduct targeted interventions [53], [46].

A. Problem Statement

We model the EEMSs by a messaging graph $G = (V, E, M)$, in which (1) V denotes the set of vertices representing users; (2) M denotes the domain of messages; (3) $E \subseteq V \times V \times M$ denotes the set of edges representing message sending and forwarding. Then, we introduce several useful terms as follows (shown in Figure 2).

- *Forwarding graph*: $G_m = (V_m, E_m, m)$ represents the dissemination paths of a particular message m, which includes all the users who sent or received the message.
- *Social graph*: $G = (V, E, M)$ represents the entire messaging system, which contains all the users and their communications, where $G = \cup_{m \in M} G_m$.
- *Sociogram*: $G^s = (V^s, E^s, -)$ represents the connections among users, which is obtained from G by merging the parallel (directed) edges into a single (undirected) edge.

Based on the above definitions, we can formalize traceability for EEMSs as a subgraph publication algorithm: Given a social graph G, starting vertex v_s, and message m, publishing a set of vertices V_m^o, such that $V_m^o \subseteq V_m$. Here, v_s represents the reporting user who triggers the tracing process. Specifically, in message traceback, V_m^o is equal to the whole vertices set of G_m, i.e., $V_m^o = V_m$; in source tracing, V_m^o only contains the originator who creates the message m. In contrast to these examples, impact tracing presents a subgraph publication challenge but exhibits two distinct disparities:

Identifying the influential spreaders. Impact tracing requires that V_m^o contains the influential spreaders in disseminating the message m. Although it is natural to assume that influential users are hubs in social networks, i.e., users who forward a reported message multiple times, this does not always hold in reality. For instance, a hub located at the periphery of a network may contribute little to message dissemination [5], [51]. The problem of identifying influential spreaders in social networks is known as the *influence maximization problem* [26]. In this work, we have adopted k-Shell decomposition [28] to evaluate vertices' impact in G_m since it performs better than other methods when the graph is weakly connected and originates from a single vertex. Specifically, the k-shell of a graph is defined as the set of vertices that belong to the k-core but not to the (k-1)-core, where the k-core is a subgraph such that every vertex has a degree at least k.

Protecting the non-influential users' privacy. Ideally, to protect privacy, impact tracing should prevent the platform from learning any information about G_m apart from the set V_m^o. Nevertheless, this appears infeasible since the platform must learn more than V_m^o from the known sociogram G^s. Let $u, v \in V_m^o$ be two influential spreaders that are only connected by a vertex $w \in \{V^s - V_m^o\}$. The platform can deduce that w is a non-influential user spreading the message since the message forwarding must be connected.

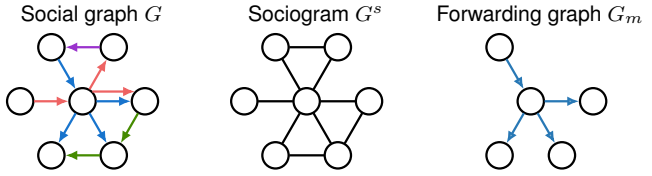


Fig. 2: Illustration of graphs. The different colors in G represent different messages.

Alternatively, we consider differential privacy (DP), a *de facto* standard for privacy protection. Specifically, we introduce noises to G_m to prevent the platform from learning the presence of an individual in the tracing results. We will provide a formal DP definition for protecting the non-influential users' privacy in Section IV-B.

B. Technical Overview

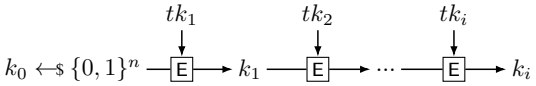
Message franking. Aiming at impact tracing, we begin with a message franking scheme [34], allowing a recipient to report a message and its sender. Specifically, it involves two phases:

1) Messaging phase: The sender first generates a message tag $tag \leftarrow F_k(m)$, where k is a randomly chosen tag key and F is a pseudorandom function. Then, the sender sends a packet $p = (\mathsf{E2EE}(m, k), tag)$ to the platform, where (m, k) is sent using end-to-end encryption. The platform stores the tag and transmits the packet to the recipient.
2) Reporting phase: A recipient submits a report as a tuple (m, k, tag). The platform verifies the report by checking for the existence of tag in storage, i.e., a membership test between the message tag and platform storage.

This simple solution achieves two nice security features. First, tag hides the message m due to the pseudo-randomness of F; Second, it provides accountability because tag commits the message, and the platform binds this tag with the sender in its storage. Technically, message franking provides an approach to trace the sender when the platform has the tuple (m, k, tag). However, it does not provide traceability since the senders generate the tag keys independently.

Message traceback. To enable traceability, we must allow the platform to trace all senders and recipients of a reported message along with the forwarding paths. Therefore, instead of choosing tag keys randomly, we introduce a novel *encryption chain* to link the tag keys on the same forwarding path via tracing keys. Specifically,

1) Messaging phase: The sender generates the tag key via $k_i \leftarrow \mathsf{E}_{tk_i}(k_{i-1})$, where tk_i represents the sender's tracing key known to the sender and the platform, and k_{i-1} is the tag key received from its precursor — namely, the user who forwards the message to the sender. If the message is a fresh one, k_{i-1} (i.e., k_0) is randomly chosen by the sender. The



sender then generates the message tag via $tag \leftarrow F_{k_i}(m)$ and sends the corresponding packet to the platform. The platform stores the communication parties' identity (i.e., edges in the sociogram G^s) when transmitting the packets.

2) Reporting-then-Tracing phase: Upon receiving (m, k_i, tag) from a reporting recipient, the platform can trace the sender as the message franking scheme. Moreover, the platform can recover the previous k_{i-1} and next tag key k_{i+1} through decryption by tk_i and encryption by tk_{i+1}, respectively. Therefore, starting from the reported user's k_i, the platform can traverse the sociogram G^s along the encryption chain. With the tag key k_{i-1} (or k_{i+1}), the platform calculates a unique message tag that binds a particular sender in its storage. Then, the platform checks whether tag exists. If so, it outputs the corresponding sender who submitted the tag during the messaging phase. Iteratively, the platform can find all users who disseminate messages.

Here, the encryption's security guarantees that the tag keys look like random strings before reporting, which hides the message from the tag. Moreover, accountability is also held because the tracing key binds the two users' identities corresponding to each edge in the sociogram G^s. Unfortunately, this scheme does not provide privacy for non-influential users since it reveals the exact forwarding graph G_m to the platform.

Impact tracing. To protect non-influential users' privacy, our main idea is to let the platform learn only a noisy graph G_m^* during tracing. At a high level, we design the impact tracing in two steps: (1) designing a *fuzzy message traceback* scheme, from which the platform only obtains the noisy graph G_m^*; and (2) identifying the influential spreaders via a novel *decoding algorithm* from G_m^*. In the design, the following three challenges arise.

C1. *How to add noises to G_m without sacrificing security?*

Achieving impact tracing is infeasible in a single-server setting. Traceability, in essence, enables the platform and reporting recipients to uncover the entire forwarding graph G_m. In contrast, impact tracing aims to identify only the influential spreaders within G_m. Relying on the platform or recipient to hide non-influential users is impractical, as we cannot prevent collusion between them. On the other hand, letting the sender obscure G_m compromises accountability, since it allows malicious senders to manipulate the tracing result. Consequently, no participant in a single-server setting can be trusted to obscure non-influential users.

Hence, we introduce a tag server in addition to the platform for managing the message tags and adding noises in the tracing phase. Instead of checking the existence of tag in its local storage (as in message traceback), the platform now queries the tag server to check whether the tag exists during tracing. The tag server answers the queries in a randomized response (RR) manner. Specifically, if tag exists, tag server outputs 1; otherwise, it returns 1 only with a preset probability ψ, also called a false positive rate (FPR). This approach results in the misidentification of a non-existent tag as existing (from the view of the platform) with FPR ψ. More importantly, it reveals a noisy graph G_m^* such that $G_m \subseteq G_m^*$ and the false positives hide the presence of non-influential users in G_m^*. For example, if $\psi = 1$ (or 0), it reveals the entire sociogram G^s (or the exact forwarding graph G_m) to the platform. Moreover, we will formally show that the above strategy achieves DP privacy protection for non-influential users in Section IV-B.

C2. *How to uncover influential spreaders from G_m^*?*

Although false positives safeguard non-influential users' privacy, they impede the tracking of influential spreaders. In detail, the tracing result G_m^* contains (1) the users who indeed disseminated the message (true positives, i.e., $\{v : v \in V_m\}$) and (2) the users who were misidentified (false positives, i.e., $\{v : v \notin V_m \cup v \in V^s\}$). One may consider using the k-shell decomposition to evaluate the users' impact on the noisy graph G_m^*. But, if a user v has a high degree in G^s, false positives overwhelm the influence evaluation, leading to the misidentification of the user v as influential, even if they never sent or received the reported message. Therefore, this approach does not perform well in uncovering the influential spreaders.

Instead, we uncover true positives from G_m^* through a decoding algorithm that computes a *membership value* for each $v \in G_m^*$. This value indicates the posterior probability that the vertex is indeed a true positive. Our main idea is to calculate the membership values based on Bayes' theorem for each $v \in G_m^*$ and then output the vertices with membership values greater than a certain threshold. Besides, we further correct the membership values through two observations: (1) a vertex should have only one (true) precursor, and (2) a vertex must be true positive if one of its descendants is truly positive. We refer readers to Section III-C for details.

C3. *How to measure the performance of impact tracing?*

Like other DP mechanisms, there is a trade-off between utility and privacy in impact tracing. In detail, the utility of impact tracing involves traceability and correctness, meaning that V_m^o should contain as many influential users and as few false positives as possible. Furthermore, privacy means that G_m^* should contain sufficient false positives to hide the true positives. We define three metrics, i.e., the detection rate, output FPR, and interval FPR, to reflect the traceability, correctness, and privacy, respectively. Refer to Section V for a detailed description of the metrics.

We measure the metrics of our design by simulations on real-world social network datasets. The results demonstrate that our scheme provides privacy for non-influential users while achieving traceability on influential spreaders. In particular, users with less influence will be hidden by more noise, resulting in better privacy protection. Conversely, users with greater influence will be more easily identified by the platform. As the amount of noise varies, the influential spreaders identified by our scheme are 82% - 99% consistent with those obtained through the k-shell decomposition of the actual forwarding graph G_m.

Our contributions. To summarize, in this paper, we:

- introduce a new tracing policy, i.e., impact tracing, for EEMSs, which balances traceability and the privacy;

- propose a fuzzy message traceback scheme that introduces noises to hide non-influential users and demonstrate that these noises do not affect identifying influential spreaders;
- formalize its security goals and show it provides accountability while preserving E2EE security. Meanwhile, it achieves DP protection for non-influential users;
- implement a prototype, benchmark its performance, and evaluate its traceability, correctness, and privacy with three well-designed metrics on real-world datasets.

Roadmap. We formulate the models and goals for impact tracing in Section II, followed by its design in Section III. Then, we present our design's security and privacy analysis in Section IV; Section V evaluates our design's utility and performance on real-world datasets. Finally, Section VI presents the related work, and Section VII concludes this paper.

II. Models and Goals

In this section, we define the models and (non-)goals that guide the design of our approach.

A. System Model

As shown in Figure 3, the system involves three types of entities. To enable impact tracing on EEMSs, the system mainly consists of (1) the messaging phase and (2) the reporting-then-tracing phase.

- **Users.** Users play the role of both senders and recipients. During the messaging phase, the senders submit encrypted messages and tracing metadata to the platform and the tag server respectively; the recipients verify the received packets and reject packets containing malformed tracing metadata. Furthermore, the recipients can report problematic messages to the platform. Additionally, the users in a forwarding graph can be divided into influential and non-influential users according to their k-shell value.
- **Platform.** In the messaging phase, the platform transmits packets from senders to corresponding recipients and the tag server. In the reporting-then-tracing phase, it receives reports from the recipients and then traces the influential spreaders with the help of the tag server.
- **Tag server.** The tag server logs tracing metadata from users in the messaging phase and assists the platform in tracking the influential users in the reporting-then-tracing phase.

B. Threat Model

We assume all user communications pass through the platform and are properly authenticated by the platform.

- **Users.** Users are either honest-but-curious or malicious. All users are curious about the forwarding path of received messages. A malicious sender may generate malformed tracing metadata to evade tracing, whereas a malicious recipient can ignore the verification of tracing metadata. Moreover, users may collude with each other to manipulate tracing results so that a non-influential user may be misidentified as influential.
- **Platform.** We assume the platform is honest-but-curious and does not collude with the other participants. In the messaging phase, the platform transmits the packets between users honestly but attempts to learn the content or forwarding path of messages from these packets. In the reporting-then-tracing phase, the platform attempts to identify the non-influential users. As with the existing designs [54], [50], the platform can obtain the users' sociogram.
- **Tag server.** The tag server is honest-but-curious, which executes the pre-defined operations but tries to learn information about messages from received packets. Moreover, we assume that the tag server will not collude with the platform, but it may collude with the users. In practice, this server can be run by a third-party moderator, such as independent organizations [10] and national authorities [11]. The setting of non-colluding two servers is widely used in research [21], [27] and is moving into practical deployment [1], [14], [19].

