Secure Complaint-Enabled Source-Tracking for Encrypted Messaging

Charlotte Peale
Stanford University

Saba Eskandarian
UNC Chapel Hill

Dan Boneh
Stanford University

ABSTRACT

While the end-to-end encryption properties of popular messaging schemes such as Whatsapp, Messenger, and Signal guarantee privacy for users, these properties also make it very difficult for messaging platforms to enforce any sort of content moderation. This can lead to the unchecked spread of malicious content such as misinformation on such platforms. In 2019, Tyagi et al. developed *message traceback*, which addresses this issue by allowing a messaging platform to recover the path of a forwarded message after a user reports it for malicious content. This paper presents an alternative to message traceback that offers more privacy to users and requires less platform-side storage. We term this approach *source-tracking* for encrypted messaging schemes. Source-tracking enables messaging platforms to provide the privacy guarantees expected from standard end-to-end encryption, but also helps hold the sources of malicious messages accountable: if malicious content is reported by a user, the source can be identified. We formalize security goals for source-tracking schemes and design and implement two source-tracking schemes with different security and performance tradeoffs.

1 INTRODUCTION

End-to-end encrypted messaging apps like WhatsApp and Signal provide users with strong privacy guarantees and deliver billions of messages a day [29]. Concerns about abuse and misinformation on such platforms have led to work on technical solutions for verifiable reporting of messages with no impact on the privacy guarantees for messages that are not reported [12, 14, 32, 33]. Techniques for enabling some degree of moderation with minimal impact on privacy are all the more important as Brazil and India consider laws that would require messaging providers to be able to identify the sources of misinformation [26–28, 30]. In the absence of strong privacy-preserving tools to minimize the impact of such policies, these requirements would cripple the hard-won privacy victories of end-to-end encrypted messaging.

The majority of work on verifiably reporting abusive messages has been in the context of *message franking* [12, 14, 32], which focuses on reporting the immediate sender of a message and not on identifying the sources of viral misinformation campaigns. While message franking techniques are important for reporting abusive messages, they do not help find the origin of a message that may be forwarded many times before someone reports it. Although it would be possible to find the source of a message by combining a message franking scheme with extensive metadata collection and retention, e.g., a graph of all message sources and destinations with forwarded messages labeled, this approach requires a great deal of storage and forces the platform to collect and keep lots of metadata, which may compromise user privacy in its own right. Solutions relying on having users sign their messages are also unsatisfactory, as this damages deniability and may reveal the identity of the original sender of a message even before it is reported.

To our knowledge, the only work to consider tracing the source of a reported message is the *traceback* scheme of Tyagi et al. [33]. Traceback boasts extremely low client-side costs and reveals the set of users who have received a reported message without revealing any additional information about a message's source before it is reported. Unfortunately, revealing all the recipients of a message may be leaking too much if the goal of reporting is only to find the source of the message. Moreover, tracing a message back to its source takes time linear in the length of the forwarding chain and requires a 32 Byte token to be stored by the server for each message sent, regardless of whether that message is ever reported. The token must be kept for as long as the message can be traced. For a high-volume messaging platform like WhatsApp, this translates into terabytes of additional long-term storage each day. Thus traceback removes the need for a platform to collect message metadata, but it does not reduce the large quantity of data the platform must store.

This paper introduces the notion of *source-tracking* for end-to-end encrypted messaging schemes. Like traceback schemes, source-tracking schemes reveal the source of a reported message, but they do not reveal the identities of users who received a message between when it was first sent and when it was reported. Moreover, our schemes do not require the server to store any data for each message sent, and the time to trace the source of a reported message does not depend on the length of the forwarding chain.

We begin by formalizing the notion of a source-tracking scheme and providing security definitions that capture and exceed the privacy requirements of prior message franking and traceback schemes. We consider confidentiality of an unreported message's origin from both the platform and other users who may be forwarded the message. In particular, we offer two notions of confidentiality, *tree-linkable* and *tree-unlinkable* confidentiality. In a tree-linkable scheme, the recipient of two messages with identical plaintexts can tell if the two messages have the same origin. A tree-unlinkable scheme hides even this information.

Our definitions also require accountability, unforgeability, and deniability. These properties require that every successfully delivered message must be able to be traced to its source, that nobody can be framed for originating a message, and that only the platform can verify the origin of a reported message.

Next, we provide two constructions of secure source-tracking schemes, one with tree-linkable security and one with tree-unlinkable security. We implement and evaluate our schemes, comparing performance both to the traceback scheme of Tyagi et al. [33] and an implementation of end-to-end encryption based on the double ratchet protocol [25, 39] without any additional functionality for finding the source of a reported message.

In exchange for its slightly weaker security guarantee, the tree-linkable scheme offers exceptional performance, introducing very reasonable overheads compared to an end-to-end encrypted messaging service with no source-tracking capability. For example, it takes under $60\mu s$ to find the source of a reported message. The additional server-side overhead of using our tree-linkable scheme is only $20\mu s$ of cryptographic operations per message delivered. The scheme only relies on standard cryptographic primitives such as signatures and commitments, re-using the security of the underlying end-to-end encrypted messaging service to provide additional security properties at low cost.

Our tree-unlinkable construction, while slower than our first construction, requires only milliseconds to send and receive messages. At a high level, we unlinkably hide the source of a message using a heavily modified version of the algebraic MAC and anonymous credential techniques used in the new Signal group messaging system [4, 5].

Our implementation and raw evaluation data are free and open source and can be accessed at https://github.com/cpeale/srctracking.

In summary, this paper makes the following contributions:

- Introduces the notion of *source-tracking* for end-to-end encrypted messaging systems and defines strong security requirements for source-tracking schemes.
- Constructs a *tree-linkable* source-tracking scheme with exceptional performance that can easily be deployed on top of existing end-to-end encrypted messaging services.
- Constructs a *tree-unlinkable* source-tracking scheme which requires milliseconds to process and deliver messages.
- Implements and evaluates both schemes, including comparisons to both prior work and a baseline messaging implementation without source-tracking functionality.

2 DESIGN GOALS

A source-tracking scheme builds on top of an end-to-end encrypted messaging system and allows a user who receives an abusive message to report that message to the messaging platform. This is achieved by augmenting the usual protocol for sending and receiving messages as well as by adding a new Report feature. Given the contents of a report, the messaging platform should be able to uncover the true author of the message, regardless of how many times the message has been forwarded. Source-tracking schemes do not identify objectionable content on their own (as in, e.g., [16]), they only deal with the process of identifying the source of content that a user has reported.

Security goals. Informally, we require the following security properties from a secure source-tracking scheme. We formalize these properties in Section 3.

- **Confidentiality**: Neither the platform nor any user not involved in sending or receiving a given message should learn anything about that message's forwarding history or contents. The scheme should also preserve unlinkability: a user who receives two different forwarded messages should not be able to tell if they were authored by the same user. After a report, the messaging platform should not learn anything new about the reported message's forwarding history other than the identity of the source.
- **Accountability**: No malicious user should be able to send a message which cannot later be traced back to them.
- **Unforgeability**: No malicious user should be able to frame another user for sending a message it did not send.
- **Deniability**: Only the platform can verify a report, so if the contents of a report are leaked, it would be impossible for a third party to verify that the report implicated a particular user.

Tree-linkable and unlinkable confidentiality. We consider two levels of confidentiality: *tree-linkable* and *tree-unlinkable*. When a message is authored and then forwarded, a *forwarding tree* is created consisting of all the users who have received and forwarded that message, rooted at the author of the message. In the strictest form of user confidentiality, a recipient shouldn't be able to tell which forwarding tree a message came from. If a user receives a forward consisting of plaintext m, and then later receives another forward of the same plaintext m, the user shouldn't be able to tell whether the two messages are from the same forwarding tree or from different trees whose messages happen to have the same plaintext. If a scheme satisfies this form of confidentiality, we will say it has *tree-unlinkable* confidentiality.

In some situations, this notion of confidentiality may be unnecessarily strong, as whether or not two identical messages are on the same forwarding tree may not be sensitive information. Additionally, depending on the exact implementation of a messaging scheme, other metadata associated with the message (such as a timestamp) would mean that no two forwarding trees could ever correspond to the same message plaintext. If a scheme is such that a recipient can determine whether two forwards came from the same tree, but can still learn nothing about the tree structure or other messaging activity, we say that it satisfies *tree-linkable* user confidentiality.

It is worth noting, however, that there are circumstances where a stronger tree-unlinkable security guarantee may be preferable. For example, a malicious insider working for a messaging platform can send the same message to multiple recipients. Later, if one of those recipients sends the message to someone who reports it, the sender will know, from tree-linkability, which person's version of the message was reported. Although we believe tree-linkability suffices for most scenarios, we design and evaluate constructions of both tree-linkable and tree-unlinkable schemes.

Accommodating report metadata. In order to properly handle a report, a messaging platform may wish to receive additional information about a reported message beyond the identity of its original author, e.g., an exact or approximate timestamp of when the message was written. Our schemes support this functionality by allowing messaging platforms to optionally add report metadata to messages such that the metadata associated with a message when it is originally authored will be revealed if and only if that message is reported. We note that this feature is intended for including timestamps or other associated data, and, like any cryptosystem, it is possible for it to be abused in ways that damage security. For example, the platform could damage deniability by including a unique id in each message's metadata and keeping a public database of hashes of ids used in authentic messages.

Limitations. Any system aimed at finding the source of a forwarded message faces certain non-technical limitations. Both our scheme and prior work on traceback [33] track the source of messages that are forwarded using a messaging app's "forward" feature. However, a user could also re-type a message instead of forwarding it using the app, at which point the message appears to be a fresh message which happens to have the same content. Thus the actual guarantee of a scheme that finds the source of a message is that it finds the first person to send the message that was later forwarded, not its real-world author.

As discussed in [15], a more significant issue raised by schemes that find the source of a message is that they allow a user to reveal information about someone other than their immediate conversation partner. This is a fundamental component of the functionality, but it should be weighed carefully whether the potential moderation benefit of such a feature outweighs the privacy risk. Part of our goal in designing source-tracking is to minimize the risk posed compared to traceback. For example, consider the case where a journalist reports a piece of misinformation to a platform. A traceback scheme would, in the process of revealing the source of the misinformation, also reveal the identity of the user who forwarded it to the journalist. Source-tracking does not reveal the path taken by a message, so only the original sender of the message would be revealed.

Finally, both our work and traceback rely on the platform being aware of the identity of the parties sending and receiving a message at the time the message is sent. This information is currently required by most end-to-end encrypted messaging systems in order to deliver messages, so our schemes do not impose additional metadata-collection or retention requirements on these systems. However, it remains an important problem for future work to develop source-tracking schemes that do not even require this minimal metadata at the time of message delivery.

3 FORMALIZING SOURCE-TRACKING

We now formalize the syntax and security definitions for a source-tracking scheme. We begin by describing some notation we will use throughout the rest of the paper.