C. Design Goals

To enhance practicality, impact tracing should minimize the impact on the security and performance of the existing EEMSs. The detailed goals are listed as follows.

Impact tracing. We aim to balance traceability and privacy with a novel tracing policy that satisfies:

- *Traceability.* The platform can identify influential spreaders who disseminate the reported message.
- *Privacy.* The platform cannot uncover non-influential users from the tracing result.

Security. The security goals involve two aspects. First, impact tracing should not weaken the end-to-end security of the underlying EEMS. Second, impact tracing should provide accountability against malicious users.

- *Platform confidentiality.* The platform cannot learn any information about the content or forwarding path of a message unless the message is reported.
- *Tag server confidentiality.* The tag server cannot learn information about a message's content, forwarding path, or connection between users.
- *User confidentiality.* Users cannot learn any information about the forwarding path of received messages regardless of whether the messages are reported or not.
- *Accountability.* Users cannot generate a message that cannot be traced, and a group of colluding users cannot manipulate the tracing results to smear an honest user.

Efficiency. EEMSs are required to process massive messages; even one extra byte of overhead on each message would significantly degrade the performance. Therefore, impact tracing should be efficient enough. Specifically,

- *Messaging latency.* Processing tracing metadata should not significantly increase the latency of messaging.
- *Storage.* Storing tracing metadata should not incur significant storage overhead for all participants.
- *Bandwidth.* Delivering tracing metadata should not incur significant bandwidth overhead.

D. Non-goals

Similar to prior works (e.g., [50], [39]), this work has some fundamentally unavoidable limitations.

Avoiding copy-paste. Copy-paste attack is an inherent limitation of any existing tracing scheme for EEMSs. A user can copy a received message and paste it into the message box to re-send it as a fresh message instead of using the forward feature. As a result, the user will be identified as the message's originator rather than a forwarder in tracing. Our scheme is built on the forward feature of EEMSs; thus, it also suffers from this limitation. To mitigate copy-paste's negative impact, an EEMS can simultaneously deploy impact and source tracing. Consequently, both the originator (including any malicious user who copy-pastes messages) and influential spreaders of the reported message will be revealed after tracing.

Supporting metadata-private messaging. To avoid metadata leakage (e.g., users' identities), many metadata-private messaging systems [2], [6] have been proposed to prevent metadata collection from government agencies or platforms. Nevertheless, most existing real-world EEMSs (e.g., WhatsApp, iMessage) do not provide metadata privacy since hiding users' metadata always incurs high costs [12]. However, the Signal system introduced and deployed a sealed sender feature [23] that hides senders' identity in messaging. Therefore, designing impact tracing schemes for metadata-private messaging systems remains a future work.

Preventing false reporting. False reporting indicates that a recipient submits a report that contains an innocent message, such as harmless memes. Ideally, the platform would verify whether the reported message is problematic and reject the invalid reports. Nevertheless, the platform may start tracing without verification to steal users' privacy. This problem can be resolved by integrating impact tracing with a threshold reporting scheme (e.g., [9], [30]). As a result, the platform learns tracing metadata if and only if a message is reported multiple times.

III. Impact Tracing

In this section, we detail the design of impact tracing.

A. Preliminaries

Cryptographic primitives. We formalize cryptographic primitives that are relevant to our scheme as follows:

- $F_k : \{0,1\}^x \times \{0,1\}^x \to \{0,1\}^y$ is a collision-resistant pseudorandom function, where x and y are the input and output length, respectively.
- $H : \{0,1\}^* \to \{0,1\}^x$ is a collision-resistant hash function.
- $\mathsf{E2EE}$ is the underlying encryption scheme of an EEMS.
- $\Sigma = (\mathsf{E}_k, \mathsf{D}_k)$ is a block cipher for the encryption chain.
- $\Lambda = (\mathsf{Enc}_k, \mathsf{Dec}_k)$ is a symmetric encryption scheme for encrypting tags. We require Λ satisfies *random key robustness (RKR)* [16], i.e., for any plaintext m and $k \neq k'$, $\Pr[\Lambda.\mathsf{Dec}_{k'}(\Lambda.\mathsf{Enc}_k(m)) \neq \perp] = \mathsf{negl}(1^\lambda)$, where 1^λ is the security parameter.

Data storages. Our scheme involves three databases. First, the platform holds two key-value databases $\mathsf{DB_{ik}}$ and $\mathsf{DB_{nbr}}$, which store users' identity keys (U_i, ik_i) and connections (U_i, U_j) (i.e., users' neighbors in the sociogram), respectively. The U_i and ik_i are a user's public identity and secret identity key, respectively. Furthermore, these key-value databases support Add and Query operations that allow the platform to add a key-value pair and query the value of a key respectively. Second, the tag server stores a set of message tags in $\mathsf{DB_{tag}}$, which allows the tag server to check the existence of a tag using the Exist operation.

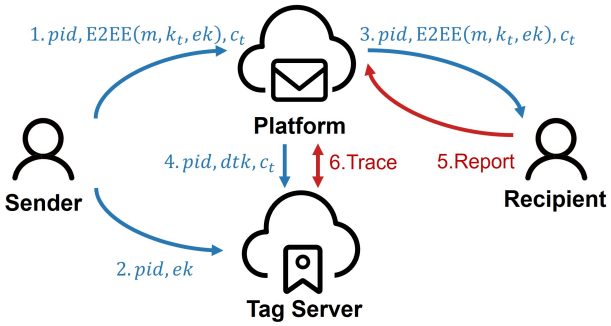


Fig. 3: Workflow of impact tracing. The blue and red lines represent the messaging and reporting-then-tracing phases.

B. Fuzzy Message Traceback

The algorithms of our scheme are given in Figure 5, and the workflow in different phases is presented as follows.

Setup Phase All users register to the platform before messaging, where the platform generates an identity-key pair (U_i, ik_i) for each user and stores it in $\mathsf{DB_{ik}}$. Moreover, the platform initializes a secret key psk for deriving keys for the tag server. The tag server initializes the false positive rate (FPR) ψ that is used in random response and shares it with the platform.

Messaging Phase This phase includes four steps.

Sending a message. To send a message m to a recipient U_j, the sender U_i generates a message packet as follows:

1) Calculate the tracing key $tk_{ij} \leftarrow H(ik_i || U_j)$.
2) Generate a tag key $k_t \leftarrow \Sigma.\mathsf{E}_{tk_{ij}}(k_{t-1})$. If m is a fresh message, $k_{t-1} \leftarrow\$ \{0,1\}^x$; otherwise, k_{t-1} is the tag key along with a forwarded message.
3) Compute a message tag as $tag \leftarrow \mathsf{TagGen}(m, k_t)$.
4) Encrypt the message tag as $c_t \leftarrow \Lambda.\mathsf{Enc}_{ek}(tag)$, where ek is a randomly chosen ephemeral key.
5) Generate an E2EE ciphertext $e \leftarrow \mathsf{E2EE}(m, k_t, ek)$.
6) Send the packet $p_c = (pid, e, c_t)$ to the platform, where pid is an unique packet identity.
7) Send the packet $p_s = (pid, ek)$ to the tag server.

Processing a message. Once receiving a packet p_c, the platform processes it as follows:

1) Record the communication parties (U_i, U_j) in $\mathsf{DB_{nbr}}$, where the users' identities U_i and U_j are contained in messaging metadata.
2) Generate the tracing key $tk_{ij} \leftarrow H(ik_i || U_j)$, where $ik_i \leftarrow \mathsf{DB_{ik}.query}(U_i)$.
3) Derive $dtk_{ij} \leftarrow \mathsf{DtkDer}(tk_{ij}, psk)$ from the tracing key.
4) Transmit the packet $p'_c = (pid, e, c_t)$ to the recipient U_j.
5) Send the packet $p'_s = (pid, dtk_{ij}, c_t)$ to the tag server.

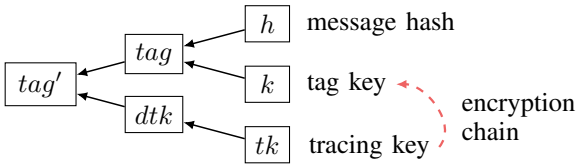


Fig. 4: Relations among the tracing metadata.

Receiving a message. Once receiving a packet p'_c from the platform, the recipient U_j verifies it as follows:

1) Decrypt e to (m, k_t, ek) via the E2EE.
2) Check whether the same key k_t has been received before (i.e., replicated tag key) and reject the packet if it has.
3) Decrypt the encrypted tag as $tag \leftarrow \Lambda.\mathsf{Dec}_{ek}(c_t)$, and reject the packet if the decryption fails.
4) Compute the message tag as $tag^* \leftarrow \mathsf{TagGen}(m, k_t)$ and reject the packet if $tag^* \neq tag$.
5) Store the key k_t in client storage for future forwarding.

Storing a tag. The tag server processes packets p_s and p'_s with the *same* pid as follows:

1) Parse $p_s = (pid, ek)$ and $p'_s = (pid, dtk_{ij}, c_t)$.
2) Decrypt the encrypted tag as $tag \leftarrow \Lambda.\mathsf{Dec}_{ek}(c_t)$ and report the packets to the platform when decrypting fails.
3) Store the processed message tag tag' in $\mathsf{DB}_{\mathsf{tag}}$, where $tag' \leftarrow \mathsf{TagProc}(dtk_{ij}, tag)$.

At a high level (as shown in Figure 4), each processed tag tag' is calculated from a hash and PRF, whose key dtk is derived from the tracing key tk and the message tag tag binds the tag key and the message. Note that the tag key k_t is transmitted through the underlying E2EE scheme; the PRF's pseudorandomness ensures that the message tags leak nothing about the underlying messages.

Reporting-then-Tracing Phase A recipient U_j can report a problematic message by submitting the tuple (m, k_t, U_i) to the platform, where U_i is their precursor. Then, the platform traces the influential spreaders as follows.

1) Execute the Trace algorithm and obtain the noisy graph $G^*_m \leftarrow \mathsf{Trace}(m, k_t, U_i)$.
2) Identify a set V^o_m of influential spreaders from G^*_m via the decoding algorithm presented in Section III-C.

Workflow of tracing. We now detail how the Trace algorithm in Figure 5 produces a noisy graph G^*_m between the platform and tag server. When the recipient reports a tuple (m, k_t, U_i), Trace starts from the reporter U_i and traverses the sociogram G^s in a breadth-first search manner. Specifically,

1) Initialize a senders' set $\mathcal{S}$ that only contains the sender U_i for the first time, and a recipients' set $\mathcal{R}$ that is empty.
2) Find all neighbors of the users in $\mathcal{S}$ and $\mathcal{R}$ from the sociogram G^s, where G^s is stored in $\mathsf{DB}_{\mathsf{nbr}}$ and collected by the platform in the messaging phase.
3) Compute message tags for all the connections between users and their neighbors. The platform first computes the tracing keys of these connections with the users' identity keys. And then, the platform recovers:
   a) the previous tag keys k_{t-1} through decryption to trace the precursors in BwdSearch;
   b) the next tag keys k_{t+1} through encryption to trace the descendants in FwdSearch.

   Finally, with the tracing keys, tag keys, and the reported message, the platform calculates unique message tags for these connections (as shown in Figure 4).
4) Check whether these tags exist in the tag server's storage

$\mathsf{TagGen}(m, k)$
$h \leftarrow H(m)$
$tag \leftarrow F_k(h)$
return tag

$\mathsf{TagProc}(dtk, tag)$
$tag' \leftarrow H(dtk||tag)$
return tag'

$\mathsf{DtkDer}(tk, psk)$
$dtk \leftarrow \Sigma.\mathsf{E}_{psk}(tk)$
return dtk

$\mathsf{TagExist}(m, k, tk, psk)$
/ platform generates tag'
$tag \leftarrow \mathsf{TagGen}(m, k)$
$dtk \leftarrow \mathsf{DtkDer}(tk, psk)$
$tag' \leftarrow \mathsf{TagProc}(dtk, tag)$
/ tag server checks its existence
$b \leftarrow \mathsf{DB}_{\mathsf{tag}}.\mathsf{exist}(tag')$
$b' \leftarrow \mathsf{RandResp}(b)$
return b'

$\mathsf{RandResp}(b)$
if $b = 1$ **then** $b' = 1$
else : / introduce noises
  $b' = 1$ with prob. ψ
  $b' = 0$ with prob. $1 - \psi$
return b'

$\mathsf{Trace}(m, k_t, U_i)$
Init $\mathcal{S} \leftarrow \{(k_t, U_i)\}, \mathcal{R} \leftarrow \emptyset$
Init $G^*_m \leftarrow U_i$ / G^*_m is initialized to a null graph with vertex U_i
while $\mathcal{S} \neq \emptyset \mid \mathcal{R} \neq \emptyset$:
  for $(k_t, U_i) \in \mathcal{S}$: / backward tracing the senders
    $\{(k_{t-1}, U_{i-1})\} \leftarrow \mathsf{BwdSearch}(m, k_t, U_i)$
    $G^*_m \leftarrow G^*_m \cup \{(U_{i-1}, U_i, m)\}$ / insert the edge to G^*_m
  Init $\mathcal{R}^* \leftarrow \emptyset$
  for $(k, U) \in \mathcal{S} \cup \mathcal{R}$: / forward tracing the recipients
    $\mathcal{R}' \leftarrow \mathsf{FwdSearch}(m, k, U)$
    $\mathcal{R}^* \leftarrow \mathcal{R}^* \cup \mathcal{R}'$
    for $(k', U') \in \mathcal{R}'$
      $G^*_m \leftarrow G^*_m \cup \{(U, U', m)\}$
  $\mathcal{R} \leftarrow \mathcal{R}^*; S \leftarrow \{(k_{t-1}, U_{i-1})\}$
return G^*_m

$\mathsf{BwdSearch}(m, k_t, U_j)$
$\mathcal{T} \leftarrow \mathsf{DB}_{\mathsf{nbr}}.\mathsf{query}(U_j)$
for $U_i \in \mathcal{T}$:
  $ik_i \leftarrow \mathsf{DB}_{\mathsf{ik}}.\mathsf{query}(U_i)$
  $tk_{ij} \leftarrow H(ik_i||U_j)$
  $b \leftarrow \mathsf{TagExist}(m, k_t, tk_{ij})$
  if $b = 1$ **then**
    $k_{t-1} \leftarrow \Sigma.\mathsf{D}_{tk_{ij}}(k_t)$
    return $\{(k_{t-1}, U_i)\}$
return $\emptyset$

$\mathsf{FwdSearch}(m, k_t, U_i)$
Init $\mathcal{R}' \leftarrow \emptyset$
$ik_i \leftarrow \mathsf{DB}_{\mathsf{ik}}.\mathsf{query}(U_i)$
$\mathcal{T} \leftarrow \mathsf{DB}_{\mathsf{nbr}}.\mathsf{query}(U_i)$
for $U_j \in \mathcal{T}$:
  $tk_{ij} \leftarrow H(ik_i||U_j)$
  $k_{t+1} \leftarrow \Sigma.\mathsf{E}_{tk_{ij}}(k_t)$
  $b \leftarrow \mathsf{TagExist}(m, k_{t+1}, tk_{ij})$
  if $b = 1$ **then**
    $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{(k_{t+1}, U_j)\}$
return $\mathcal{R}'$

Fig. 5: Algorithms of the fuzzy traceback scheme.

using the TagExist algorithm. Note that TagExist is an interactive algorithm between platform and tag server.
5) For the tags that exist in the tag server's storage, add the corresponding precursors and descendants to the sets $\mathcal{S}'$ and $\mathcal{R}'$ respectively, which are initialized to empty sets.
6) If $\mathcal{S}'$ or $\mathcal{R}'$ is not empty after all these neighbors have been checked, the next round of loops begins from them; otherwise, the traceback ends.

The Randomized Response (RR). To prevent the platform from learning the exact forwarding graph G_m, we let the tag server answer the tags' existence in random response (see the RandResp algorithm). Specifically, for each tag not in DB_{tag}, tag server responds 'yes' to the platform with probability ψ, which introduces false positives to the output. Consequently, the Trace algorithm produces a noisy version G_m^* of G_m. We formally demonstrate that the RR mechanism achieves DP privacy protection for tracked users in Section IV-B.

Remark 1 (Tag key reuse). A sender may forward a message to a recipient more than once. In this case, the sender will generate the same tag key that already appeared, i.e., tag key reuse. The recipient will reject the second message because the same tag key has been received. To resolve this problem, we only need to introduce a counter w to the tag key generation. w is initialized to 0 and increments by one when the message is forwarded to the same recipient. From the second forwarding, the sender send (k_t', w) instead of k_t to the recipient, where $k_t' \leftarrow H(k_t||w)$. The recipient can verify k_t' since it already obtains k_t. When the recipient forwards the received message, it uses k_t first received from that sender to derive subsequent tag keys. This guarantees that a traceback from the recipient can be traced to the sender successfully.

Remark 2 (Multiple reports). A problematic message may be reported multiple times by distinct users, resulting in multiple traces within the same forwarding graph. This situation undermines the privacy protection provided by random responses. To mitigate this issue, the tag server mandates that the platform submits a different tag for each tag query. Consequently, the platform must initiate traceback from previous tracing results when handling subsequent reports of the same message. Even if a platform queries a duplicated tag, the tag server can readily identify this misbehavior by comparing it with prior queries.

Remark 3 (Multi-modal messages). Misinformation is often disseminated through multi-modal messages, including images and videos. Our scheme inherently supports such multi-modal messages by transforming input messages into message tags before further operations. Specifically, in the TagGen algorithm, users hash the input message before computing the corresponding tag, and all subsequent operations are performed on the message hash. Since hash functions are compatible with any message format, our scheme can seamlessly handle messages of any format. Consequently, the size of the tracing metadata is independent of the message size.

C. The Decoding Algorithm

Without considering the non-influential users' privacy, we can let the platform obtain the exact forwarding graph $G_m = (V_m, E_m, m)$ during tracing. Naturally, the platform can use the k-shell decomposition to evaluate the users' impact on G_m. However, to protect these users' privacy, impact tracing lets the platform only obtain a noisy tracing result. Specifically, the Trace algorithm traverses the entire sociogram $G^s = (V^s, E^s, -)$ (defined in Section I-A), and outputs a noisy graph $G_m^* = (V_m^*, E_m^*, m)$ contains: 1) The users who indeed disseminated the message (true positives, i.e., $\{v|v \in V_m\}$), and 2) The users who were misidentified (false positives, i.e., $\{v|v \notin V_m \cup v \in V^s\}$). Henceforth, we aim to identify the influential spreaders in G_m^*.

Certainly, one may consider using the k-shell decomposition directly to identify influential spreaders on the noisy graph G_m^*. Unfortunately, due to the noises, we found that the k-shell decomposition performs poorly on G_m^*. Let $n(v)$, $n_m(v)$ be the number of v's neighbors in G^s and in G_m, respectively. According to the RR mechanism, the number of v's neighbors in G_m^* can be denoted as $n_m^*(v) = n_m(v) + \mathbf{x}$, where $\mathbf{x} \sim B(n(v) - n_m(v), \psi)$ and $B(\cdot,\cdot)$ denotes the binomial distribution. In practice, there is a group of vertices with large $n(v)$ since the degree distribution of social networks is similar to the power-law distribution [35]. When $n(v)$ is large enough, $\mathbf{x}$ dominates the value $n_m^*(v)$, where $\mathbf{x}$ is the number of false positives in v's neighbors. Ultimately, this naive approach would take all false positives into the impact assessment and misidentify the false positives as influential ones. Therefore, we need an alternative approach to minimize the false positives' effect on the tracing results.

Decode strategy. For each vertex $v \in V_m^*$, the platform cannot recover the exact $n_m(v)$ from the $n_m^*(v)$. Therefore, our intuitive idea is to calculate a posterior probability μ_v (also called membership value in the fuzzy systems [55]) for each $v \in G_m^*$ and then select vertices with posterior probabilities greater than a certain threshold as the output, i.e., V_m^o.

In detail, our decoding algorithm (shown in Algorithm 1) includes four steps, where the used notations are defined in TABLE I. *First*, since the false positives satisfy the binomial distribution, we can calculate the posterior probability α_v for each vertex in G_m^* based on Bayes' theorem [8] (lines 1 - 3). *Second*, we know that a vertex should have only one (true) precursor. Therefore, we can calculate a correction value β_v^1 with the backward information (i.e., the red paths in Figure 6a) from G_m^{1*} (lines 4 - 6). *Third*, we observe that a vertex must be true positive if one of its descendants is truly positive. Therefore, vertices with more descendants are more likely to be true positives. We can also calculate another correction value

TABLE I: Notations used in the decoding algorithm

Notation	Definition
$n(v)$	The number of neighbor of a vertex in G^s
$s(v)$	The number of siblings of a vertex
$out(v)$	The out-degree of a vertex
$in(v)$	The in-degree of a vertex
$p(v)$	The parent vertex of a vertex
$c(v)$	The child vertex of a vertex
ψ	The FPR of random response
α_v	The FPR of the edges that starting at a vertex
β_v	The true positive rate of a vertex
μ_v	The membership value of a vertex

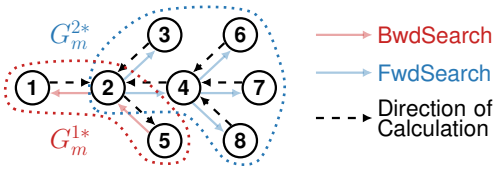
Algorithm 1: Decoding Algorithm

Input : $(G^s, \psi, G_m^{1*}, G_m^{2*})$
Output: $\{(v, \mu_v) | v \in V_m\}$

1 **for** $b \in \{1, 2\}$ **do**
2 **for** $v \in V_m^{b*}$ **do**
3 $\alpha_v^b = \sum_{i=0}^{out(v)} \left(\frac{i}{out(v)} \cdot \frac{\binom{n(v)}{i}\psi^i(1-\psi)^{n(v)-i}}{\sum_{j=0}^{out(v)}\binom{n(v)}{j}\psi^j(1-\psi)^{n(v)-j}}\right)$;
4 **for** $v \in V_m^{1*}$ **do**
5 **if** $out(v) = 0$ **then** $\beta_v^1 = \frac{1-\alpha_{p(v)}^1}{s(v)}$;
6 **else** $\beta_v^1 = 1 - \alpha_{p(v)}^1(1 - \max\{\beta_{v_i}^1 | v_i \in c(v)\})$;
7 **for** $v \in V_m^{2*}$ **do**
8 **if** $in(v) = 0 \wedge out(v) \neq 0$ **then** $\beta_v^2 = 1 - \prod_{i=0}^{out(v)}(1 - \beta_{c(v)}^2)$;
9 **else if** $in(v) \neq 0 \wedge out(v) = 0$ **then** $\beta_v^2 = 1 - \alpha_{p(v)}^2$;
10 **else if** $in(v) \neq 0 \wedge out(v) \geq 1$ **then** $\beta_v^2 = 1 - \alpha_{p(v)}^2 \prod_{i=0}^{out(v)}(1 - \beta_{c(v)}^2)$;
11 **for** $v \in V_m^*$ **do**
12 **if** $\beta_v^1 \neq 0 \wedge \beta_v^2 \neq 0$ **then** $\mu_v = 1 - (1 - \beta_v^1)(1 - \beta_v^2)$;
13 **else** $\mu_v = \max\{\beta_v^1, \beta_v^2\}$;

β_v^2 with the forward information (i.e., the blue paths in Figure 6a) from G_m^{2*} (lines 7 - 10). *Finally*, we integrate the above results to obtain the membership values μ_v (lines 11 - 13).

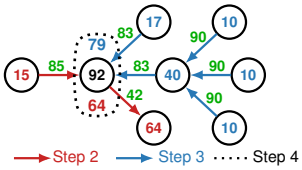
A Case Study. Figure 6 provides an example of calculating μ_v using our decoding algorithm. First, we compute α_v for the vertices' that are parents of other vertices (i.e., v_2, v_4, and v_5 in G_m^*) based on the number $n(v)$ and $n^*(v)$ of vertices' neighbors in G^s and G_m^*. Here, α_v represents the posterior probability that an edge is a false positive. Although v_4 has three neighbors in G_m^*, its α_v is high since it has many neighbors in G^s. Therefore, the noise dominates the value n_4^*, and the three out-degrees can all be false positives. Second, we compute β_v^1 and β_v^2 on G_m^{1*} and G_m^{2*} from leaves to root (i.e., dashed line in Figure 6a) respectively, which denote the FPR of a vertex. Because $(v_1, v_3, v_6, v_7, v_8)$ are leaves, their membership value is only determined by the incident edges, which equals $1 - a_{p(v)}$. Next, β_2 and β_4 are determined by their children and incident edges. Specifically, $\beta_5^2 = \beta_2^2$ since v_5 has only one children and no parent in G_m^{1*}, i.e., the reporting vertex. Finally, we integrate the values of v_2 in step 4, because it exists in G_m^{1*} and G_m^{2*}. Meanwhile, the other vertices' membership value equals to β_v^1 or β_v^2. The results show that v_2 has the highest membership value in the graph.



(a) Illustration of G_m^{1*} and G_m^{2*}

Graph	Vertex	$n(v)$	$n^*(v)$	$\alpha_v(\%)$
G_m^{1*}	v_2	50	1	85
	v_2	50	2	83
G_m^{2*}	v_4	100	3	90
	v_5	10	1	42

(b) Step 1: Caculate α_v



(c) Step 2-4: Caculate β_v^1, β_v^2, and μ_v

Fig. 6: Example of decoding on G_m^*, where $\psi = 10\%$. (c) The blue and red labels of vertices denote β_v^1 and β_v^2 in percentage. The green labels denote α_v of vertices.