Let $x \leftarrow F(y)$ denote the assignment of the output of $F(y)$ to x, and let $x \xleftarrow{R} S$ denote assignment to x of an element sampled uniformly random from set S. We use $\mathcal{A}^H$ to denote that $\mathcal{A}$ has oracle access to some function H. A function $\mathsf{negl}(x)$ is *negligible* if for all $c > 0$, there is a x_0 such that for all $x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We omit x if the parameter is implicit. Throughout the paper we also omit an implicit security parameter λ. Finally, we use $\perp$ to indicate an empty message or special character indicating failure.

Interactive protocols. We define an interaction between two parties using the notation

$$(out_1, out_2) \leftarrow \langle P_1(\text{secret params}), P_2(\text{secret params})\rangle(\text{public params})$$

Where the first party acts according to the protocol defined by P_1 and has access to the secret parameters of P_1 as well as all public parameters, and the second party acts according to P_2 with access to the public parameters as well as its analogous secrets.

Tables and sets. Our security definitions use tables to keep track of important information about adversary queries. Tables are denoted with a capital T and a subscript name, and store key/value pairs. To add a key/value pair to a table, we use the notation $T[key] \leftarrow value$. To retrieve a value corresponding to a particular key from a table, we use the notation $value \rightarrow T[key]$. We use standard set notation to check if a key is included in a table ($key \in T$). Sets are similar to tables, but only store a set of values. We add values $v_1, ..., v_k$ to a set $\mathcal{S}$ with $\mathcal{S}.add(v_1, ..., v_k)$.

3.1 Source-Tracking Syntax and Correctness

A source-tracking scheme ST consists of six algorithms: KGen, $\mathsf{NewUser}$, $\mathsf{AuthMsg}$, FwdMsg, RecMsg, and Report, where the latter five are interactive protocols between a user and the messaging platform. Source-tracking involves users $U_1, ..., U_n$, each represented by a unique identifier, e.g., a username or long-term public key taken from some set $\mathcal{U}$, and a platform P.

In contrast with prior work, which operates independently of the underlying end-to-end encrypted messaging scheme [33], we allow source-tracking schemes to make use of the underlying messaging scheme as a black box. We see this as striking a happy medium where source-tracking can take advantage of the security benefits already provided by the messaging scheme while still being easy to deploy on top of existing applications. Although modern end-to-end encrypted messaging protocols provide many useful security properties [1, 7, 25], our constructions will only rely on them to provide authenticated encryption and protect against replay attacks. To abstract away the details of the underlying messaging scheme, we will assume the existence of a messaging oracle

$$\mathcal{E} = (\mathsf{send}(m, U_s, U_r), \mathsf{receive}(ct, U_s, U_r))$$

that sends and authenticates messages between users using the underlying end-to-end encryption scheme the source-tracking scheme is associated with.

The syntax of a source-tracking scheme is defined as follows:

- $\mathsf{KGen}(pp) \rightarrow (\mathsf{pk}, \mathsf{sk})$: The platform runs this algorithm at system setup. It takes in public parameters pp and outputs a platform key pair (pk,sk).
- $\mathsf{NewUser}$ is an interactive protocol between a new user and the messaging platform to register that user in the system. It is represented by the pair of algorithms U_{new}, P_{new}:

  $$(ad, \mathcal{U}') \leftarrow \langle U_{new}, P_{new}(\mathsf{sk}, \mathcal{U})\rangle(U_n, \mathsf{pk})$$

  Where $\mathcal{U}$ is the set of currently registered users, $(\mathsf{pk}, \mathsf{sk})$ is the platform's key pair, and U_n is the user information or id associated with the new user. We assume that the user runs a separate protocol to register with the underlying messaging scheme as needed. Optionally, U_{new} can add additional private inputs, e.g., a secret used to prove its identity.

  On success, the protocol should return some authoring data ad to the user that can be used to send messages and the updated membership set $\mathcal{U}'$ to the platform.

  While we define this function interactively in our security definitions for maximum flexibility, a non-interactive function run by the platform suffices for both of our schemes. For ease of notation in the first scheme, which uses no authoring data, we will denote the non-interactive version of this protocol as the function $newUser(U_i, \mathsf{sk})$, run by the platform, which returns only the new set of users $\mathcal{U}$ or $\perp$ on failure.
- $\mathsf{AuthMsg}$ is an interactive protocol between a user authoring a message and the platform, represented by the pair of interactive

algorithms

$$((ad', e), (pd, e)) \leftarrow \langle U_{auth}(msg), P_{send}(\mathsf{sk}, md)\rangle(U_s, U_r, \mathsf{pk})$$

where msg is a tuple (m, ad), m is the plaintext being sent, and ad ("authoring data") is any associated data required to send a message.
The platform has secret inputs sk, its secret key, and report metadata md that it wants to recover if the message is later reported. Both algorithms receive the identities of the sender and receiver as well as the platform public key. Upon successful completion of the protocol, the sender gets updated authoring data ad' that it can store for authoring future messages, and the platform receives platform data pd to be used when the message is delivered to its recipient. Both parties receive the message identifier e for the message sent using the underlying messaging platform $\mathcal{E}$.

- FwdMsg is an interactive protocol between a user forwarding a message and the platform, represented by interactive algorithms

$$((fd', e), (pd, e)) \leftarrow \langle U_{fwd}(msg), P_{send}(\mathsf{sk}, md)\rangle(U_s, U_r, \mathsf{pk})$$

where each algorithm's inputs and outputs are identical to AuthMsg except that $msg = (m, fd)$, where fd ("forwarding data") represents associated data required to forward a message, and the user's output is updated forwarding data fd' to be used if the user wishes to forward the same message again.
We note that because the platform shouldn't be able to distinguish between an authored and a forwarded message, the platform protocol P_{send} is identical to the platform's protocol for an authored message. The report metadata (md) is only included if the message is new, so in this case it is passed to the forwarding function but remains unused.

- RecMsg is an interactive protocol between a user receiving a message and the platform, represented by the pair of interactive algorithms

$$((m, fd), \perp) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd)\rangle(U_s, U_r, e, \mathsf{pk})$$

The platform has access to its secret key sk and the transaction data pd generated when the message was sent (from either AuthMsg or FwdMsg), and both user and platform have access to the sending and receiving users' identities, the message identifier for the message sent by the underlying message scheme, and the platform's public key. Upon successful completion, the receiving user gets the message plaintext m and forwarding data fd that it can use to forward or report the message in the future.

- Report: This is an interactive protocol between a user who would like to report a received message and the platform, represented by the pair of interactive algorithms

$$(fd', (U, md)) \leftarrow \langle U_{rep}(fd), P_{rep}(\mathsf{sk})\rangle(m, \mathsf{pk})$$

where the user knows the forwarding data fd for the message it would like to report, the platform knows the platform secret key, and both participants have access to the platform public key and the plaintext m of the message being reported. Upon successful completion, the platform gets the source user identity U and associated metadata md for the reported message. Optionally, the reporting user could get new forwarding data fd' used to forward the message again in the future, but we do not require use of this in either of our schemes.

We define correctness for a source-tracking scheme as follows.

Definition 3.1. A source-tracking scheme ST is *correct* if when all users and the platform are honest (i.e. follow the protocols and don't try to forward or report messages that they have not received), all protocols will fail with zero probability, messages will be delivered as intended, and if a user reports a message to the platform, the platform can recover the identity of the original author of the message as well as the metadata that the platform included at the time of authoring with probability one.

3.2 Confidentiality

Confidentiality guarantees that a source-tracking scheme does not break the privacy of the underlying end-to-end encryption system or leak more forwarding metadata than the underlying messaging system to either the platform or to users. We formalize this by defining *user confidentiality*, which ensures that a malicious user cannot learn more about other users' messaging activity than it could without the source-tracking scheme, and *platform confidentiality*, which makes the same guarantee for the platform before a message is reported and additionally requires that, even after a report, the platform learns only the source and associated metadata of a reported message and nothing more.

We present definitions for both tree-linkable and tree-unlinkable user confidentiality. For the platform, we only define tree-linkable platform confidentiality because the platform can set the report metadata to be unique for each forwarding tree, allowing it to distinguish between two forwarding trees with the same plaintext when messages are reported. Note that tree-linkability/unlinkability for the platform is only relevant when a report is made: the security guarantee for unreported messages is identical in the two settings. For simplicity, we present our definitions without allowing for adaptive corruption of users.

User Confidentiality. We define a security game for user confidentiality in Figure 1 with respect to a source tracking scheme ST, an adversary $\mathcal{A}$, and an underlying messaging scheme $\mathcal{E}$.

Intuitively, the user confidentiality game allows an adversary to assume the identity of a malicious user and simulate messaging activity with access to the view of that user. An adversary is given the ability to forward an arbitrary number of message pairs to adversary-controlled users. Only one message from each pair is actually forwarded, and the adversary wins if it can determine which of the two messages was sent.

To prevent trivial wins, the game requires that within a pair, messages have the same plaintext and have been sent to the adversary by the same honest user, although the forwarding history of the two messages, including their original source, can differ. The tree-linkable version of the game adds the additional restriction that the forwarding trees of the queried messages must either be the same tree, always be previously queried together, or have never before been sent to the malicious user. While the adversary cannot make calls to the challenge with two different message plaintexts, the game still captures basic unlinkability because an adversary can make two or more calls to the challenge, allowing it to receive multiple different messages along paths of its choice.