IV. Protocol Analysis

In this section, we analyze our proposed scheme and prove it satisfies impact tracing's goals for security and privacy.

A. Security Analysis

The security of our scheme includes three aspects: confidentiality, accountability, and deniability.

Confidentiality. We consider three types of confidentiality.

Platform confidentiality implies that the platform cannot know the content and forwarding graph of a message before it is reported. During the messaging phase, the platform only observes the ciphertexts of the E2EE and the encryption Λ. Therefore, the platform confidentiality is reduced to the semantic security of E2EE and Λ.

Tag server confidentiality ensures that the tag server remains unaware of the messages' content, regardless of whether the message is reported. In our design, the tag server receives the message tags from the sender and the derived tracing key dtk from the platform. The PRF's pseudorandomness ensures these tags reveal nothing to the tag server, and dtk equals to a random bit string since the psk is only known to the platform and the block cipher Σ is a PRP.

User confidentiality prevents the recipients from learning the forwarding path of the received messages. A recipient receives the ephemeral key, tag key, and message tag in messaging, where the ephemeral key is a random bit string. Without the knowledge of other users' tracing keys (that is the block cipher's encryption keys), the recipient cannot link the tag keys that reflect the users on the forwarding path. Therefore, user confidentiality is maintained, as none of the forwarding paths is disclosed to the recipient. The formal definition is provided in Appendix B, with the proof detailed in the full version.

Accountability. Accountability prevents malicious users from manipulating the tracing result to smear an honest user or evade tracing. Four circumstances are required to be dealt with in our security analysis. Specifically,

- *Identity replacement.* Malicious user A_1 sends message m to malicious user A_2. Then A_2 or A_1 successfully reports the path $U_1 \xrightarrow{m} A_2$ or $A_1 \xrightarrow{m} U_1$, where U_1 is an honest user that does not send or receive the message m.
- *Message replacement.* Honest user U_1 first sends message m to a malicious user A_1 and forms a path $U_1 \xrightarrow{m} A_1$. Then A_1 successfully reports the path $U_1 \xrightarrow{m_1} A_1$, where m_1 is different to the actual message m.
- *Edge replacement.* Honest user U_1 receives a message m

from two malicious users A_1 and A_2. Then, U_1 forwards the message received from A_1 to another malicious user A_3. This forms two forwarding paths, i.e., $A_1 \xrightarrow{m} U_1 \xrightarrow{m} A_3$ and $A_2 \xrightarrow{m} U_1$. Finally, A_2 or A_3 successfully reports the path $A_2 \xrightarrow{m} U_1 \xrightarrow{m} A_3$, where U_1 actually forwards m received from A_1.

- *Path partition.* Honest user U_1 receives a message m from a malicious user A_1 and then forwards it to another malicious user A_2, i.e., $A_1 \xrightarrow{m} U_1 \xrightarrow{m} \mathrm{A}_2$. The adversary wins when the traced path is $A_1 \xrightarrow{m} U_1$ or $U_1 \xrightarrow{m} \mathrm{A}_2$, i.e., the path is partitioned at the honest user U_1.

As shown in Figure 4, the processed message tag tag' serves as a 'commitment' that binds the message, the tag key, and the tracing key. By carefully analyzing the above circumstances, we can reduce them to the collision resistance of the PRF and hash function, the block cipher's security, and the encryption's RKR. First, identity replacement results in a hash collision since dtk is derived from the tracing key, which represents the communication parties' identity and is controlled by the platform. Second, message replacement requires finding a PRF collision that outputs the same with two different input messages. Third, all possible cases in edge replacement can be reduced to message or identity replacement, unless a collision occurs in the block cipher's output. Nevertheless, this collision is impossible because the block cipher is a permutation. Finally, our scheme is secure against the path partition attack because the message tag is calculated or verified by an honest user, which binds the neighboring users. We provide a detailed definition in Appendix C to capture this accountability with proof in the full version.

Continuing accountability. In the messaging phase, malicious senders may send packets that contain malformed tracing metadata to recipients. Since the tag server stores message tags before the recipients' verification, rejecting the malformed packets creates inconsistent views between the tag server and the recipients. This inconsistency also breaks accountability, allowing malicious users to smear an honest user or evade tracing. We note that this problem has been neglected in existing schemes [50], [27]. In Appendix A-A, we demonstrate the problem using Tyagi et al.'s scheme [50], which is of independent interest.

Letting the platform revoke all the rejected packets is a possible solution to the above problem. But, this allows a malicious recipient who rejects all the received packets to evade tracing in the future. More importantly, the platform cannot distinguish whether the recipient is malicious in messaging since it cannot verify the tracing metadata without additional information. Therefore, in the messaging phase, we must introduce an additional mechanism for verifying rejected packets, detailed in Appendix A-B. Here, our insight is letting both the sender and recipient of a rejected packet submit information to the platform. Because one of the users must be honest, at least one of two keys from the sender and recipient can successfully decrypt the packet; otherwise, a pair of malicious users can evade tracing trivially. With the decrypted tracing metadata, the platform can redo the tag verification as the recipient, which allows it to check the rejected packets' correctness.

Remark 4. (Deniability) Deniability guarantees only the platform can confirm the tracing result, i.e., who sent/received a particular (reported) message. For example, in the real world, a whistleblower may use EEMSs to communicate with others, and deniability allows them to deny their activities. Similar to prior work [49], [50], we achieve deniability by allowing an entity to forge tracing metadata and messages indistinguishable from real ones. Specifically, in our scheme, the platform can forge any user's message sending/forwarding and receiving since it holds all the users' identities and identity keys.

B. Differential Privacy Analysis

We now demonstrate that our scheme provides differential privacy guarantees for the tracked users, especially the non-influential users. Specifically, to match the design goals of impact tracing, we propose a new customized differential privacy, i.e., individualized asymmetric subtree differential privacy (IAS-DP).

To accommodate the design goals of impact tracing, IAS-DP differs from conventional DP [13] in two aspects. First, individualized DP [24] allows different users to have different privacy budgets since impact tracing requires diversified privacy protection for other users according to their influence. Conversely, conventional DP requires all the users to share the same privacy budget, leading to over-protecting influential spreaders. In our scheme, while the servers choose the FPR, a user's privacy budget varies depending on its position in the forwarding graph, which reflects its impact. Second, asymmetric DP [45] mitigates the imbalance between false positives and negatives. In contrast, conventional DP inevitably introduces two-sided noises (i.e., false positives and negatives). Specifically, consider a forwarding graph as a directed tree with the root representing the message originator. Introducing false negatives in such a tree would prematurely halt traceback, potentially allowing influential spreaders to escape. Consequently, two-sided noise does not apply to the scenario of message traceback.

We formalize fuzzy message traceback that lets the platform learn a noisy graph as follows. Given a private social graph G and a subgraph $G_m \in G$, the algorithm outputs a noisy graph G_m^* such that $G_m \in G_m^*$. Both the graphs G_m^* and G_m can be viewed as trees that are rooted in the report vertex. The differential privacy requires that a vertex in the input G_m does not affect the final output G_m^* when it has limited influence in G_m. To capture this requirement in IAS-DP, we define *neighboring graphs* as two graphs that differ in a subtree $tree(v)$. Here, $tree(v)$ is a subtree rooted in a vertex v that includes all its parents and descendants. For instance, each user in a forwarding graph is the root of their descendants, and the messages propagate through paths from the root to its descendants. Therefore, a user's influence in spreading a message can be measured by the size of $tree(v)$. Formally, we define the subtree-based neighboring graph as follows, and an example is presented in Figure 7.

Definition 1 (Subtree-based neighboring graph). *Given a graph $G = (V, E)$, we say a graph $G' = (V', E')$ is a neighboring graph of G if $G \oplus G' = tree(v)$, where $G \oplus G'$ denotes the difference between the two graphs.*

Now, we define the IAS-DP on two subtree-based neighboring graphs G_m and G'_m that differ only in $tree(v)$.

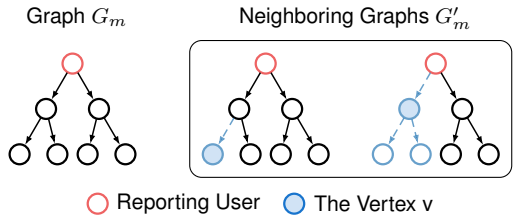


Fig. 7: Example of subtree-based neighboring graphs, where the $tree(v)$ consists of blue vertices and edges.

Definition 2 (ε_v-individualized asymmetric subtree differential privacy). *A randomized algorithm $\mathcal{M}$ is ε_v-IAS-DP if given a graph G the following equation holds for any $S \in Range(\mathcal{M})$ and neighboring subgraph pair (G_m, G'_m), where G'_m is obtained by removing a subtree $tree(v)$ in G_m.*

$$\Pr[\mathcal{M}(G_m) = S] \leq e^{\varepsilon_v} \cdot \Pr[\mathcal{M}(G'_m) = S], \tag{1}$$

where ε_v is the privacy budget for vertex v.

The following theorem formally proves that our scheme provides DP guarantees for the tracked users.

Theorem 1. *The subgraph publication algorithm (i.e., fuzzy message traceback) satisfies ε_v-individualized asymmetric subtree differential privacy, where $\varepsilon_v = ln(\frac{1}{\psi^n})$, n is the number of edges in $tree(v)$, and ψ is the FPR of random response.*

Proof: Let $G_m = (V_m, E_m)$ and $G'_m = (V'_m, E'_m)$. Since $G'_m \subseteq G_m$ and the RR mechanism only introduce false positives, for any $S \in Range(\mathcal{M})$, we have two cases.

- $G'_m \subseteq S \subset G_m$: In this case, the Equation (1) holds because $\Pr[\mathcal{M}(G_m) = S] = 0$.
- $G'_m \subseteq G_m \subseteq S$: Let $S = (V, E)$, $\widetilde{E'_m} = E - E'_m$, and $\widetilde{E_m} = E - E_m$, then we have

$$\Pr[\mathcal{M}(G'_m) = S] = \prod_{e \in \widetilde{E'_m}} \Pr[e \text{ is false positive}];$$

$$\Pr[\mathcal{M}(G_m) = S] = \prod_{e \in \widetilde{E_m}} \Pr[e \text{ is false positive}].$$

Note that G_m and G'_m only differ in $tree(v)$, thus $E_m - E'_m$ denote the edge set in $tree(v)$. We say $tree(v)$ is false positive if and only if all edges in the $tree(v)$ are false positives, i.e., caused by the RR mechanism.

$$\begin{aligned}
&\Pr[\mathcal{M}(G'_m) = S] \\
&= \Pr[\mathcal{M}(G_m) = S] \cdot \prod_{e \in \widetilde{E'_m}} \Pr[e \text{ is false positive}] \\
&= \Pr[\mathcal{M}(G_m) = S] \cdot \Pr[tree(v) \text{ is false positive}].
\end{aligned}$$

Let n be the number of edges in $tree(v)$. Each edge $e \in tree(v)$ becomes the false positive only if the coin flipped by the tag server is 1, which has a probability ψ. Then,

$$\Pr[tree(v) \text{ is false positive}] = \psi^n.$$

The above equation holds since the FPR of each edge is independent. Hence, we have

$$\frac{\Pr[\mathcal{M}(G_m) = S]}{\Pr[\mathcal{M}(G'_m) = S]} = \frac{\Pr[\mathcal{M}(G_m) = S]}{\Pr[\mathcal{M}(G_m) = S] \cdot \psi^n} = \frac{1}{\psi^n} \leq e^{\varepsilon_v}.$$

Therefore, given the false positive rate ψ, our scheme achieves differential privacy protection for vertex v with a privacy budget $\varepsilon_v = ln(\frac{1}{\psi^n})$. ■

Finally, executing a conventional DP algorithm multiple times on the same input may lead to the composition of the privacy budget, ultimately diminishing the privacy level. Our scheme avoids the composition for two reasons. First, the privacy loss caused by a trace is confined to a specific forwarding graph, as the forwarding graphs for different messages are independent. Second, within the same forwarding graph of a given message, all the edges are traced/queried only once (see multiple reports in Section III-B). Therefore, we only consider the privacy budget for one single trace.

V. Implementation and Evaluation

We implement our scheme in Rust[1] and all experiments are conducted on a PC with Intel Core i5-8500 CPU @ 3.00GHz and 32GiB of memory. We evaluate the performance of two real-world datasets of social networks.

- **College IM [36]** records 193 days of instant messages on an online community whose users are college students, which contains 1,899 vertices and 59,835 edges.
- **EU Email [37]** records 803 days emails of a research institution, which contains 986 vertices and 332,334 edges.

A. Implementation Details

We use KECCAK MAC derived from SHAKE256, SHA-3, AES-128, and AES-GCM-256 to implement PRF, hash function, block cipher, and symmetric encryption, respectively. Our implementation of these primitives utilizes the APIs offered by the Rust Crypto library [38], ensuring compatibility with existing EEMSs. Moreover, we implement $\mathsf{DB}_{\mathsf{ik}}$, $\mathsf{DB}_{\mathsf{nbr}}$, and $\mathsf{DB}_{\mathsf{tag}}$ with Redis that is an in-memory database.

Storage and runtime optimizing. We implement $\mathsf{DB}_{\mathsf{tag}}$ with a Bloom filter (BF), which supports efficient membership testing and storage. To prevent false positives, we set the FPR of BF to 1.0×10^{-9} (i.e., one-in-a-trillion), making false positives extremely unlikely. Furthermore, we implement the Trace algorithm as a breadth-first search (BFS) algorithm, where DB queries are batched, and the computations are paralleled.

Simulation. To simulate real-world scenarios, we generate the graphs and execute our scheme as follows. Figure 8 shows a visual simulation of the EU Email [29] dataset.

1) Simulate the dissemination of a message on the social network G using the susceptible-infected-recovered (SIR) model [4] with an infection rate of 0.05 and recovery rate of 0.6 [40], [41]. This step generates a forwarding graph G_m of the message.
2) Evaluate the vertices' influence in the forwarding graph G_m using the k-shell decomposition, and the results serve as the reference standard to assess our scheme.

[1] https://github.com/Ming-bc/impact-tracing

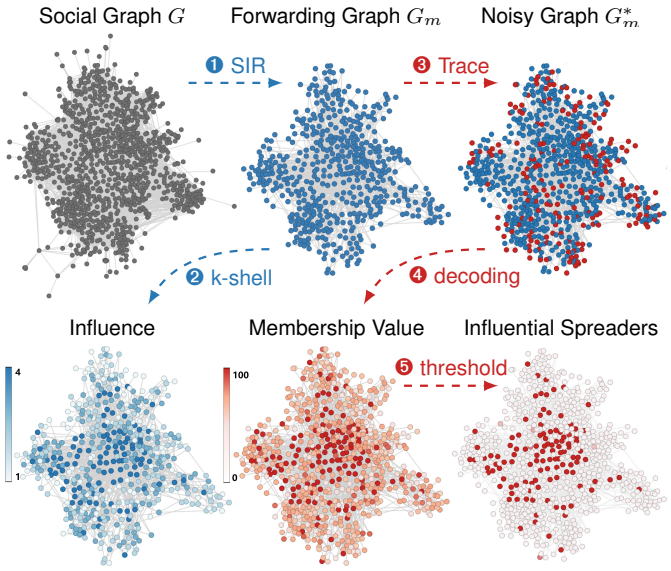


Fig. 8: Illustration of simulation steps and results on the EU Email dataset. (Top) Blue and red vertex are true and false positive, respectively. (Bottom) The left and middle graphs show the influence and membership value of vertices in G_m and G_m^*, respectively. The red vertices in the right graph are the output V_m^o of impact tracing.

3) Trace the forwarding graph G_m in the social network G using our impact tracing scheme, and obtain a noisy graph G_m^* from the Trace algorithm's output.
4) Compute the membership value μ_v of each vertex v in the graph G_m^* using the decoding algorithm in Section III-C.
5) Identify a set V_m^o of the vertices in the graph G_m^* with membership value beyond a threshold as influential spreaders.

Parameters. Two parameters dominate the performance:

- **Privacy budget ε_v:** determines the overall FPR ψ of the random response, thereby determining the number of noises (i.e., the number of false positives in G_m^*) introduced by the tag server. The relation between the privacy budget ε_v and the overall FPR ψ is presented in Theorem 1.
- **Output threshold:** determines the number of vertices in the output V_m^o. For example, when the threshold is zero, V_m^o contains all the vertices in G_m^*.

B. Metrics

We present three useful metrics: detection rate, output FPR, and interval FPR, which measure impact tracing's traceability, correctness, and privacy, respectively.

Traceability. Let $ks(v)$ be the k-shell value of a vertex v in G_m. We define the detection rate as

$$\textit{detection rate} = \frac{|\{v : v \in V_m^o \wedge ks(v) = k\}|}{|\{v : v \in V_m \wedge ks(v) = k\}|}.$$

This measures the percentage of vertices with shell value k in G_m that is output in V_m^o. Traceability requires that as the shell value k increases, the detection rate should be as large as possible, and vice versa. This reflects the goal of revealing only the influential spreaders while preserving the privacy of non-influential users.

Correctness. We define output false positive rate (FPR) as

$$\textit{output FPR} = \frac{|\{v : v \in V_m^o \wedge v \notin V_m\}|}{|\{v : v \in V_m^o\}|}.$$

The output FPR measures the percentage of false positives in the output V_m^o. This metric should be small enough to reflect the correctness that requires the output V_m^o to contain only true positives (i.e., actual message forwarders).

Privacy. The decoding algorithm calculates the membership value μ_v for each vertices $v \in V_m^*$. We split the range of μ_v into distinct intervals. Let I be one of the intervals. We define the interval false positive rate (FPR) as

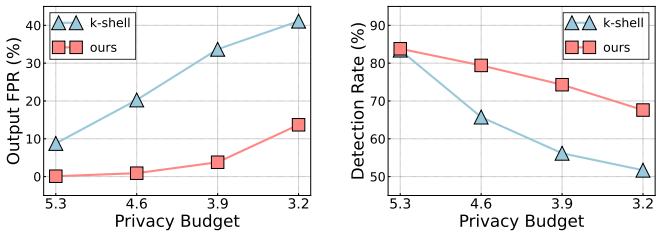
$$\textit{interval FPR} = \frac{|\{v : v \in V_m^* \wedge v \notin V_m \wedge u_v \in I\}|}{|\{v : v \in V_m^* \wedge u_v \in I\}|}.$$

The interval FPR quantifies the percentage of false positives to the vertices in G_m^* whose membership values lie within a specific interval I. The design of the decoding algorithm implies that vertices with high membership values, such as (0.99, 1.00], are likely to be influential spreaders. Conversely, vertices with low membership values (e.g., (0.1, 0.2]) are typically non-influential users. Hence, (individualized) privacy for impact tracing requires the interval FPR to be higher for intervals with lower membership values.

C. Utility and Privacy Analysis

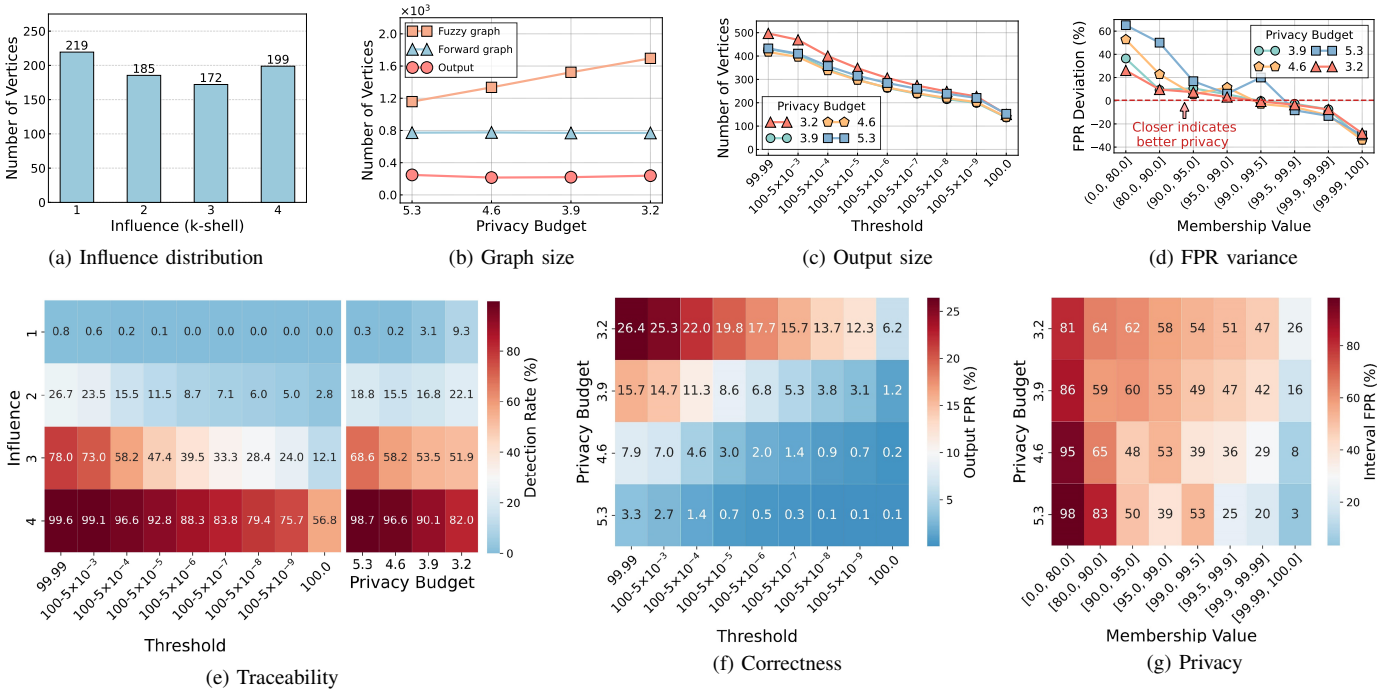
Figure 9, 10, and 11 present our experimental results on the real-world datasets under the above metrics. In the experiments, we follow [56] and set the privacy budget of the users who receive the message only once to $\varepsilon_v = \{5.30, 4.60, 3.91, 3.21\}$, corresponding to the overall FPR $\psi = \{0.5\%, 1\%, 2\%, 4\%\}$ in random response. Due to page limits, we present results for the College IM dataset here and defer results for the EU Email dataset to the full version.

Comparison with k-shell. We begin by comparing our decoding algorithm with the naive approach that decodes G_m^* with the k-shell decomposition. Figure 9a demonstrates the superiority of our decoding algorithm in minimizing the effect of false positives. For example, when the privacy budget is 5.3, the k-shell output contains 8.77% false positives, while ours is less than 0.1%. Moreover, as shown in Figure 9b, our decoding



(a) Correctness (lower is better) (b) Traceability (higher is better)

Fig. 9: Utility comparison with k-shell decomposition. The k-shell approach outputs vertices with the highest k-shell value in G_m^*. The threshold of ours is $100 - 5 \times 10^{-8}$.



(a) Influence distribution (b) Graph size (c) Output size (d) FPR variance

(e) Traceability (f) Correctness (g) Privacy

Fig. 10: Utility and privacy evaluation of impact tracing. (b) The threshold of output is $100 - 5 \times 10^{-8}$. (e) Left: the overall FPR is 1%; right: the threshold is $100 - 5 \times 10^{-4}$.

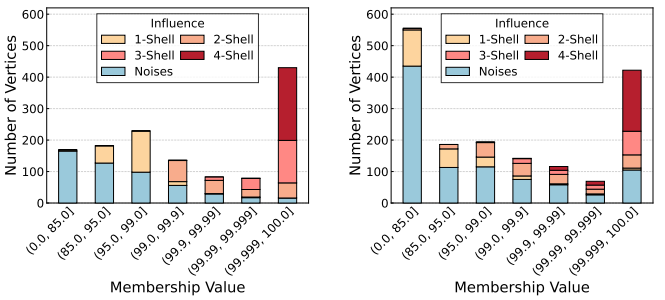
algorithm also has better traceability, especially when there is more noise in G_m^*.

Graph and output size. Figure 10a reports the influence distribution in G_m with an average of 773 vertices. These vertices are divided into four shells by the k-shell decomposition, and we say that a vertex located in the k-shell of G_m has k influence. Figure 10b presents the number of vertices in the forwarding graph G_m, noisy graph G_m^*, and output V_m^o. It shows that our scheme outputs approximately 30% of the vertices in G_m and hides the others with false positives. Furthermore, Figure 10b and 10c demonstrate that the threshold dominates the output size instead of the privacy budget since our decoding algorithm minimizes noise interference when calculating the membership values.

Traceability. Figure 10e presents how traceability (detection rate) varies with threshold and privacy budget. First, our scheme outputs most of the influential spreaders ($> 80\%$) with the highest k-shell value (i.e., 4-shell vertices). For example, when the threshold is $100-5\times10^{-4}$, our scheme identifies the most influential spreaders from 82% to 99% as the amount of noise varies. Second, in most cases, our scheme outputs only a few of the least influential users ($< 1\%$). Here, the least influential users are the 1-shell vertices of G_m, which contain all users who either receive or forward the message only once (i.e., non-influential users with no doubt).

Correctness. Ideally, the output V_m^o should contain no false positives, i.e., output FPR equals zero. But, V_m^o may contain false positives because of the additional noises. As shown in Figure 10f, correctness improves as the threshold and the privacy budget increase. The optimal correctness appears in the bottom right corner of the figure, corresponding to the maximum threshold and privacy budget.

Privacy. Figure 11 shows that the users in G_m with higher influence always have higher membership values in G_m^*, indicating that the platform can identify them more easily. This is consistent with our individualized DP, i.e., different users have different privacy budgets. Conversely, when all the users share the same privacy budget, the platform will have the same probability of identifying users with different influences. However, this would defeat the goal of traceability. Furthermore, Figure 10g and 11 demonstrate that false positives are more uniformly distributed on the membership value intervals when the privacy budget is lower. When considering privacy only, the platform should not be able to differentiate between true and false positives based on their membership value.