This game is constructed by giving the adversary access to a number of oracles it can use to simulate activity in a messaging system. The getUser function allows the adversary to create an

```
$\mathsf{getUser}(U, isMal)$
$\mathcal{U} \leftarrow \mathcal{U}_{honest} \cup \mathcal{U}_{mal}$
if $U \in \mathcal{U}$ : return $\bot$
if $isMal$ :    $\mathcal{U}' \leftarrow \langle \mathcal{A}, P_{new}(\mathsf{sk}, \mathcal{U}) \rangle (U, \mathsf{pk})$
    if $\mathcal{U}' = \bot$ : return $\bot$
    return $\mathcal{U}_{mal}.add(U)$
$(ad, \mathcal{U}') \leftarrow \langle U_{new}, P_{new}(\mathsf{sk}, \mathcal{U}) \rangle (U, \mathsf{pk})$
if $(ad, \mathcal{U}') = \bot$ : return $\bot$
$T_{auth}[U] \leftarrow ad$
return $\mathcal{U}_{honest}.add(U)$
```

```
$\mathsf{goodAuth}(U_s, U_r, m, md)$
if $U_s \notin \mathcal{U}_{honest} \vee U_r \notin \mathcal{U}_{honest}$ :    return $\bot$
$ad \leftarrow T_{auth}[U_s], msg \leftarrow (m, ad)$
$(ad', pd, e) \leftarrow \langle U_{auth}(msg), P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
$(m, fd) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
if $(m, fd) = \bot$ :    return $\bot$
$T_{auth}[U_s] \leftarrow ad'$
$mid \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid] \leftarrow (U_s, U_r, m, fd, mid)$
return $mid$
```

```
$\mathsf{goodFwd}(U_s, U_r, mid, md)$
if $U_s \notin \mathcal{U}_{honest} \vee U_r \notin \mathcal{U}_{honest} \vee mid \notin T_{rec}$ :    return $\bot$
$(U_s', U_r', m, fd, tid) \leftarrow T_{rec}[mid]$
if $U_r' \neq U_s$ :    return $\bot$
$msg \leftarrow (m, fd)$
$(fd_s, pd, e) \leftarrow \langle U_{fwd}(msg), P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
$(m, fd_r) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
if $fd_r \vee fd_s = \bot$ :    return $\bot$
$mid' \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid] \leftarrow (U_s', U_r', m, fd_s, tid)$
$T_{rec}[mid'] \leftarrow (U_s, U_r, m, fd_r, tid)$
return $mid'$
```

```
$UCONF^{\mathcal{A},b,\ell}_{ST,\mathcal{E}}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(params)$
$b' \leftarrow \mathcal{A}^{O_{\ell,b}}(\mathsf{pk})$
return $b'$
```

```
$\mathsf{malSend}(U_s, U_r, md)$
if $U_s \notin \mathcal{U}_{mal}$ : return $\bot$
$(pd, e) \leftarrow \langle \mathcal{A}, P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
if $U_r \in \mathcal{U}_{mal}$ :
    return $\langle \mathcal{A}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$(m, fd) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
if $fd = \bot$ :    return $\bot$
$mid \xleftarrow{R} \{0,1\}^n$
$\mathcal{S}_{seen}.add(mid), \mathcal{S}_{allowed}.add((mid, mid))$
$T_{rec}[mid] \leftarrow (U_s, U_r, m, fd, mid)$
return $mid$
```

```
$\mathsf{malRec}_{linkable,b}(U_r, mid_0, mid_1, md)$
if $U_r \notin \mathcal{U}_{mal}$ : return $\bot$
if $mid_0 \vee mid_1 \notin T_{rec}$ :    return $\bot$
$(U_s^{(0)}, U_r^{(0)}, m_0, fd_0, tid_0) \leftarrow T_{rec}[mid_0]$
$(U_s^{(1)}, U_r^{(1)}, m_1, fd_1, tid_1) \leftarrow T_{rec}[mid_1]$
if $U_r^{(0)} \neq U_r^{(1)} \vee U_r^{(0)} \in \mathcal{U}_{mal} \vee m_0 \neq m_1$ :    return $\bot$
if $linkable \wedge (tid_0, tid_1) \notin \mathcal{S}_{allowed} \wedge \{tid_0, tid_1\} \cap \mathcal{S}_{seen} \neq \emptyset$ :
    return $\bot$
$\mathcal{S}_{allowed}.add((tid_0, tid_1)), \mathcal{S}_{seen}.add(tid_0, tid_1)$
$msg \leftarrow (m_b, fd_b)$
$(fd_s, pd, e) \leftarrow \langle U_{fwd}(msg), P_{send}(\mathsf{sk}, md) \rangle (U_r^{(b)}, U_r, \mathsf{pk})$
$T_{rec}[mid_b] \leftarrow (U_s^{(b)}, U_r^{(b)}, m_b, fd_s, tid_b)$
return $\langle \mathcal{A}, P_{rec}(\mathsf{sk}, pd) \rangle (U_r^{(b)}, U_r, e, \mathsf{pk})$
```

Figure 1: User confidentiality game and oracles. The oracle $O_{\ell,b}$ gives the adversary access to $\mathsf{getUser}(\cdot,\cdot)$, $\mathsf{goodAuth}(\cdot,\cdot,\cdot,\cdot)$, $\mathsf{goodFwd}(\cdot,\cdot,\cdot,\cdot)$, $\mathsf{malSend}(\cdot,\cdot,\cdot)$, and $\mathsf{malRec}_{\ell,b}(\cdot,\cdot,\cdot,\cdot)$ as well as the $\mathsf{send}(\cdot,U,\cdot)$ and $\mathsf{receive}(\cdot,\cdot,U)$ oracles for the underlying encrypted messaging scheme for all $U \in \mathcal{U}_{mal}$. ℓ is a fixed variable that determines whether the $UCONF$ game requires tree-linkability ($\ell = 1$) or tree-unlinkability ($\ell = 0$), and b is a hidden variable that determines which version of the game the adversary sees.

arbitrary number of malicious (adversary-controlled) users, which are stored in the set $\mathcal{U}_{mal}$, as well as honest users that it does not control, which are stored in $\mathcal{U}_{honest}$. goodAuth and goodFwd allow the adversary to send messages between users it does not control, while malSend and malRec allow an adversary to send messages between an honest user and a malicious user. The adversary is also given oracles $\mathsf{send}(\cdot, U, \cdot)$ and $\mathsf{receive}(\cdot,\cdot,U)$ to send and receive messages from the adversary-controlled users $U \in \mathcal{U}_{mal}$ in the underlying encrypted messaging scheme. In all cases, the adversary gets to choose the plaintext m that is sent as well as the metadata md included with authored messages. The same game applies for both tree-linkable and tree-unlinkable security, with the only change being that the *linkable* flag in the malRec oracle is set to true for tree-linkable security.

In order to keep track of messages and their associated forwarding trees, each message sent in the game is identified by a unique message id *mid*, and each forwarding tree is identified by a unique *tid*, which is the value of *mid* for the message at the root of the tree. The game uses table T_{auth} to keep track of authoring data *ad* for each honest user and table T_{rec} to keep track of messages received by honest users. The set $\mathcal{S}_{seen}$ records which forwarding trees the adversary has encountered, and the set $\mathcal{S}_{allowed}$ keeps track of the pairs of trees from which malRec can be given messages without enabling a trivial win in the tree-linkable variant of the game.

Definition 3.2. The *user confidentiality* advantage of an adversary $\mathcal{A}$ against a source-tracking scheme ST and messaging scheme $\mathcal{E}$ is defined as

$$\mathsf{Adv}^{\mathrm{uconf}}_{ST,\mathcal{E}}(\mathcal{A}) = \left|\Pr\left[UCONF^{\mathcal{A},1,\ell}_{ST,\mathcal{E}} = 1\right] - \Pr\left[UCONF^{\mathcal{A},0,\ell}_{ST,\mathcal{E}} = 1\right]\right|$$

where the adversary is given access to send and receive oracles for all malicious users in $\mathcal{U}_{mal}$ as well as oracles getUser, goodAuth, goodFwd, malSend, and $\mathsf{malRec}_{\ell,b}(\cdot,\cdot,\cdot,\cdot)$. We refer to the advantage as the *tree-linkable* advantage if ℓ = true and as the *tree-unlinkable* advantage otherwise.

Platform confidentiality. We assume that a platform for an encrypted messaging scheme can see that a user U_s sent a message to U_r, but that it cannot learn anything about the content of that message, or whether it was a forward. There has been research into the possibility of metadata-hiding messaging schemes where the sender or receiver of a message can be hidden from the platform (e.g., [8, 17–19, 37]), but we will be aligning our confidentiality goals with the former setting, which corresponds to most deployed messaging schemes, and assuming that the identities of the sending and receiving users are visible to the platform.

The platform confidentiality game operates similarly to the user confidentiality game, except that now the adversary controls the platform and an arbitrary number of malicious users. Similar to user confidentiality, the game gives the adversary oracles goodSend and goodRec for sending messages between honest users and malSend and malRec to send a message from or to a malicious user, respectively. We separate the sending and receiving of messages into separate oracles to allow the adversary to launch attacks that involve delivering message data to unintended recipients or otherwise tampering with the message delivery process. Additionally, a new report oracle allows a user to report a message to the platform. Finally, since the security game must ensure that the platform adversary learns nothing about the plaintext of messages, the challenge message in the platform confidentiality game is sent to an honest user instead of a malicious user. We formalize platform confidentiality with the *PCONF* game, described in Appendix A.1.

Definition 3.3. The advantage of an adversary $\mathcal{A}$ in the *PCONF* game against a source-tracking scheme ST and messaging scheme $\mathcal{E}$ is defined as

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

where the adversary is given access to the oracles in Figure 7.

Definition 3.4. We say that a source-tracking scheme ST and associated messaging scheme $\mathcal{E}$ satisfy *confidentiality* if for all efficient adversaries $\mathcal{A}_u$ and $\mathcal{A}_p$, we have that both $\mathsf{Adv}^{\mathrm{uconf}}_{ST,\mathcal{E}}(\mathcal{A})$ and $\mathsf{Adv}^{\mathrm{pconf}}_{ST,\mathcal{E}}(\mathcal{A}_p)$ are negligible. If the advantage used in the *UCONF* game is tree-linkable, then we say the scheme is tree-linkably confidential, and likewise tree-unlinkably confidential if the advantage is tree-unlinkable.

3.3 Accountability and Unforgeability

Accountability and unforgeability ensure that a source-tracking scheme can provide the platform with useful and accurate information when a message is reported. We present security games for both properties in Figure 2.

The *srcBIND* game addresses accountability by allowing a user to send and receive any messages it would like, and then challenging the adversary to send a message that is received by an honest user, but fails when the honest user tries to report it to the platform. The *unFORGE* game addresses unforgeability by challenging an adversary who can see the result of receiving and sending messages to malicious users to create a report that is validated by the platform but implicates an honest user in sending a message it never sent.

Both games give the adversary a set of oracles which allow it to send and receive messages between arbitrary users and control the sending and receiving on a set of users that it controls. The oracle getUser allows an adversary to add honest and malicious users. Oracles goodAuth and goodFwd allow sending messages from honest users, and malSend allows malicious users to send messages to honest users. The adversary is also given access to a reporting oracle $\langle\cdot, P_{rep}(\mathsf{sk})\rangle(\cdot, \mathsf{pk})$ and oracles for sending messages between malicious users in the underlying messaging scheme.

We assume the oracle has an associated $\mathcal{M}_{sent}$ set that keeps track of the messages that have been sent by honest users. As in the confidentiality games (Section 3.2), we use unique message ids (*mid*) to refer to message transactions between honest users, use a table T_{auth} to keep track of the authoring data for honest users, and use a table T_{rec} indexed by *mid* to keep track of the data associated with messages received by honest users.

Definition 3.5. We define the advantage of an adversary $\mathcal{A}$ in the *srcBIND* game for a source-tracking scheme ST and messaging scheme $\mathcal{E}$ as

$$\mathsf{Adv}^{\mathrm{src-bind}}_{ST,\mathcal{E}}(\mathcal{A}) = \Pr[srcBIND^{\mathcal{A}}_{ST,\mathcal{E}} = 1].$$

We say that ST is *accountable* or has accountability if $\mathsf{Adv}^{\mathrm{src-bind}}_{ST,\mathcal{E}}(\mathcal{A})$ is negligible for all efficient adversaries $\mathcal{A}$.

Definition 3.6. We define the advantage of an adversary $\mathcal{A}$ in the *unFORGE* game for a source-tracking scheme ST and messaging scheme $\mathcal{E}$ as

$$\mathsf{Adv}^{\mathrm{unforge}}_{ST,\mathcal{E}}(\mathcal{A}) = \Pr[unFORGE^{\mathcal{A}}_{ST,\mathcal{E}} = 1].$$

We say that ST is *unforgeable* if $\mathsf{Adv}^{\mathrm{unforge}}_{ST,\mathcal{E}}(\mathcal{A})$ is negligible for all efficient adversaries $\mathcal{A}$.