(a) Privacy Budget = 5.3 (b) Privacy Budget = 3.2

Fig. 11: Individualized privacy for the tracked users with different influences. The noises are false positives in G_m^*.

This requires the false positives to be uniformly distributed across all membership value intervals. Specifically, Figure 10d presents the variance between the interval FPR and the ideal FPR of G_m^*, where ideal FPR equals the ratio of false positives to all vertices in G_m^*. The horizontal line 0 indicates the two FPRs are perfectly matched; thus, the closer the curves to line 0, the better the privacy.

The trade-off. In summary, utility (that includes correctness and traceability) and privacy form a dilemma. With suitable parameters, our scheme can efficiently navigate these properties' trade-offs. According to Figure 10, when the privacy budget = 5.3 and threshold = $100 - 5 \times 10^{-8}$, our scheme identifies 84% of the most influential spreaders and no the least influential users with 99.9% correctness (i.e., less than 0.25 false positives on average). Meanwhile, other users are hidden by at least 20% false positives. Furthermore, the detection rate of the most influential spreaders boosts to 95.6% when the threshold = $100 - 5 \times 10^{-5}$.

D. Cost Analysis

Bandwidth and Storage Overhead. We first compare the overheads between our scheme and other existing schemes in TABLE II. In our design, the users hold tracing keys while the server holds the encrypted tracing metadata. Conversely, a source tracing scheme such as PEB21 lets the users hold the tracing metadata in end-to-end encrypted ciphertexts while the platform holds the key. As a result, PEB21 incurs 3.4× higher bandwidth and 10× client storage than ours, although eliminating extra platform storage.

Compared to the message traceback scheme TMR19, our scheme reduces 94% platform, 53% client storage, and 5% bandwidth overhead. The storage improvement stems from using the encryption chain and bloom filter. The encryption chain implicitly links the edges in the same forwarding path, thus amortizing the costs of identity keys to all the messages. In contrast, TMR19 employs an unidirectional encrypted pointer $C_k = \mathsf{E}_{k_t}(k_{t-1})$ to link two consecutive tag keys explicitly. This introduces an additional storage overhead that is linear in the number of messages. For instance, to handle the daily volume of messages on WhatsApp (i.e., 100 billion [44]), our scheme requires only 0.54 TiB platform storage, while TMR19 needs 9.4 TiB.

Runtime. Next, we analyze the additional latency of delivering a single message. Given all the cryptographic primitives in our scheme are symmetric and lightweight, delivering a 1 kB message spends less than 0.3 ms (send: 13.4 µs; platform process: 232.4 µs; receive: 13.6 µs) when combined with the double ratchet algorithm [3]. The only time-consuming operation in messaging is that the platform needs to query the $\mathsf{DB}_{\mathsf{ik}}$ for a user's identity key during processing. Except for that, all the other operations take less than 10 µs. In addition, the messaging latency is independent of the message size since the platform only calculates the hash values of messages during the messaging phase.

We finally benchmark the runtime of tracing the graphs generated from the College IM dataset with the FPR $\psi = 1\%$. The experimental results show that our scheme traces a graph with 4,000 edges under 13s. Fortunately, tracing is not latency-sensitive work, and it would not be executed frequently. Therefore, our performance is sufficient for real-world deployment.

TABLE II: Bandwidth and storage per message (byte)

Tracing Policy	Schemes†	Bandwidth		Storage	
		$\mathcal{S}$ - $\mathcal{P}$	$\mathcal{P}$ - $\mathcal{R}$	$\mathcal{C}$	$\mathcal{P}$
Source Tracing	PEB21 [39]	256	320	160	-
	LRTY22 [30]	243	243	243	-
	IAV22 [21]	380	484	380	-
Message Traceback	TMR19 [50]	96	80	34	104
	KTW22 [27]	203	203	16	136
Impact Tracing	Ours	96‡	72	16	6

$\mathcal{S}$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{C}$: Sender, recipient, platform, and client.
†: We choose the unlinkable, path and tree traceback scheme in PEB21, KTW22, and TMR19, respectively;
‡: Packets to the platform and tag server: 72 bytes and 24 bytes.

VI. Related Works

Existing works are classified into three categories to solve the traceability problem in EEMSs.

Tracing policy. Some references aim to balance traceability and privacy, which is also our focus. Apart from impact tracing, all existing works fall into message traceback [50], [27] or source tracing [39], [21], [30], [7], [9]. Message traceback is impractical in the real world since it over-compromised users' privacy. Source tracing adheres more closely to real-world regulations [42], [33], but it may not work as intended in real-world scenarios. For instance, consider a user forwarding a message from another platform, source tracing alone cannot reliably identify the actual originator. Moreover, restricting the originator's actions to prevent misinformation raises doubts about its effectiveness, as malicious users can easily create new accounts to disseminate problematic content.

Malicious reporting. In practice, users may intentionally submit reports with harmless messages, allowing a malicious platform to trace innocent users. This misuse of reporting undermines the motivation of traceability. To mitigate this problem, Liu et al. [30] and Bell [9] et al. proposed fuzzy and accurate threshold reporting schemes, respectively. These schemes allow the platform to trace only messages reported to exceed a (fuzzy) threshold but cannot resist the colluding between the platform and recipients. Therefore, Bartusek et al.[7] designed a protocol that restricts the platform to trace a message only within a blocklist, which preserves users' privacy even in the presence of a colluding platform and recipients.

Metadata private. Another line of work aims to enable traceability on metadata-private messaging systems. Kenny et al.[27] extended the path traceback scheme [50] to achieve anonymous path traceback and source tracing. Issa et al.[21] introduced Hecate, which offers source tracing and message franking under an anonymous setting. Both schemes rely on a two-server setting, where Kenny et al. employ an extra server to store users' identities; Issa et al. involve an additional moderator to issue anonymous tokens to users before messaging.

VII. Conclusion and Discussion

In this paper, we introduced impact tracing to bridge the gap between traceability proposals and real-world scenarios. Our impact tracing scheme not only protects the non-influential users' privacy but also identifies the influential users using our novel decoding algorithm. We presented a security analysis of our design and quantified its privacy level using differential privacy. Experimental results show that our scheme identifies the most influential spreaders ($> 82\%$) while avoiding non-influential spreaders ($< 1\%$) and false positives ($< 1\%$). Compared to the current state-of-the-art message traceback scheme [50], our scheme requires $17.3\times$ and $2.1\times$ smaller platform and client storage respectively.

Future work. Here, we outline two prospective directions:

Misidentification. While noise protects privacy, it can lead to misidentification, which seems inevitable for impact tracing since the noises are introduced to protect non-influential users' privacy. In our scheme, these misidentified users are still influential, albeit not the most influential ones. We post an interesting open problem on designing an impact tracing solution without misidentification.

Group messaging. Due to the ease of sharing messages in group chat, problematic messages proliferate in it [32]. Our scheme can be extended for group messaging by treating group messages as multiple two-party messages at the expense of linear overheads to the group size. It also remains open to designing an impact tracing solution that supports group messaging with sub-linear overheads.

Practical deployment and ethical consideration. From the technical perspective, existing EEMSs can readily adopt our scheme. First, our scheme treats the underlying EEMS as a black box, allowing it to be adopted by any non-anonymous EEMS. Second, the cryptographic tools utilized in our scheme are already implemented in EEMSs, such as Signal, eliminating the need for additional tool development.

However, deploying impact tracing in EEMSs extends beyond mere technical considerations. Currently, there are two extremes to deploying content moderation within EEMSs. One extreme is forbidding all content moderation approaches to unconditionally protect user privacy, inadvertently allowing the dissemination of problematic messages. The other extreme permits platforms to trace or detect problematic messages without any limitations; this capability enables platforms to monitor specific messages, resulting in the risk of mass surveillance. Our work explores the technical feasibility of a balanced approach between these extremes. But, it is imperative to consider additional non-technical factors. For example, new content moderation legislation may conflict with existing privacy laws such as the General Data Protection Regulation (GDPR) [15]. Designing a protocol that complies with contradictory legal regulations is technically infeasible. Therefore, we emphasize that any content moderation scheme must be meticulously evaluated for its ethical implications on all stakeholders before practical deployment.

Acknowledgments

The authors thank anonymous reviewers for their constructive reviews. We also thank Peng Wang for his insights on the design of the decoding algorithm.

This work was supported by the National Key R&D Program of China under Grant 2022YFB3103500, the National Natural Science Foundation of China under Grants U20A20176, 62072062, and 62472056, the Natural Science Foundation of Chongqing, China, under Grant CSTB2022NSCQ-MSX0582. In addition, Guomin Yang was supported by the Lee Kong Chian Fellowship awarded by Singapore Management University, and Robert H. Deng was supported by the AXA Research Fund.

References

[1] J. Aas and T. Geoghegan, "Introducing isrg prio services for privacy respecting metrics," https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio, Nov. 2020.

[2] I. Ahmad, Y. Yang, D. Agrawal, A. El Abbadi, and T. Gupta, "Addra: Metadata-private voice communication over fully untrusted infrastructure," in *Proc. of USENIX Symposium on Operating Systems Design and Implementations (OSDI)*, 2021.

[3] J. Alwen, S. Coretti, and Y. Dodis, "The double ratchet: security notions, proofs, and modularization for the signal protocol," in *Proc. of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019, pp. 129–158.

[4] R. M. Anderson and R. M. May, *Infectious diseases of humans: dynamics and control*. Oxford university press, 1992.

[5] E. Bakshy, I. Rosenn, C. Marlow, and L. Adamic, "The role of social networks in information diffusion," in *Proc. of International World Wide Web Conferences (WWW)*, 2012.

[6] L. Barman, M. Kol, D. Lazar, Y. Gilad, and N. Zeldovich, "Groove: Flexible metadata-private messaging," in *Proc. of USENIX Symposium on Operating Systems Design and Implementations (OSDI)*, 2022.

[7] J. Bartusek, S. Garg, A. Jain, and G. Policharla, "End-to-end secure messaging with traceability only for illegal content," in *Proc. of Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2023.

[8] T. Bayes, "Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[9] C. Bell and S. Eskandarian, "Anonymous complaint aggregation for secure messaging," in *Proc. of The Annual Privacy Enhancing Technologies Symposium (PETS)*, 2024.

[10] O. Board, "How we do our work," https://www.oversightboard.com/our-work/, 2024.

[11] E. Commission, "The digital services act," https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32022R2065, Oct. 2022.

[12] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2018.

[13] C. Dwork, "Differential privacy," in *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12.

[14] S. Englehardt, "Next steps in privacy-preserving telemetry with prio," https://www.abetterinternet.org/post/introducing-prio-services/, June 2019.

[15] C. Europe, "Cdt europe feedback on the belgian presidency's proposal on the regulation on child sexual abuse material," https://cdt.org/wp-content/uploads/2024/06/May-2024-CDT-Europe-Feedback-on-the-Belgian-Presidencys-compromise-on-the-Regulation-on-Child-Sexual-Abuse-Material.pdf, May. 2024.

[16] P. Farshim, C. Orlandi, and R. Rosie, "Security of symmetric primitives under incorrect usage of keys," *IACR Transactions on Symmetric Cryptology*, vol. 2017, no. 1, p. 449–473, Mar. 2017.

[17] C. for Countering Digital Hate, "The disinformation dozen: Why platforms must act on twelve leading online anti-vaxxers," https://counterhate.com/research/the-disinformation-dozen/, Mar. 2021.

[18] S. Frenkel, "The most influential spreader of coronavirus misinformation online," https://www.nytimes.com/2021/07/24/technology/joseph-mercola-coronavirus-misinformation-online.html, July 2021.

[19] Google, "Addendum to the analytics in exposure notifications express: Faq," https://github.com/google/exposure-notifications-android/blob/master/doc/enexpress-analytics-faq-addendum.md, Oct. 2021.

[20] P. Grubbs, J. Lu, and T. Ristenpart, "Message franking via committing authenticated encryption," in *Proc. of Annual International Cryptology Conference (CRYPTO)*, 2017.

[21] R. Issa, N. Alhaddad, and M. Varia, "Hecate: abuse reporting in secure messengers with sealed sender," in *Proc. of Usenix Security Symposium (Usenix Security)*, 2022.

[22] C. Jack, "Lexicon of lies: Terms for problematic information," https://datasociety.net/library/lexicon-of-lies/, Aug. 9 2017.

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

[24] Z. Jorgensen, T. Yu, and G. Cormode, "Conservative or liberal? personalized differential privacy," in *Proc. of IEEE International Conference on Data Engineering (ICDE)*, 2015.

[25] J. Kastrenakes, "Whatsapp limits message forwarding in fight against misinformation," https://www.theverge.com/2019/1/21/18191455/whatsapp-forwarding-limit-five-messages-misinformation-battle, Jan. 21 2019.

[26] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. of ACM Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[27] E. Kenney, Q. Tang, and C. Wu, "Anonymous traceback for end-to-end encryption," in *Proc. of European Symposium on Research in Computer Security (ESORICS)*, 2022.