We note that our unforgeability and accountability requirements do not apply to the platform, only to users. As observed by [32], requiring that a scheme be secure against platform-produced forgeries would conflict with the deniability properties discussed below.

3.4 Deniability

We require that a source-tracking scheme satisfies two types of deniability to protect users if the messaging system is compromised:

- *universal deniability*: Users should be able to deny participating in a forwarding path of a reported message.
- *platform compromise deniability*: If the platform's secret key sk is leaked publicly, a user should still be able to deny participating in a reported forwarding path.

Motivated by the approach of [32], we capture these goals by requiring the existence of efficient protocols UForge and PForge that can be executed by any user (with access to the platform's secret

```
$srcBIND^{\mathcal{A}}_{ST,\mathcal{E}}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(params)$
$(U_s, U_r, md, st) \leftarrow \mathcal{A}_1^{O}(\mathsf{pk})$
if $U_r \notin \mathcal{U}_{honest}$ :   return 0
$(pd, e) \leftarrow \langle \mathcal{A}_2(st), P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
$(m, fd) \leftarrow \langle U_{rec}, P_{rec}(pd, \mathsf{sk}) \rangle (U_s, U_r, e, \mathsf{pk})$
if $fd = \perp$ :   return 0
$(src, md) \leftarrow \langle U_{rep}(fd), P_{rep}(\mathsf{sk}) \rangle (m, \mathsf{pk})$
if $(src, md) = \perp$ :   return 1
else : return 0
```

```
$unFORGE^{\mathcal{A}}_{ST,\mathcal{E}}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(params)$
$(m, st, \mathcal{M}_{sent}) \leftarrow \mathcal{A}_1^{O}(\mathsf{pk})$
$(src, md) \leftarrow \langle \mathcal{A}_2(st), P_{rep}(\mathsf{sk}) \rangle (m, \mathsf{pk})$
if $(src, md) = \perp$ :   return 0
if $src \notin \mathcal{U}_{mal} \wedge (src, md, m) \notin \mathcal{M}_{sent}$ :
    return 1
else : return 0
```

```
$\mathsf{goodAuth}(m, md, U_s, U_r)$
if $U_s \notin \mathcal{U}_{honest}$ :   return $\perp$
$ad \leftarrow T_{auth}[U_s]$
$msg \leftarrow (m, ad)$
$(ad', pd, e) \leftarrow \langle U_{auth}(msg), P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
$T_{auth}[U_s] \leftarrow ad', \mathcal{M}_{sent}.add((U_s, md, m))$
if $U_r \in \mathcal{U}_{mal}$ :
    return $\langle \mathcal{A}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$(m, fd) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$mid \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid] \leftarrow (U_s, U_r, m, fd)$
return $mid$
```

```
$\mathsf{goodFwd}(mid, md, U_r)$
if $mid \notin T_{rec}$ :   return $\perp$
$(U_s', U_s, m, fd) \leftarrow T_{rec}[mid]$
$msg \leftarrow (m, fd)$
$(fd', pd, e) \leftarrow \langle U_{fwd}(msg), P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
$T_{rec}[mid] \leftarrow (U_s', U_s, m, fd')$
if $U_r \in \mathcal{U}_{mal}$ :
    return $\langle \mathcal{A}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$(m, fd'') \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$mid' \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid'] \leftarrow (U_s, U_r, m, fd'')$
return $mid'$
```

```
$\mathsf{getUser}(U, isMal)$
$\mathcal{U} \leftarrow \mathcal{U}_{honest} \cup \mathcal{U}_{mal}$
if $U \in \mathcal{U}$ :   return $\perp$
if $isMal$ :
    $\mathcal{U}' \leftarrow \langle \mathcal{A}, P_{new}(\mathsf{sk}, \mathcal{U}) \rangle (U, \mathsf{pk})$
    if $\mathcal{U}' = \perp$ : return $\perp$
    return $\mathcal{U}_{mal}.add(U)$
$(ad, \mathcal{U}') \leftarrow \langle U_{new}, P_{new}(\mathsf{sk}, \mathcal{U}) \rangle (U, \mathsf{pk})$
if $(ad, \mathcal{U}') = \perp$ :   return $\perp$
$\mathcal{U}_{honest}.add(U)$
$T_{auth}[U] \leftarrow ad$
```

```
$\mathsf{malSend}(U_s, U_r, md)$
if $U_s \notin \mathcal{U}_{mal}$ :   return $\perp$
$(pd, e) \leftarrow \langle \mathcal{A}, P_{send}(\mathsf{sk}, md) \rangle (U_s, U_r, \mathsf{pk})$
if $U_r \in \mathcal{U}_{mal}$ :
    return $\langle \mathcal{A}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
$(m, fd) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_s, U_r, e, \mathsf{pk})$
if $fd = \perp$ :   return $\perp$
$mid \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid] \leftarrow (U_s, U_r, m, fd)$
return $mid$
```

Figure 2: Unforgeability and accountability games with their accompanying oracles. The adversary is also given access to $\mathsf{send}(\cdot, U, \cdot)$ **and** $\mathsf{receive}(\cdot, \cdot, U)$ **oracles for the underlying encrypted messaging scheme for all adversary-controlled users** $U \in \mathcal{U}_{mal}$**, as well a reporting oracle** $\langle \cdot, P_{rep}(\mathsf{sk}) \rangle (\cdot, \mathsf{pk})$**.**

keys in the case of PForge). These protocols must successfully create a set of forged transcripts and forwarding data indistinguishable to a third party from the actual transcripts resulting from the path and subsequent report of a forwarded message. In universal deniability, the party tasked with differentiating transcripts has access to the platform's public keys and the secret keys of all users in the system, while for platform compromise deniability, they are given access to the platform's secret keys as well. We discuss the space of potential deniability definitions and formalize security games for universal deniability (*UnivDEN*) and platform-compromise deniability (*PlatDEN*) in Appendix A.2.

Definition 3.7. For a deniability game *DEN* and forgery algorithm Forge, the advantage of an adversary $\mathcal{A}$ against this game, $\mathsf{Adv}^{\mathrm{den}}_{ST,\mathcal{E},\mathsf{Forge}}(\mathcal{A})$, is defined as

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

Definition 3.8. We say that a source-tracking scheme *ST* is *deniable* if there exist efficient (possibly interactive) algorithms UForge and PForge such that for any efficient adversaries $\mathcal{A}_u$ and $\mathcal{A}_p$, $\mathsf{Adv}^{\mathrm{univden}}_{ST,\mathcal{E},\mathsf{UForge}}(\mathcal{A}_u)$ and $\mathsf{Adv}^{\mathrm{platden}}_{ST,\mathcal{E},\mathsf{PForge}}(\mathcal{A}_p)$ are both negligible.

4 TREE-LINKABLE SOURCE TRACKING

Straw-man scheme. A first straw-man attempt at tree-linkable source-tracking could have the identity U_S of the original sender of a message included with the message plaintext, so the final message becomes an encryption of (U_s, m). While this approach would satisfy platform confidentiality and deniability requirements, it does not satisfy user confidentiality, accountability, or unforgeability, as nothing would prevent malicious users from observing or tampering with the sender identity included in the message.

Our scheme. Instead, the starting point for our approach is to have the platform append a signature on $(\mathsf{Enc}(k, U_s), c_m)$ to each message, where k is an encryption key known only to the platform and c_m is a commitment to the message plaintext, provided by the sender. This signature is passed along whenever a message is forwarded, and the signature is checked by each message recipient using the platform's public key and commitment randomness r included by the sender alongside the end-to-end encrypted plaintext. This scheme enforces user confidentiality, unforgeability, and accountability, but it does not yet provide platform confidentiality because the platform can always decrypt the sender identity that is sent alongside each message.

Our final scheme combines the construction thus far with the straw-man construction. When a message is forwarded for the first time, the forwarder includes the signature, commitment, and commitment randomness alongside the message plaintext that is end-to-end encrypted and hidden from the platform, i.e., it encrypts and sends $(m, \mathsf{Enc}(k, U_S), c_m, r)$. Subsequent forwards pass along the same information *inside* the end-to-end encrypted ciphertext, with each recipient checking that the signature and commitment match the provided message and encrypted sender pair. Fresh messages include padding to hide the fact that they do not contain forwarding information, and forwarded messages send the server a commitment to an empty message to hide the fact that they are forwards. Users report messages by sending the message plaintext, commitment, randomness, and signature to the platform, who checks the signature and decrypts the identity of the original sender.

In terms of security, hiding the forwarding information inside the end-to-end encrypted message provides platform confidentiality, while encrypting the original sender provides user confidentiality. The fact that the forwarding information a user sees does not change when a message is forwarded makes the scheme tree-linkable: a user can tell when the same original message has been forwarded to it twice. Accountability and unforgeability are provided by the platform being the one to include the sender identity and signing it, and deniability results from the sender identity being encrypted and therefore hidden from everyone except the platform (and being trivial to forge if the platform keys are known).

Our scheme, formalized in Figure 3, makes use of the underlying messaging scheme $\mathcal{E} = (\mathsf{send}(m, U_s, U_r), \mathsf{receive}(ct, U_s, U_r))$ as well as the following cryptographic tools.

- A symmetric encryption scheme

$$\mathcal{P} = (\mathsf{KGen}_{sym} \to \mathsf{k},\ \mathsf{Enc}(\mathsf{k}, m) \to ct,\ \mathsf{Dec}(\mathsf{k}, ct) \to m).$$

- A signature scheme

$$\mathcal{S} = (\mathsf{KGen}_{sig} \to (\mathsf{vk}_s, \mathsf{sk}_s), \mathsf{Sig}(\mathsf{sk}_s, m) \to \sigma,$$
$$\mathsf{Vf}(\mathsf{vk}_s, \sigma, m) \to \{0, 1\}.$$

- A commitment scheme

$$C = (\mathsf{Commit}(m) \to (c_m, r),\ \mathsf{Open}(c_m, m, r) \to \{0, 1\}).$$

One of the merits of the scheme in Figure 3 is that it is non-interactive. The communication for each protocol only consists of a single message sent from the user to the platform in the case of a send, or from the platform to the user in the case of a receipt, though optional status messages could be added to let a platform or user know if an operation succeeds.

We prove the following security theorems in Appendix B. Definitions of CPA- and AE-security can be found in [2].

Theorem 4.1. *Assuming that the platform encryption scheme $\mathcal{P}$ is CPA-secure, the commitment scheme C is hiding, and that the messaging encryption scheme $\mathcal{E}$ is AE-secure, then Scheme 1 is tree-linkably confidential (Definition 3.4).*

Theorem 4.2. *Assuming that the platform's signature scheme S is secure against any efficient adversary $\mathcal{A}$, then Scheme 1 satisfies accountability (Definition 3.5).*

Theorem 4.3. *Assuming that the signature scheme $\mathcal{S}$ is unforgeable and the commitment scheme C is binding against any efficient adversary $\mathcal{A}$, then Scheme 1 is unforgeable (Definition 3.6).*

```
KGen(params)
  k ← KGen_sym
  (sk_s, vk_s) ← KGen_sig
  pk ← vk_s, sk ← (k, sk_s)
  return (pk, sk)

newUser(U_i, sk)
  if U_i ∈ 𝒰 :
    return ⊥
  𝒰.add(U_i)
  return 𝒰

Report(m, pk)
  U_rep(fd)          P_rep(sk = (k, sk_s))
        fd
     ------>
                     (σ, src, c_m, r) ← fd
                     if ¬Open(c_m, m, r) :
                       return ⊥
                     if ¬Vf(pk, σ, (c_m, src)) :
                       return ⊥
                     return Dec(k, src)

AuthMsg(U_s, U_r, pk)
  U_auth(msg)                                P_send(sk = (k, sk_s), md)
  (m, ⊥) ← msg
  (c_m, r) ← Commit(m)
  //send (m,data) via underlying msg scheme
  e ← send((m, ⊥, c_m, r), U_s, U_r)
          (c_m, e)
     ------------------>
  return e                                   src ← Enc(k, (U_s, md))
                                             σ ← Sig(sk_s, (c_m, src))
                                             return (pd = (σ, src), e)

FwdMsg(U_s, U_r, pk)
  U_fwd(msg)                                 P_send(sk = (k, sk_s), md)
  (m, fd) ← msg
  (c_m, r) ← Commit(⊥)
  e ← send((m, fd, c_m, r), U_s, U_r)
          (c_m, e)
     ------------------>
                                             src ← Enc(k, (U_s, md))
                                             σ ← Sig(sk_s, (c_m, src))
                                             return (pd = (σ, src), e)

RecMsg(U_s, U_r, e, pk)
  U_rec                                              P_rec(sk, pd)
                                                          pd
                                                     <---------
  (σ, src) ← pd
  (m, fd′, c_m, r) ← receive(e, U_s, U_r)
  //verify platform signature
  if ¬Vf(pk, σ, (c_m, src)) :   return ⊥
  if fd′ = ⊥ : //If message is new
    if ¬Open(c_m, m, r) :   return ⊥
    return m, fd ← (σ, src, c_m, r)
  else :   (σ′, src′, c′_m, r′) ← fd′
    if ¬Open(c_m, ⊥, r) :   return ⊥
    if ¬Open(c′_m, m, r′) :   return ⊥
    if ¬Vf(pk, σ′, (c′_m, src′)) :   return ⊥ //verify platform signature
    return (m, fd′)   //msg = (m, fd′) will be used to forward the message
```

Figure 3: Tree-linkable construction.

Theorem 4.4. *If for any efficient adversary, the messaging system's encryption scheme $\mathcal{E}$ satisfies deniability*[1] *and the platform's encryption scheme $\mathcal{P}$ is CPA-secure, then the scheme is deniable (Definition 3.8).*

5 TREE-UNLINKABLE SOURCE-TRACKING

Intuitively, we could achieve tree-unlinkability in the scheme described in Section 4 if there was a way to re-randomize the forwarding data *fd* each time a message is forwarded. However, such a re-randomization is not easily achieved because re-randomizing the platform's signature is not sufficient to make forwarding data unlinkable. Both the signature *and* the signed contents need to be re-randomized without revealing the contents to the platform.

Our tree-unlinkable construction preserves the basic framework of the tree-linkable scheme in that forwarding data consists of a "signature" (in this case a MAC) on a commitment to the message and an encryption of the source user and optional metadata. Users re-randomize this forwarding data by re-randomizing the contents of the signature and proving the validity of the re-randomized values in zero-knowledge to the platform, which then issues the user a fresh signature on those values.

Our scheme can be thought of as similar to a keyed-verification anonymous credential scheme [4–6] that we modify to allow for anonymous and unlinkable credential delegation via forwarding. The attributes included in each credential are an encryption of the source identity and metadata as well as a hash of the message content. Since our attributes are a mix of group elements and scalars and we need users to efficiently prove properties about these attributes to the platform in zero-knowledge, we use the keyed-verification anonymous credential scheme presented in [5]. This construction relies on an algebraic MAC rather than a signature for the credential. This is fine for us because only the platform will need to distribute and verify forwarding credentials.

5.1 Tools

Zero Knowledge Proofs. We use zero knowledge proofs, denoted with the standard Camenisch-Stadler notation ([3]), i.e.,

$$\pi \leftarrow PK\{(\text{secrets}) : \text{expressions with secret and public vals}\}$$

When verifying a proof π, we use the notation $\mathsf{Vf}(\pi, [\text{optional: } P_1, P_2, \ldots])$, where $P_1, P_2, \ldots$ are any public values used in the proof.

Algebraic MAC. The main building block of our scheme is an algebraic MAC construction from Chase et al. [5]. Our particular instantiation requires a MAC on three attributes, two of which are group elements encoding the source information about a message (E_1 and E_2), and the other a scalar corresponding to a hash of the message plaintext m, which we refer to by $d \leftarrow H(m)$.

This MAC requires a group $\mathbb{G}$ of prime order q where the discrete log problem is hard and 10 (public) fixed group elements. Because we use most of the MAC functionality as a black-box in our scheme, we will only deal with five of these parameters explicitly, and refer to them as $G, G_d, G_{y_1}, G_{y_2}, G_{y_3}$. These parameters correspond to the parameters of the same name described in Section 3.1 of [5].

[1]See Appendix A.2 for a formal definition of deniability for encrypted messaging schemes.

In an effort to reduce redundancy and keep our scheme simple, we use this MAC as a black-box via the following functions:

- $\mathsf{KGen}_{MAC}(params) \rightarrow \mathsf{sk}_{MAC}$: Generates a MAC key.
- $\mathsf{issue}(d, E_1, E_2, \mathsf{sk}_{MAC}) \rightarrow \sigma, \pi_{issue}$: This function issues a MAC, σ, on the attributes d, E_1, and E_2, which are provided in the clear, as well as a proof that the MAC has been computed correctly, π_{issue} (See [5], Section 3.2). When verifying the unblinded MAC on attributes $info = (d, E_1, E_2)$, we use the verification function $\mathsf{Vf}(\sigma, \pi_{issue}, info)$.
- $\mathsf{blindIssue}(ct_d, ct_{E_1}, ct_{E_2}, \mathsf{pk}, \mathsf{sk}_{MAC}) \rightarrow ct_\sigma, \pi_{issue}$: Given ElGamal ciphertexts of each attribute encrypted under pk, this function blindly issues an encrypted MAC ct_σ encrypted under the same key. π_{issue} is a proof that the MAC is well-formed. (See [5], Section 5.10).
- $\mathsf{Vf}_{issue}(\pi_{issue}, ct_\sigma, ct_d, ct_{E_1}, ct_{E_2}, \mathsf{sk}) \rightarrow \sigma$ or $\perp$: This is called by a user to verify that a blindly issued MAC is well-formed. If the proof is correct, the function returns the decrypted MAC. (See [5], Section 5.10).
- $\mathsf{prepPresent}(\sigma, d, E_1, E_2) \rightarrow C_d, C_{E_1}, C_{E_2}, z, \pi_{present}$: When a user wants to prove to a platform that it has a valid MAC, it uses this function to commit to its attributes and MAC with opening z, and a proof $\pi_{present}$ that it has a valid MAC for these commitments. The commitments have the form $C_{E_1} = G^z_{y_1} E_1, C_{E_2} = G^z_{y_2} E_2, C_d = G^z_{y_3} G^d_d$. We include these commitments separately because they are later used for a second proof of the MAC contents, but assume all necessary commitments are included in $\pi_{present}$. (See [5], Section 3.2).
- $\mathsf{blindVf}(\pi_{present}, C_F, \mathsf{sk}_{MAC}) \rightarrow$ valid or $\perp$: Verifies that $\pi_{present}$ is a valid proof that the user holds a MAC on the committed attributes C_F. (See [5], Section 3.2).

Protocols using El-Gamal Encryption. Chase et al.'s algebraic MAC (Section 5.1) relies on a number of protocols based on manipulating El-Gamal ciphertexts, variants of which will also be used in our scheme. These protocols all rely on the homomorphic properties of El-Gamal encryption, which allow users to re-randomize ciphertexts without knowledge of the platform's secret key.

Similarly to the Chase et al. scheme, re-randomizing El-Gamal ciphertexts will enable blind issuance and unlinkable presentation of a MAC on hidden attributes in our scheme. We will use their techniques to prove that a user possesses valid credentials to forward a message. Whereas the original Chase et al. paper shows how to prove that a commitment commits to the same value as a given ciphertext, our protocol will augment theirs and show that a commitment commits to a *re-randomization* of a given ciphertext. We need the re-randomization to render the encryption of a message author's identity unlinkable to past forwards of the same message.

5.2 Our Construction

Our scheme requires a group $\mathbb{G}$ of prime order q in which the discrete log problem is hard and the 10 fixed group elements required by the MAC described in Section 5.1.

We assume that each message m can be hashed to a representation in $\mathbb{Z}_q$, which we denote as $d \leftarrow H(m)$. We also assume that each (src, md) pair corresponding to a source user and metadata value have a reversible representation in $\mathbb{G}$. Our scheme can easily

```
present$(d, E_1, E_2, \sigma, \mathsf{pk} = Y)$
$(C_d, C_{E_1}, C_{E_2}, z, \pi_{present}) \leftarrow \mathsf{prepPresent}(\sigma, d, E_1, E_2)$
//Additional commitments for proving re-randomization
$(z', rnd) \xleftarrow{R} \mathbb{Z}_q^2$
$C'_d \leftarrow C_d G_{y_3}^{z'}$
$C'_{E_1} \leftarrow G_{y_1}^{z'} G^{rnd} C_{E_1}, C'_{E_2} \leftarrow G_{y_2}^{z'} Y^{rnd} C_{E_2}$
$\pi_{rerand} = PK\{(z', rnd) : C'_d / C_d = G_{y_3}^{z'}$
        $\wedge C'_{E_1}/C_{E_1} = G_{y_1}^{z'} G^{rnd} \wedge C'_{E_2}/C_{E_2} = G_{y_2}^{z'} Y^{rnd}\}$
$C_F \leftarrow (C_d, C_{E_1}, C_{E_2})$
$C'_F \leftarrow (C'_d, C'_{E_1}, C'_{E_2})$
$o_f \leftarrow (z + z', d, (E_1 G^{rnd}, E_2 Y^{rnd}))$
return $(C_f, C'_f, o_f, \pi_{present}, \pi_{rerand})$
```