[28] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse, "Identification of influential spreaders in complex networks," *Nature physics*, vol. 6, no. 11, pp. 888–893, 2010.

[29] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *TKDD*, vol. 1, no. 1, p. 2, 2007.

[30] L. Liu, D. S. Roche, A. Theriault, and A. Yerukhimovich, "Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS)," in *Proc. of ISOC Network and Distributed System Security Symposium (NDSS)*, 2022.

[31] S. Loomba, A. de Figueiredo, S. J. Piatek, K. de Graaf, and H. J. Larson, "Measuring the impact of covid-19 vaccine misinformation on vaccination intent in the uk and usa," *Nature human behaviour*, vol. 5, no. 3, pp. 337–348, 2021.

[32] C. Machado, B. Kira, V. Narayanan, B. Kollanyi, and P. Howard, "A study of misinformation in whatsapp groups with a focus on the brazilian presidential elections." in *Proc. of International World Wide Web Conferences (WWW)*, 2019.

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

[34] J. Millican, "Challenges of E2E encryption in facebook messenger," in *Proc. of Real World Crypto Symposium (RWC)*, 2017.

[35] L. Muchnik, S. Pei, L. C. Parra, S. D. Reis, J. S. Andrade Jr, S. Havlin, and H. A. Makse, "Origins of power-law degree distribution in the heterogeneity of human activity in social networks," *Scientific reports*, vol. 3, no. 1, p. 1783, 2013.

[36] P. Panzarasa, T. Opsahl, and K. M. Carley, "Patterns and dynamics of users' behavior and interaction: Network analysis of an online community," *JASIST*, vol. 60, no. 5, pp. 911–932, 2009.

[37] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM)*, 2017, p. 601–610.

[38] A. Pavlov and T. Arcieri, "Rust crypto," https://github.com/RustCrypto, 2016.

[39] C. Peale, S. Eskandarian, and D. Boneh, "Secure complaint-enabled source-tracking for encrypted messaging," in *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2021.

[40] G. Resende, P. Melo, J. CS Reis, M. Vasconcelos, J. M. Almeida, and F. Benevenuto, "Analyzing textual (mis) information shared in whatsapp groups," in *Proc. of ACM Web Science Conference (WebSci)*, 2019.

[41] G. Resende, P. Melo, H. Sousa, J. Messias, M. Vasconcelos, J. Almeida, and F. Benevenuto, "(mis)information dissemination in whatsapp: Gathering, analyzing and countermeasures," in *Proc. of The World Wide Web Conference (WWW)*, 2019.

[42] K. 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] F. Sharevski, A. Devine, E. Pieroni, and P. Jachim, "Folk models of misinformation on social media," *Proc. of ISOC Network and Distributed System Security Symposium (NDSS)*, 2023.

[44] M. 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.

[45] S. Takagi, F. Kato, Y. Cao, and M. Yoshikawa, "Asymmetric differential privacy," in *IEEE Big Data*, 2022.

[46] TechCrunch, "Whatsapp's new limit cuts virality of 'highly forwarded' messages by 70%," https://techcrunch.com/2020/04/27/whatsapps-new-limit-cuts-virality-of-highly-forwarded-messages-by-70/, April 2020.

[47] D. Thakur, G. Post, M. Knodel, E. Llansó, G. Nojeim, and C. 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.

[48] K. Thomas, D. Akhawe, M. Bailey, D. Boneh, E. Bursztein, S. Consolvo, N. Dell, Z. Durumeric, P. G. Kelley, D. Kumar *et al.*, "Sok: Hate, harassment, and the changing landscape of online abuse," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2021.

[49] N. Tyagi, P. Grubbs, J. Len, I. Miers, and T. Ristenpart, "Asymmetric message franking: Content moderation for metadata-private end-to-end encryption," in *Proc. of Annual International Cryptology Conference (CRYPTO)*, 2019.

[50] N. Tyagi, I. Miers, and T. Ristenpart, "Traceback for end-to-end encrypted messaging," in *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2019.

[51] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg, "Structural diversity in social contagion," *PNAS*, vol. 109, no. 16, pp. 5962–5966, 2012.

[52] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, 2018.

[53] WhatsApp, "More changes to forwarding," https://blog.whatsapp.com/more-changes-to-forwarding, July. 2019.

[54] ——, "Whatsapp privacy policy," https://www.whatsapp.com/legal/privacy-policy, Jan. 2021.

[55] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[56] X. Zhu, V. Y. F. Tan, and X. Xiao, "Blink: Link local differential privacy in graph neural networks via bayesian estimation," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

APPENDIX A
CONSISTENT VERIFICATION ON TAGS

In this section, we first analyze the accountability problem exemplified by the tree traceback scheme proposed by Tyagi et al. [50] at CCS'19, then present our approach to mitigate this problem.

A. Analysis of Tyagi et. al.'s Scheme

At a high level, Tyagi et al. follow the framework of message franking we introduced in Section I-B. The main difference is that the tag is now a tracing metadata data tmd that contains the tag and a ciphertext that links the keys on the same forwarding chain (i.e., encrypt the previous key with the current key).

Background. We recall the framework of Tyagi et al. with the necessary details as follows:

1) To send a message m, a sender U_i generates a tag key k and corresponding tracing metadata tmd. Then, the sender sends (m, k) and tmd in end-to-end encrypted ciphertext and plaintext, respectively.
2) Upon receiving a message, the platform processes the tmd and stores the processed metadata tmd'. It then forwards encrypted (m, k) and tmd' to the recipient.
3) Once receiving (m, k, tmd'), the recipient verifies the tracing metadata and rejects the malformed packets.

In the protocol described above, the platform stores all the processed tracing metadata. This strategy introduces different views between the platform and recipient when dealing with packets containing malformed tracing metadata. In particular, while the recipients promptly reject such malformed packets, the platform persists in storing and utilizing these malformed packets for tracing.

Accountability problem. Next, we show how the malformed packets affect the accountability of Tyagi et al.. Specifically, we consider three colluding malicious users (A_1, A_2, A_3) and one honest user U_4, where the malicious users attempt to smear the honest user for receiving a rejected problematic message.

- First, in the messaging phase, the users interact as follows:
  1) A_1 originates a problematic message m_1 and send it to (A_2, A_3) along with well-formed tracing metadata tmd_1.
  2) Upon receiving (m_1, tmd_1), A_2 generates tracing metadata tmd_2 that corresponds to forwarding m_1 to U_4. Then, A_2 sends (m_2, tmd_2) to U_4, where m_2 is an innocent message.
  3) Once receiving (m_2, tmd_2), U_4 can easily detect that tmd_2 is malformed since it is generated for m_1. Therefore, U_4 rejects the packet. Meanwhile, the platform stores tmd_2 in storage since it cannot verify the tracing metadata and there is no feedback from the recipient U_4.
- Second, in the reporting-then-tracing phase, the adversary A_3 reports A_1 to the platform for sending problematic message m_1. Then, the platform traces the dissemination of m_1 as follows:
  1) Verify A_1 indeed sent m_1 to A_3.
  2) Trace to A_2 from A_1 via forward search.
  3) Trace to U_4 from A_2 since the tracing metadata tmd_2 is a well-formed tracing metadata of m_1, and identify U_4 as malicious for receiving the problematic message m_1.

Obliviously, the above tracing result contradicts the accountability defined in Tyagi et al. (i.e., "no colluding set of adversarial users can frame an honest user as having performed some action that they did not") since the malicious users (A_1, A_2, A_3) successfully smear U_4 for receiving the rejected problematic message m_1. In Tyagi et al., this problem stems from their security model (the $\mathsf{SendMal}$ algorithm, Figure 17), where they mark all rejected packets as not received by users. However, in their system, the packets are already stored by the platform before the users reject them. Therefore, the inconsistency between their system and security model causes this problem.

B. Our Solution

We finally describe our approach that allows the platform to verify the rejected packets' correctness and revoke the malformed packets. In detail, to verify and revoke a (malformed) packet, the entities in impact tracing interact as follows:

1) Once receiving a malformed packet, a recipient U_j submits a revoke request (pid, mk_r) to the platform, where mk_r is the $\mathsf{E2EE}$ key. If the packet contains a replicated tag key, the recipient additionally sends the previous message m_r that has the same tag key to the platform.
2) For a revoking request, the platform processes it as follows:
   a) Request the sender U_i to provide corresponding $\mathsf{E2EE}$ key mk_s of the packet with identity pid.
   b) Decrypt the packet $(pid, \mathsf{E2EE}(m, k_t, ek), c_t)$ using mk_r and mk_s, and revoke the packet if both keys fail to decrypt.
   c) Compute a tag as $tag^* \leftarrow \mathsf{TagGen}(m, k_t)$ and revoke the packet if $tag^* \neq tag$, where tag is decrypted from c_t. Note that the RKR guarantees that c_t has only one valid key.
   d) Check whether the processed tag $tag' \leftarrow \mathsf{TagProc}(dtk_{ij}, tag)$ exists in $\mathsf{DB}_{\mathsf{tag}}$, and revoke the packet if it exists.
   e) Send the pid of revoke packets to the tag server.
3) Finally, the tag server deletes the processed tags corresponding to the revoked packet identities from the platform.

Note that the steps (b-d) allow the platform to verify

the correctness of the rejected packet, which is the same as the tag verification made by the recipient when receiving a message. This avoids letting the platform distinguish which one of the communication parties is honest; thus, it guarantees the correctness of packets in storage.

APPENDIX B
USER CONFIDENTIALITY

User confidentiality ensures that recipients cannot learn additional information about a message's forwarding path from received packets. To capture the implication of user confidentiality, we define a security game $\mathrm{UConf}_{FT}^{\mathcal{A},b}$ in Figure 12. In the game, we consider an adversary $\mathcal{A}$ with capabilities equivalent to both the tag server and recipients, as we allow the two participants to collude. The adversary's goal is to distinguish the tag key from random bit strings. Besides, in the game, we define three oracles:

- $\mathcal{O}_{\mathsf{ik}}$ allows $\mathcal{A}$ to learn the colluding users' identity key.
- $\mathcal{O}_{\mathsf{dtk}}$ allows $\mathcal{A}$ to obtain the derived tracing key dtk between any pair of users, even the non-colluding ones.
- $\mathcal{O}_{\mathsf{send}}$ allows $\mathcal{A}$ to create communication between two users. Based on the random value b, this oracle either outputs random bit strings or real tracing metadata. To prevent trivial wins, we require the sender U_i to be a non-colluding user; otherwise, $\mathcal{A}$ can compute the message tag and tag key itself.

Definition 3 (User Confidentiality). *Let FT be a fuzzy message traceback scheme. We say FT achieves user confidentiality if, for any PPT adversary $\mathcal{A}$, $\mathcal{A}$ has a negligible advantage in $UConf_{FT}^{\mathcal{A},b}$. Specifically, $\mathcal{A}$'s advantage is defined as:*

$$\mathsf{Adv}_{FT}^{UConf}(\mathcal{A}) = \left|\Pr\left[UConf_{FT}^{\mathcal{A},1} = 1\right] - \Pr\left[UConf_{FT}^{\mathcal{A},0} = 1\right]\right|$$

Intuitively, a block cipher (i.e., pseudorandom permutation) Σ's with a secret tracing key tk_{ij} ensures that a tag key k_t^0 is indistinguishable from a random bit string k_t^1 from $\mathcal{A}$'s view. The tracing key tk_{ij} is derived from the identity key ik_i of non-colluding users $\mathcal{U}_{nc}$. Importantly, ik_i remains unobservable to the adversary $\mathcal{A}$, as it is generated by the platform and known only to the sender U_i and the platform itself. Moreover, the platform's secret key psk and the PRF's pseudorandomness guarantee that $\mathcal{A}$ gain no information about tk_{ij} from the derived key dtk_{ij}. As a result, the tag key reveals nothing about the forwarding path of any message to $\mathcal{A}$.

Theorem 2. *Let FT be the fuzzy message traceback scheme (defined in Section III) building on a block cipher Σ and pseudorandom function F. For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}$ and $\mathcal{C}$ such that:*

$$\mathsf{Adv}_{FT}^{UConf}(\mathcal{A}) \leq \mathsf{Adv}_{F}^{prf}(\mathcal{B}) + \mathsf{Adv}_{\Sigma}^{prp}(\mathcal{C}),$$

where $\mathsf{Adv}_{F}^{prf}(\mathcal{B})$ and $\mathsf{Adv}_{\Sigma}^{prp}(\mathcal{C})$ is the advantage that $\mathcal{B}$ and $\mathcal{C}$ breaks the pseudorandomness of F and the security of the block cipher Σ, respectively.

For a rigorous proof, please see the full version.

APPENDIX C
ACCOUNTABILITY

To capture the goal of accountability, we formalize it within the game $\mathrm{ACT}_{FT}^{\mathcal{A}}$ in Figure 13. In detail, at the beginning of

$\mathrm{UConf}_{FT}^{\mathcal{A},b}(1^\lambda)$

$(psk, \mathcal{L}_k) \leftarrow\$ \mathsf{KGen}(1^\lambda)$
$(\mathcal{U}_c, \mathcal{U}_{nc}) \leftarrow \mathcal{A}$
$b' \leftarrow\$ \mathcal{A}^{\mathcal{O}_{\mathsf{ik}},\mathcal{O}_{\mathsf{dtk}},\mathcal{O}_{\mathsf{send}}}(1^\lambda)$
return b'

$\mathcal{O}_{\mathsf{send}}(k_{t-1}, U_i, U_j)$

if $U_i \in \mathcal{U}_{nc}$: **return** $\perp$
if $(U_i, k_{t-1}) \in \mathcal{L}_k$: **return** $\perp$
$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(U_i, k_{t-1})\}$
$tk_{ij} \leftarrow H(\mathcal{L}_{ik}[U_i] || U_j)$
$k_t^0 \leftarrow \Sigma.\mathsf{E}_{tk_{ij}}(k_{t-1})$
$k_t^1 \leftarrow\$ \{0,1\}^\lambda$
return k_t^b

$\mathcal{O}_{\mathsf{ik}}(U_i)$

if $U_i \in \mathcal{U}_c$: **return** $\perp$
$ik_i \leftarrow \mathcal{L}_{ik}[U_i]$
return ik_i

$\mathcal{O}_{\mathsf{dtk}}(U_i, U_j)$

$ik_i \leftarrow \mathcal{L}_{ik}[U_i]$
$tk_{ij} \leftarrow H(ik_i, U_j)$
$dtk_{ij} \leftarrow \mathsf{DtkDer}(tk_{ij}, psk)$
return dtk_{ij}

Fig. 12: Security game for user confidentiality. $\mathcal{U}_c$ and $\mathcal{U}_{nc}$ denote the users who collude and do not collude with the tag server, respectively. $\mathcal{L}_{ik}$ and $\mathcal{L}_k$ store users' identity keys and received tag keys, respectively. The secret key psk and users' identity key in $\mathcal{L}_{ik}$ were randomly generated by the challenger.

the game, $\mathcal{A}$ outputs two sets $\mathcal{U}_{mal}$ and $\mathcal{U}_{hon}$, which represent the malicious users and the honest users, respectively. Note that this game is static, and we do not allow $\mathcal{A}$ to update $\mathcal{U}_{mal}$ and $\mathcal{U}_{hon}$ during the game. As in the real-world scenario, $\mathcal{A}$ obtains the identity keys of malicious users in $\mathrm{ACT}_{FT}^{\mathcal{A}}$, i.e., $\mathcal{L}_{ik}^m$. Then, we allow $\mathcal{A}$ to create communication between two arbitrary users through two oracles. Specifically,

- $\mathcal{O}_{\mathsf{HonSend}}$ allows $\mathcal{A}$ to create communication as an honest sender. Note that $\mathcal{L}_{ik}$, $\mathcal{L}_{tag}$, and $\mathcal{L}_k$ record identity keys, message tags, and received tag keys of users, respectively. Moreover, $\mathcal{L}_{ik}$ and $\mathcal{L}_{tag}$ correspond to $\mathsf{DB}_{\mathsf{ik}}$ and $\mathsf{DB}_{\mathsf{tag}}$ in our design, respectively.
- $\mathcal{O}_{\mathsf{MalSend}}$ allows $\mathcal{A}$ to create communication as a malicious sender, which can input arbitrary tag keys and message tags. However, when the recipient is honest, $\mathcal{O}_{\mathsf{MalSend}}$ prevents this attack by checking whether the input tag and reproduced tag match. In contrast, a malicious recipient can omit the tag verification.

In the oracles, the lists $\mathcal{L}_{ik}^h$, $\mathcal{L}_{tag}$, and $\mathcal{L}_k$ of honest users are unobservable to the adversary $\mathcal{A}$. Furthermore, the list $\mathcal{T}_q$ records the actual forwarding graph in the oracle queries, where each element in $\mathcal{T}_q$ is a tuple $(U_1, U_2, m, k, U_0, k_0)$. Here, U_1 is the sender, U_2 is the recipient, m is the forwarded message, k is the current tag key, U_0 is the precursor from whom m is received, and k_0 is the previous tag key from U_0. If the message is fresh, the tuple is defined by setting $U_0 =$ -, and k_0 is sampled from a uniform distribution. Based on the $\mathcal{L}_{tag}$ and $\mathcal{T}_q$, we can reconstruct the $\mathsf{DB}_{\mathsf{tag}}$, and $\mathsf{DB}_{\mathsf{nbr}}$ as in our design. Therefore, we can use the Trace algorithm to find all the forwarding paths of the message m^*, which is submitted by $\mathcal{A}$ for challenging.

We next consider the possible winning cases of the adversary and present the analysis in Figure 14. The task of

$\mathrm{ACT}^{\mathcal{A}}_{FT,n}(1^\lambda)$

$\mathcal{L}_{tag} \leftarrow \emptyset, \mathcal{T}_q \leftarrow \emptyset$
$(\mathcal{U}_{hon}, \mathcal{U}_{mal}) \leftarrow \mathcal{A}$
$(m^*, k^*, U^*, U_1, U_2, m, k_1, U_0, k_0)$
$\quad \leftarrow \mathcal{A}^{\mathcal{O}}(\mathcal{U}_{hon}, \mathcal{U}_{mal}, \mathcal{L}^m_{ik})$
$\mathcal{T}_{tr} \leftarrow \mathsf{Trace}(m^*, k^*, U^*)$
if $(U_1, U_2, m, k_1, U_0, k_0) \in \mathcal{T}_{tr}$ **do**
  / Identity, message or edge replacement
  if $(U_1, U_2, m, k_1, U_0, k_0) \notin \mathcal{T}_q$
    return 1
  / Path partition in backward search
  if $((U_0, k_0) = (\text{-}, \text{-})) \,\&(U_1 \in \mathcal{U}_{hon})$
    if $!\mathsf{IsSrc}(U_1, U_2, m, k_1, \mathcal{T}_q)$
      return 1
  / Path partition in forward search
  if $\mathsf{IsLeaf}(U_2, m, k_1, \mathcal{T}_{tr}) \& (U_2 \in \mathcal{U}_{hon})$
    if $\mathsf{HasFwd}(U_1, U_2, m, k_1, \mathcal{T}_q)$
      return 1
return 0

$\mathcal{O}_{\mathsf{HonSend}}(m, U_1, U_2, k_0, U_0)$

if $U_1 \notin \mathcal{U}_{hon}$ **then return** $\perp$
$ik_1 \leftarrow \mathcal{L}^h_{ik}[U_1]; \quad tk_{12} \leftarrow H(ik_1 || U_2)$
if $(k_0, U_0) = (\text{-}, \text{-})$
  $k_0 \leftarrow\!\!\$\ \{0,1\}^n; \quad k \leftarrow \mathsf{E}_{tk_{12}}(k_0)$
else $k \leftarrow \mathsf{E}_{tk_{12}}(k_0)$
$tag \leftarrow \mathsf{TagGen}(m, k, tk_{12})$
$dtk_{ij} \leftarrow \mathsf{DtkDer}(tk_{12}, psk)$
$\mathcal{L}_{tag} \leftarrow \mathcal{L}_{tag} \cup \{\mathsf{TagProc}(dtk_{ij}, tag)\}$
$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(U_2, k)\}$
$\mathcal{T}_q \leftarrow \mathcal{T}_q \cup \{(U_1, U_2, m, k, U_0, k_0)\}$
if $U_2 \in \mathcal{U}_{mal}$
  return (k, tag)
return $(\text{-}, \text{-})$

$\mathsf{HasFwd}(U_1, U_2, m, k_1, \mathcal{T})$

for $(U_1', *, m', *, U_0', k_0') \in \mathcal{T}$ **do**
  if $(U_0', U_1', m', k_0') = (U_1, U_2, m, k_1)$
    return true
return false

$\mathsf{IsFwd}(k_1, m, tk_{12}, U_1, U_2, \mathcal{T})$

for $(U_0, U_1, m, k_0, *, *) \in \mathcal{T}$ **do**
  $k_1' \leftarrow \mathsf{E}_{tk_{12}}(k_0)$
  if $k_1' = k_1$
    return (k_0, U_0)
return $(\text{-}, \text{-})$

$\mathcal{O}_{\mathsf{MalSend}}(k, m, tag_{ts}, tag_r, U_1, U_2)$

if $U_1 \notin \mathcal{U}_{mal}$ **then return** $\perp$
$ik_1 \leftarrow \mathcal{L}^m_{ik}[U_1]; \quad tk_{12} \leftarrow H(ik_1 || U_2)$
$dtk_{ij} \leftarrow \mathsf{DtkDer}(tk_{12}, psk)$
$\mathcal{L}_{tag} \leftarrow \mathcal{L}_{tag} \cup \{\mathsf{TagProc}(dtk_{ij}, tag_{ts})\}$
$\hat{tag} \leftarrow \mathsf{TagGen}(m, k, tk_{12})$
if $(U_2 \in \mathcal{U}_{hon}) \& (((U_2, k) \in \mathcal{L}_k) | (tag_r \neq \hat{tag}))$
  return $\perp$
$\mathcal{L}_k \leftarrow \mathcal{L}_k \cup \{(U_2, k)\}$
$\mathcal{T}_q \leftarrow \mathcal{T}_q \cup \{(U_1, U_2, m, k, \text{-}, \text{-})\}$
$(k_0, U_0) \leftarrow \mathsf{IsFwd}(k, m, tk_{12}, U_1, U_2, \mathcal{T}_q)$
if $(k_0, U_0) \neq (\text{-}, \text{-})$
  $\mathcal{T}_q \leftarrow \mathcal{T}_q \cup \{(U_1, U_2, m, k, U_0, k_0)\}$
return (k, tag)

$\mathsf{IsLeaf}(U_1, m, k_1, \mathcal{T})$

for $(*, *, m', *, U_0', k_0') \in \mathcal{T}$ **do**
  if $(U_0', m', k_0') = (U_1, m, k_1)$
    return false
return true

$\mathsf{IsSrc}(U_1, U_2, m, k_1, \mathcal{T})$

for $(U_1', U_2', m', k_1', U_0', k_0') \in \mathcal{T}$ **do**
  if $(U_1', U_2', m', k_1') = (U_1, U_2, m, k_1)$
    if $(U_0', k_0') \neq (\text{-}, \text{-})$
      return false
return true

Fig. 13: Security game for accountability. When sending a fresh message, the adversary access the $\mathcal{O}_{\mathsf{HonSend}}$ oracle as $\mathcal{O}_{\mathsf{HonSend}}(\cdot, \cdot, \cdot, \text{-}, \text{-})$, where (-,-) denotes empty elements. And $*$ is a wildcard symbol when it appears in the elements of $\mathcal{T}_q$. $\mathcal{L}_{tag}$ and $\mathcal{T}_q$ are maintained by the oracles that the adversary cannot observe. The identity keys in $\mathcal{L}^h_{ik}$ and $\mathcal{L}^m_{ik}$ were randomly generated. psk is a secret key generated and maintained by the challenger.

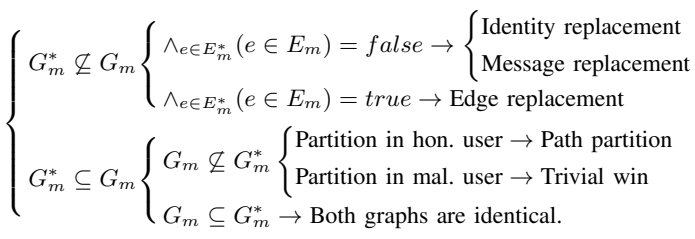


Fig. 14: Adversaries' winning cases of accountability

the adversary is to intentionally manipulate traceback so that the Trace algorithm's output G^*_m differs from the (actual) forwarding graph G_m. We can exhaust all possible winning cases of the adversary by listing all possible scenarios where the two graphs differ.

To prevent trivial wins, we do not consider the winning cases where both sender and recipient are malicious users, i.e., path partition in two malicious users. A malicious sender could easily evade tracing when the recipient omits the tag verification. Nevertheless, we allow two malicious users to communicate in the game; therefore, a malicious sender can add arbitrary message tag tag to $\mathcal{L}_{tag}$. Note that even colluding users cannot manipulate dtk in tag' since it is embedded by the challenger (i.e., platform and tag server in impact tracing). This prevents the colluding users from smearing an honest user. Formally, we define the accountability for FT as follows.

Definition 4 (Accountability). *The advantage of the adversary in breaking the game of accountability is defined as:*

$$\mathsf{Adv}^{act}_{FT}(\mathcal{A}) = \Pr\left[ACT^{\mathcal{A}}_{FT} = 1\right].$$

Theorem 3. *Let FT be the fuzzy message traceback scheme defined in Section III. For any PPT adversary $\mathcal{A}$, there exist PPT adversaries such that:*

$$|\mathsf{Adv}^{act}_{FT}(\mathcal{A}) - \psi| \leq \mathsf{negl}(\lambda),$$

where ψ is the FPR of random response in FT.

For a rigorous proof, please see the full version.