Figure 4: Construction of a sender's proof in our tree-unlinkable scheme. This function constructs a proof that a user has a valid forwarding credential, and additionally that C'_F is a commitment to the same message and a re-randomization of the source of the forwarding credential in question.

```
redeem$(C_A, src, o_A, C_F, o_F, type, \mathsf{pk} = Y)$
$(C_d, C_{E_1}, C_{E_2}) \leftarrow C_F, (E_1^{(A)}, E_2^{(A)}) \leftarrow src$
$(z_F, d_F, E_1^{(F)}, E_2^{(F)}) \leftarrow o_F, (z_A, d_A) \leftarrow o_A$
$h, r_1, r_2, r_3, rnd \xleftarrow{R} \mathbb{Z}_q, H \leftarrow G^h$
$(A_1, A_2) \leftarrow (G^{r_1}, H^{r_1} G_d^{d_{type}})$
$(B_1, B_2) \leftarrow (G^{r_2}, H^{r_2} E_1^{(type)} G^{rnd})$
$(C_1, C_2) \leftarrow (G^{r_3}, H^{r_3} E_2^{(type)} Y^{rnd})$
$P_1 \leftarrow H = G^h \wedge A_1 = G^{r_1} \wedge B_1 = G^{r_2} \wedge C_1 = G^{r_3}$
    $\wedge C_A = G_d^{d_A} G_{y_3}^{z_A} \wedge C_d = G_d^{d_F} G_{y_3}^{z_F}$
$P_A \leftarrow A_2 = H^{r_1} G_d^{d_A} \wedge B_2 / E_1^{(A)} = H^{r_2} G^{rnd}$
    $\wedge C_2 / E_2^{(A)} = H^{r_3} Y^{rnd}$
$P_F \leftarrow A_2 = H^{r_1} G_d^{d_F} \wedge B_2 / C_{E_1} = H^{r_2} G^{rnd} / G_{y_1}^{z_F}$
    $\wedge C_2 / C_{E_2} = H^{r_3} Y^{rnd} / G_{y_2}^{z_F}$
$\pi \leftarrow PK\{(h, r_1, r_2, r_3, rnd, d_A, z_A, d_F, z_F) :$
   $P_1 \wedge (P_A \vee P_F)\}$
$info \leftarrow (H, (A_1, A_2), (B_1, B_2), (C_1, C_2))$
return $(\pi, info, h, E_1^{(type)} G^{rnd}, E_2^{(type)} Y^{rnd})$
```

Figure 5: Proof to receive a message in our tree-unlinkable scheme. This function constructs a proof that the provided ciphertexts to be used for blind issuance of forwarding credentials encrypt either 1) the same message and re-randomized source as are committed to in C_F, or 2) commit to the same message as committed to in C_A and a re-randomization of *src*.

be extended if more than one element is needed to represent these values.

Keys. Our scheme assumes that a platform has an El-Gamal key pair $(y, Y = G^y)$ for $y \in \mathbb{Z}_q$. This is used for encrypting source ciphertexts. The platform also has a MAC key sk_{MAC}.

Authoring and forwarding data structure. Forwarding data *fd* held by a user consists of a tuple $(m, (E_1, E_2), \sigma)$, where m corresponds to the message plaintext, (E_1, E_2) is an ElGamal encryption of some (src, md) pair $S \in \mathbb{G}$, and σ is a MAC on attributes $d \leftarrow H(m), E_1, E_2$.

Users also store an authoring credential *ad*, which has the same form as the forwarding data, but for an unused message $\perp$. The authoring data is given to the user by the platform when the new user is created. The authoring data is used by the user to prove credentials on a new message so that new messages are indistinguishable from forwarded messages to the platform.

Once a user gets forwarding data after receiving a message, the forwarding data stays constant for the rest of its sending activity for that message. The MAC is presented blindly during future forwards, and so the possession of a valid forwarding credential can be proved to a platform multiple times unlinkably.

Sending a message. To send a message, the sender provides authoring information C_A and forwarding information $(C_F, C'_F, \pi_p, \pi_r)$. Only one of these is filled with useful information, depending on the type of message that the user wants to send. If the message is an authored message, the authoring information C_A is for the actual message plaintext, and the forwarding information is for an unused message $\perp$ (created from the user's authoring data *ad*). If the message is a forward, the message contents are swapped, and the relevant forwarding data *fd* is used to create the forwarding information.

The authoring information is a commitment, C_A, to the $\mathbb{Z}_q$ representation of the message, while the forwarding information is constructed in the present sub-protocol (Figure 4) and consists of C_F, which is a commitment to the forwarded message's $\mathbb{Z}_q$ representation and source ciphertext, as well as C'_F, which is a new commitment to the same message and a re-randomization of the source ciphertexts. The forwarding information also includes π_p, a proof that the user holds a valid MAC on the values stored in C_F, and π_r, which proves that C'_F commits to the same message and (re-randomized) source as C_F. We note that these two proofs can easily be combined into a single statement, but we've written them out separately to distinguish between scheme-specific statements and the standard MAC issuance proof.

The authoring commitment to a hash of a message $d \in \mathbb{Z}_q$ has the form of a standard Pedersen commitment [24] with bases G_{y_3} and G_d: $C_A = G_{y_3}^r G_d^d$ for a random $r \xleftarrow{R} \mathbb{Z}_q$. A forwarding commitment takes the form of a multi-attribute Pedersen commitment on d and two group elements E_1 and E_2 with bases $G_{y_3}, G_{y_1}, G_{y_2}, G_d$: $C_F = (G_{y_1}^r E_1, G_{y_2}^r E_2, G_{y_3}^r G_d^d)$. Hashing the plaintext to an element in $\mathbb{Z}_q$ ensures that these schemes are binding without the need for an additional group element to fix the opening to a particular value of r. We note that this structure can easily be extended to commit to multiple additional attributes that may be required to store additional metadata.

The sending user encrypts openings to C_A and C'_F alongside the message plaintext and passes that information on to the receiver.

Processing a message. To process a sent message, the platform first checks that the proofs π_p and π_r are valid for the provided commitments. If this is the case, it creates a new encryption *src* of the sending user's identity and some associated metadata to be

```
$\mathsf{KGen}(params)$
$(y, Y \leftarrow G^y) \xleftarrow{R} \mathsf{KGen}_{\mathcal{P}}(params)$
$\mathsf{sk}_{MAC} \leftarrow \mathsf{KGen}_{MAC}(params)$
return $(\mathsf{pk} = Y, \mathsf{sk} = (y, \mathsf{sk}_{MAC}))$
```

```
$\mathsf{newUser}(U_i, \mathsf{sk}, \mathsf{pk} = Y)$
$U_{new}$                                   $P_{new}(\mathsf{sk}, \mathcal{U})$
                                            if $U_i \in \mathcal{U}$ :   return $\perp$
                                            $r \xleftarrow{R} \mathbb{Z}_q, (E_1, E_2) \leftarrow (G^r, U_i Y^r)$
                                            $(\sigma, \pi_{issue}) \leftarrow \mathsf{issue}(\perp, E_1, E_2, \mathsf{sk}_{MAC})$
$info \leftarrow (\perp, E_1, E_2)$         $\xleftarrow{\sigma, \pi_{issue}, (E_1, E_2)}$
if $\mathsf{Vf}(\sigma, \pi_{issue}, info)$ :
    $ad \leftarrow (H(\perp), \sigma, (E_1, E_2))$
    return $ad$                             $\mathcal{U}.add(U_i)$
return $\perp$                              return $\mathcal{U}$
```

```
$\mathsf{AuthMsg}(U_s, U_r, \mathsf{pk} = Y)$
$U_{auth}(msg)$                             $P_{send}(\mathsf{sk}, md)$
$m, ad \leftarrow msg$
//create proof sender has valid authoring data
$out \leftarrow \mathsf{present}(ad, \mathsf{pk})$
$(C_F, C'_F, o_F, \pi_p, \pi_r) \leftarrow out$
$z \xleftarrow{R} \mathbb{Z}_q, d \leftarrow H(m)$
$o_A \leftarrow (z, d)$
$C_A \leftarrow G^z_{y_3} G^d_d$ //commit to new message
//send via underlying msg scheme
$e \leftarrow \mathsf{send}((m, o_A, o_F), U_s, U_r)$
$\xrightarrow{e, C_A, C_F, C'_F, \pi_p, \pi_r}$        if $!\mathsf{blindVf}(\pi_p, C_F, \mathsf{sk}_{MAC})$
                                              $\vee\ !\mathsf{Vf}(\pi_r, C_F, C'_F)$ : return $\perp$
                                            $r \xleftarrow{R} \mathbb{Z}_q, S \leftarrow (U_s, md)$
                                            $src \leftarrow (G^r, SY^r)$
return $e$                                  return $(pd = (C_A, src, C'_F), e)$
```

```
$\mathsf{FwdMsg}(U_s, U_r, \mathsf{pk} = Y)$
$U_{fwd}(msg)$                              $P_{send}(\mathsf{sk}, md)$
$(m, fd) \leftarrow msg$
$out \leftarrow \mathsf{present}(fd, \mathsf{pk})$
$(C_F, C'_F, o_F, \pi_p, \pi_r) \leftarrow out$
$z \xleftarrow{R} \mathbb{Z}_q, C_A \leftarrow G^z_{y_3} G^{\perp}_d$
$o_A \leftarrow (z, \perp)$
$e \leftarrow \mathsf{send}((m, o_A, o_F), U_s, U_r)$
$\xrightarrow{e, C_A, C_F, C'_F, \pi_p, \pi_r}$        if $!\mathsf{blindVf}(\pi_p, C_F, \mathsf{sk}_{MAC})$
                                              $\vee !\mathsf{Vf}(\pi_r, C_F, C'_F)$ : return $\perp$
                                            $r \xleftarrow{R} \mathbb{Z}_q, S \leftarrow (U_s, md)$
                                            $src \leftarrow (G^r, SY^r)$
return $e$                                  return $(pd = (C_A, src, C'_F), e)$
```

```
$\mathsf{RecMsg}(U_s, U_r, e, \mathsf{pk} = Y)$
$U_{rec}$                                   $P_{rec}(\mathsf{sk} = (y, \mathsf{sk}_{MAC}), pd)$
                                            $\xleftarrow{pd}$
$(C_A, src, C_F) \leftarrow pd$
$(m, o_A, o_F) \leftarrow \mathsf{receive}(e, U_s, U_r)$
$(z_A, d_A) \leftarrow o_A$
if $C_A \neq G^{z_A}_{y_3} G^{d_A}_d$ :   return $\perp$
$(z_F, d_F, E_1, E_2) \leftarrow o_F$
$(C_d, C_{E_1}, C_{E_2}) \leftarrow C_F$
if $C_d \neq G^{z_F}_{y_3} G^{d_F}_d \vee C_{E_1} \neq G^{z_F}_{y_1} E_1$
    $\vee\ C_{E_2} \neq G^{z_F}_{y_2} E_2$ :   return $\perp$
if $d = d_A \neq \perp \wedge d_F = \perp$ :
    $type \leftarrow A$   //new message
elseif $d_A = \perp \wedge d_F = d \neq \perp$ :
    $type \leftarrow F$   //forwarded message
else :   return $\perp$
//prove rerandomizations commit to $o_a$ or $o_f$
$o \leftarrow \mathsf{redeem}(C_A, src, o_A, C_F, o_F, type, \mathsf{pk})$
$(\pi, info, h, E'_1, E'_2) \leftarrow o$
$\xrightarrow{\pi, info}$
                                            $(C_A, src, C_F) \leftarrow pd$
                                            //verify proof that rerandomizations are valid
                                            if $!\mathsf{Vf}(\pi, info, C_A, src, C_F)$ :
                                                return $\perp$
                                            //blindly issue MAC on rerandomized values
                                            $(ct_\sigma, \pi) \leftarrow \mathsf{blindIssue}(info, \mathsf{sk}_{MAC})$
                                            $\xleftarrow{ct_\sigma, \pi}$
//verify proof that the new MAC is valid
$\sigma \leftarrow \mathsf{Vf}_{issue}(\pi, ct_\sigma, info, h)$
if $\sigma == \perp$ : return $\perp$
return $(m, fd = (H(m), (E'_1, E'_2), \sigma))$ //(m, fd) used to forward the message
```

```
$\mathsf{Report}(m, \mathsf{pk} = Y)$
$U_{rep}(fd)$                               $P_{rep}(\mathsf{sk} = (y, \mathsf{sk}_{MAC}))$
//prove knowledge of valid fd
$out \leftarrow \mathsf{present}(fd, \mathsf{pk})$
$\xrightarrow{out}$                         $C_F, (C'_F, o_F, \pi_p, \pi_r) \leftarrow out$
                                            if $!(\mathsf{blindVf}(\pi_p, C_F, \mathsf{sk}_{MAC}) \wedge \mathsf{Vf}(\pi_r, C_F, C'_F))$ :
                                                return $\perp$
                                            $(z, d', E_1, E_2) \leftarrow o_F$
                                            $d \leftarrow H(m)$
                                            if $C'_F \neq (G^z_{y_3} G^d_d, G^z_{y_1} E_1, G^z_{y_2} E_2)$
                                                $\vee\ d \neq d' \vee d = \perp$ :   return $\perp$
                                            return $E_2 / E^y_1$   //decrypt to recover (src, md)
```

Figure 6: Protocols for Scheme 2.

used if the message is new. It then passes along the platform data $pd = C_A, C'_F, src$ to the receiver.

Receiving a message. During the receipt of a message, the receiver presents encryptions of a message plaintext and *src* ciphertext and proves that these are either (re-randomizations of) the forwarding information committed to in C'_F or of the authored message committed to in C_A and the new *src* ciphertext. The platform then blindly issues the receiver a new MAC on those attributes. The helper function redeem (Figure 5), is used by the receiver to construct a proof to the platform that the values it would like to get a MAC on are valid.

Reporting a message. To report a message, a user presents the platform with the message plaintext, a re-randomization of the source ciphertext it would like to report, and a proof that it has a valid credential on those values. This proof is created in the same manner as when sending a message by calling the helper function present (Figure 4). After verifying the proof, the platform can decrypt the ciphertext to reveal the identity of the source.

Security. We prove the following theorems in Appendix C.

Theorem 5.1. *Assuming that the MAC of [5] satisfies blind issuance and anonymity as defined in [5], the platform's El Gamal encryption scheme is CPA-secure, the proof system employed is zero-knowledge, and $\mathcal{E}$ is an AE-secure encryption scheme, then Scheme 2 satisfies tree-unlinkable confidentiality.*

Theorem 5.2. *Assuming the correctness of the MAC's blind presentation protocol [5] and the completeness of the zero-knowledge proof system, Scheme 2 satisfies accountability.*

Theorem 5.3. *Assuming that the MAC presentation and issuance protocols of [5] satisfy unforgeability, the proof of knowledge system satisfies a strong extractability property, and the discrete log problem is hard in $\mathbb{G}$, then Scheme 2 is unforgeable.*

Theorem 5.4. *Assuming that the El Gamal encryption scheme $\mathcal{P}$ is CPA-secure and $\mathcal{E}$ is deniable, Scheme 2 is deniable.*

6 IMPLEMENTATION AND EVALUATION

We implemented both our constructions in Rust using the `double-ratchet` crate, an implementation of Signal's Double Ratchet protocol, as a baseline implementation of a messaging scheme [39]. The implementation of our tree-linkable scheme relies on the `ed25519-dalek` crate, which provides an implementation of ed25519 signing and verification [20]. For the tree-unlinkable scheme, we used `curve25519-dalek`'s implementation of curve25519 [21] to implement the algebraic MAC of Chase et al. [5], El-Gamal encryption and verification, and the necessary zero-knowledge proofs, which we implemented using the `zkp` crate [9]. The `zkp` crate does not currently provide functionality to prove disjunctions of statements, so to implement the receive proof (Figure 5), we modified the `zkp` prover and verifier to support the OR-protocol for Schnorr proofs [2]. Only our tree-linkable implementation includes report metadata, which we set to be a UNIX timestamp.

We evaluated our implementations on a variety of message lengths, ranging from 10 to 8000 bytes. Because we found that increased message lengths impacted runtime by <1% over this range, we only provide data for 1KB messages. Evaluation was performed using Criterion on an Intel i7-6700 processor @ 3.40GHz running Ubuntu Linux, and all estimates are computed from the average of at least 300 trials, where the number of iterations was determined by the number sufficient to achieve a <1% margin of error (p = 0.05). We conclude that our tree-linkable scheme does not significantly increase computation or communication costs in comparison to schemes deployed today that do not support source-tracking.

	No Tracking	Tree-Linkable	Tree-Unlinkable
Send	175 μs	201 μs	2.7 ms
Rec. (fresh)	116 μs	181 μs	6.4 ms
Rec. (forward)	116 μs	237 μs	6.3 ms
Server	N/A	20 μs	4.1 ms
Report	N/A	57 μs	2.6 ms

Table 1: Computation time for using our constructions to send 1KB messages compared to a standard double ratchet end-to-end encrypted message. Although the cost of sending a message increases with message size, we found that the increase was always less than 1% when going from 10B to 1KB messages.

	Tree-Linkable	Tree-Unlinkable
Send	256B	712B
Rec.	320B	1688B
Report	160B	648B

Table 2: Additional communication incurred to send a message using our constructions compared to sending the message via a standard double ratchet end-to-end encrypted message.

We compare our schemes to a baseline messaging scheme without source-tracking in Table 1. The times for server-side computation and reporting reflect the additional computations incurred by our scheme. We do not include a baseline time for a scheme with no tracking for these values, as there are no server-side operations to support source-tracking in the baseline.

While our tree-linkable scheme results in increased client-side costs, the overall client-side cost is still on the same order of magnitude as a scheme without source-tracking, and costs remain concretely very low. The costs on the server-side, where cost increases are much more sensitive, are minimal by comparison. The majority of the cost of sending messages in this scheme is the necessary cost of delivering a message in the underlying messaging scheme. Note that we separately report receiving times for fresh and forwarded messages. The different times do not open a timing side channel, as the components of each protocol that involve interaction with the server are identical. A constant-time implementation would incur the greater of the two costs for each message.

Our tree-unlinkable scheme has costs that are an order of magnitude higher than the linkable scheme, but concrete costs still remain on the order of milliseconds for each operation.

Table 2 Shows the additional communication we incur compared to a standard double ratchet end-to-end encrypted messaging scheme that does not use source-tracking. The increase in communication costs across our two schemes mirrors the increase in computation costs. Although the greatest communication increase is in the process of receiving a message, the data that clients must keep in order to report a message is only 128 Bytes in the tree-linkable scheme and 136 Bytes in the tree-unlinkable scheme.

Comparison to prior work. The only prior work that considers finding the source of a forwarded message is the traceback scheme of Tyagi et al. [33]. Whereas their work considers the setting where the system wishes to reveal all the users who received a particular message, we explicitly aim to prevent leaking this information. Despite the difference in security properties, it is worthwhile to compare our tree-linkable scheme to the path-traceback of Tyagi et al. (their more efficient scheme) to understand the performance tradeoffs between the two approaches.

Traceback incurs lower client overhead than our scheme, requiring less than 8μs overhead to send a message and less than 2μs to receive one, meaning their client-side sending and receiving costs almost match the performance of a messaging scheme with no tracking as shown in Table 1. Per-message communication costs for traceback are also lower. Sending and receiving a message require an additional 64 and 48 bytes of communication, respectively (compared to 256 and 320 bytes for tree-linkable tracing), while a report consists of only 16 bytes in addition to the message plaintext. A report in our tree-linkable scheme requires 160 additional bytes.

The lower client costs to support message traceback come at the cost of increased server side storage and report verification costs. First, finding the source of a reported message using traceback requires computation linear in the length of the forwarding chain. At approximately 90μs per forward, the report verification cost is comparable to ours (57μs on a slightly faster processor) for a fresh message but becomes worse if a message is forwarded several times. The time to find the source of a reported message in our schemes is constant, regardless of the number of forwards.

More importantly, the traceback scheme requires the server to store 36 Bytes per message sent. The server cannot know when a client has read or deleted a message, so it must conservatively store tracing information for as long as the message should be traceable. At the scale of a messaging system like WhatsApp that delivers on the order of 100 billion messages a day [29], this requires an additional 3.6 Terabytes of storage per day. On the other hand, our source-tracking schemes require no persistent server-side storage.

ACKNOWLEDGMENTS

This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

REFERENCES

[1] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. 2019. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Springer, Cham, Darmstadt, 129–158.

[2] Dan Boneh and Victor Shoup. 2020. A Graduate Course in Applied Cryptography.

[3] Jan Camenisch and Markus Stadler. 1997. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science* 260 (1997).

[4] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. 2014. Algebraic MACs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Scottsdale, 1205–1216.

[5] Melissa Chase, Trevor Perrin, and Greg Zaverucha. 2020. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1445–1459. https://doi.org/10.1145/3372297.3417887

[6] David Chaum. 1985. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM* 28, 10 (1985), 1030–1044.

[7] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A formal security analysis of the signal messaging protocol. *Journal of Cryptology* 33, 4 (2020), 1914–1983.

[8] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, San Jose, 321–338.

[9] Henry de Valence. 2019. zkp: Experimental zero-knowledge proof compiler in Rust macros. https://github.com/hdevalence/zkp.

[10] Mario Di Raimondo and Rosario Gennaro. 2009. New Approaches for Deniable Authentication. *Journal of Cryptology* 22, 4 (Oct. 2009), 572–615. https://doi.org/10.1007/s00145-009-9044-3

[11] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. 2005. Secure Off-the-Record Messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*. Association for Computing Machinery, New York, NY, United States, 81–89.

[12] Yevgeniy Dodis, Peul Grubbs, Thomas Ristenpart, and Joanne Woodage. 2018. Fast message franking: From invisible salamanders to encryptment. In *Annual International Cryptology Conference*. Springer, Springer, Cham, Santa Barbara, 155–186.

[13] Marc Fischlin and Sogol Mazaheri. 2015. Notions of Deniable Message Authentication. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society (WPES '15)*. Association for Computing Machinery, New York, NY, USA, 55–64. https://doi.org/10.1145/2808138.2808143

[14] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. 2017. Message Franking via Committing Authenticated Encryption. *IACR Cryptol. ePrint Arch.* 2017 (2017), 664.

[15] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. 2021. 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/

[16] Anunay Kulshrestha and Jonathan Mayer. 2021. Identifying Harmful Media in End-to-End Encrypted Communication: Efficient Private Membership Computation. In *USENIX Security*. USENIX, Virtual Event.

[17] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. 2017. Atom: Horizontally Scaling Strong Anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. Association for Computing Machinery, New York, NY, United States, 406–422.

[18] Albert Kwon, David Lu, and Srinivas Devadas. 2020. XRD: Scalable Messaging System with Cryptographic Privacy. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*. USENIX, Santa Clara, 759–776.

[19] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*. USENIX, Carlsbad, 711–725.

[20] Isis Lovecruft. 2020. ed25519-dalek: Fast and efficient ed25519 signing and verification in Rust. https://github.com/dalek-cryptography/ed25519-dalek.

[21] Isis Lovecruft and Henry de Valence. 2020. curve25519-dalek: A pure-Rust implementation of group operations on Ristretto and Curve25519. https://github.com/dalek-cryptography/curve25519-dalek.

[22] Moxie Marlinspike. 2013. Simplifying OTR Deniability.

[23] Movie Marlinspike and Trevor (ed.) Perrin. 2016. Signal » Specifications » The X3DH Key Agreement Protocol. https://signal.org/docs/specifications/x3dh/. (Accessed on 09/03/2020).

[24] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91 (Lecture Notes in Computer Science)*, Joan Feigenbaum (Ed.). Springer, Berlin, Heidelberg, 129–140. https://doi.org/10.1007/3-540-46766-1_9

[25] Trevor Perrin and Moxie Marlinspike. 2016. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/.

[26] Katitza Rodriguez and Seth Schoen. 2020. 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.

[27] Prasanto K Roy. 2019. Why India wants to track WhatsApp messages. https://www.bbc.com/news/world-asia-india-50167569.

[28] Manish Singh. 2020. India likely to force Facebook, WhatsApp to identify the originator of messages. https://techcrunch.com/2020/01/21/india-likely-to-force-facebook-whatsapp-to-identify-the-originator-of-messages/.

[29] Manish Singh. 2020. 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/.

[30] Udbhav Tiwari and Jochai Ben-Avie. 2020. Mozilla's analysis: Brazil's fake news law harms privacy, security, and free expression. https://blog.mozilla.org/netpolicy/2020/06/29/brazils-fake-news-law-harms-privacy-security-and-free-expression/.

[31] Yiannis Tsiounis and Moti Yung. 1998. On the security of ElGamal based encryption. In *International Workshop on Public Key Cryptography*. Springer, Springer-Verlag, Berlin, 117–134.

[32] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. 2019. Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption. *IACR Cryptol. ePrint Arch.* 2019 (2019), 565.

[33] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. 2019. Traceback for End-to-End Encrypted Messaging. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 413–430. https://doi.org/10.1145/3319535.3354243

[34] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Perl Henning, Ian Goldberg, and Matthew Smith. 2015. SoK: Secure Messaging. In *2015 IEEE Symposium on Security and Privacy*. IEEE, San Jose, 232–249.

[35] Nik Unger and Ian Goldberg. 2015. Deniable Key Exchanges for Secure Messaging. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1211–1223. https://doi.org/10.1145/2810103.2813616

[36] Nik Unger and Ian Goldberg. 2018. Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging. *Proceedings on Privacy Enhancing Technologies* 2018, 1 (Jan. 2018), 21–66. https://doi.org/10.1515/popets-2018-0003

[37] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP 2015, Monterey, CA, USA, October 4-7, 2015*. Association for Computing Machinery, New York, NY, United States, 137–152.

[38] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. 2020. On the Cryptographic Deniability of the Signal Protocol. In *ACNS 2020*. Springer, Rome, Italy, 188–209.

[39] S.R. Verschoor. 2020. double-ratchet: Pure Rust implementation of the Double Ratchet algorithm. https://github.com/sebastianv89/double-ratchet.

A DEFERRED DEFINITIONS

A.1 Platform Confidentiality

The platform confidentiality game gives the adversary oracles goodSend and goodRec for sending messages between honest users and malSend and malRec to send a malicious from or to a malicious user, respectively. We separate the sending and receiving of messages into separate oracles to allow the adversary to launch attacks that involve delivering message data to unintended recipients or otherwise tampering with the message delivery process. Additionally, the report oracle allows a user to report a message to the platform.

To further strengthen the adversary, we allow for a platform to create a set $\mathcal{U}_{mal}$ of platform-controlled users that can collude with the platform using the getUser oracle passed with $malUser$ set to true. We define the set of honest users $\mathcal{U}_{honest}$ as the set of users not controlled by the adversary, each created by a call to getUser with $malUser$ set to false. These honest users are created by running the newUser protocol with the adversary as the platform, and adding the new user to the set of users only if the protocol is successful. The adversary is given access to oracles send$(\cdot, U, \cdot)$ and receive$(\cdot, \cdot, U)$ for $U \in \mathcal{U}_{mal}$ to send and receive messages from malicious users in the underlying messaging scheme.

As in the user confidentiality game, messages are identified by a unique identifier mid, trees are identified by tid, the identifier of the message at their root. When set to $\perp$, the tid value acts as a flag for messages that can't be revealed to be the adversary without allowing it to trivially win the confidentiality game.

The game keeps track of messages sent and received in tables T_{send} and T_{rec}, respectively. The table T_{auth} additionally keeps track of authoring data (ad) received by honest users when they author a new message. The goodSend oracle represents the challenge portion of the game. It allows the adversary to choose two messages c_0 and c_1, one of which will actually be sent between two honest users, and the adversary must guess which message was sent. The c_is include an associated value c_i[type] that determines whether they are new messages or forwarded messages. If the type is new, then c_i is simply a message plaintext to be sent. Otherwise, c_i is the mid value for a message to be forwarded.

A.2 Deniability

Here, we present an extended discussion of our deniability definitions discussed in Section 3.4, and present formalizations of deniability for source-tracking schemes.

Deniability, or the guarantee that only the messaging platform can prove that a sender sent a message, is a common goal of many secure messaging applications [22]. Because source-tracking is designed to be applied on top of these existing messaging applications, it is important that source-tracking schemes preserve the messaging application's deniability guarantees.

One of the key settings where deniability is important is protection for whistleblowers who use the messaging application to anonymously relay sensitive information. From this perspective, deniability for source-tracking schemes becomes particularly important because we want to guarantee that not only the source of the forwarded message can deny sending it, but also that intermediate users along the path who may have reported or forwarded the message to an authority can deny this action if their messages are compromised.

Prior works [10, 11, 13, 32, 34] have investigated the large space of potential deniability definitions and their associated tradeoffs with notions of authentication and unforgeability.

While there are many different flavors of deniability definitions that could be extended to source-tracking, we chose to focus on definitions that were reasonable with respect to the current deniability guarantees offered by secure messaging systems. *Online deniability*, which guarantees a user can maintain deniability even when a third party interacts with it during the protocol to gain evidence of the user's participation, is a strong form of deniability that has been shown not to hold for the Signal messaging protocol [35, 36]. Instead, we focus on *offline deniability*, which guarantees a user can deny participating in a conversation in the event that a transcript of the exchange is provided to a third party. This form of deniability has been shown to hold for the Signal protocol under strong assumptions [34, 38]. We show that our source-tracking schemes satisfy the strongest form of offline deniability using the simulated transcript approach of [11], meaning that any user can forge a transcript of a forwarding path and report that is indistinguishable from a real transcript to a third party who holds the long-term keys for all involved users, assuming that the underlying messaging scheme satisfies this guarantee as well. This is stated formally in the following definition:

Definition A.1 (Deniability of an Encrypted Messaging Scheme)**.** We say an encrypted messaging scheme

$$\mathcal{E} = (\mathsf{send}(m, U_s, U_r), \mathsf{receive}(ct, U_s, U_r))$$

```
$PCONF^{\mathcal{A},b}_{ST,\mathcal{E}}$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KGen}(params)$
$b' \leftarrow \mathcal{A}^{O}(\mathsf{pk},\mathsf{sk})$
return $b'$
```

```
$\mathsf{getUser}(U, isMal)$
if $U \in \mathcal{U}_{honest} \cup U_{mal}$ :
    return $\perp$
if $isMal$ :
    return $U_{mal}.add(U)$
$ad \leftarrow \langle U_{new}, \mathcal{A}\rangle(U, \mathsf{pk})$
if $ad = \perp$ : return $\perp$
$T_{auth}[U] \leftarrow ad$
return $U_{honest}.add(U)$
```

```
$\mathsf{report}(mid)$
if $mid \notin T_{rec}$ :    return $\perp$
$(U_s, U_r, m, fd, tid) \leftarrow T_{rec}[mid]$
//check if report contents can be revealed
if $tid = \perp$ :    return $\perp$
return $\langle U_{rep}(fd), \mathcal{A}\rangle(m, \mathsf{pk})$
```

```
$\mathsf{goodRec}(mid, e)$
if $mid \notin T_{send}$ :    return $\perp$
$(U_s, U_r, m, e', tid) \leftarrow T_{send}[mid]$
if $U_s \vee U_r \notin \mathcal{U}_{honest}$ :    return $\perp$
$(m', fd) \leftarrow \langle U_{rec}, \mathcal{A}\rangle(U_s, U_r, e, \mathsf{pk})$
if $fd = \perp$ :    return $\perp$
$mid' \xleftarrow{R} \{0,1\}^n$, if $e \neq e'$ : $tid \leftarrow \perp$
$T_{rec}[mid'] \leftarrow (U_s, U_r, m, fd, tid)$
return $mid'$
```

```
$\mathsf{goodSend}(U_s, U_r, c_0, c_1)$
if $U_s \vee U_r \notin \mathcal{U}_{honest}$ :    return $\perp$
$mid \xleftarrow{R} \{0,1\}^n$
for $i \in \{0,1\}$ :
    if $c_i[\mathsf{type}] = \mathsf{new}$ :
        $m_i \leftarrow c_i$
        $tid_i \leftarrow mid$
    else :
        $mid_i \leftarrow c_i$
        if $mid_i \notin T_{rec}$ :    return $\perp$
        $(U_s^{(i)}, U_r^{(i)}, m_i, fd_i, tid_i) \leftarrow T_{rec}[mid_i]$
        if $U_r^{(i)} \neq U_s$ : return $\perp$
if $m_0 \neq m_1 \vee tid_0 \neq tid_1 \vee tid_0 = \perp \vee tid_1 = \perp$ :
    $tid \leftarrow \perp$
if $c_b[type] = \mathsf{new}$ :
    $ad \leftarrow T_{auth}[U_s], msg \leftarrow (m_b, ad)$
    $(ad', e) \leftarrow \langle U_{auth}(msg), \mathcal{A}\rangle(U_s, U_r, \mathsf{pk})$
    $T_{auth}[U_s] \leftarrow ad'$
else :
    $msg \leftarrow (m_b, fd_b)$
    $(fd, e) \leftarrow \langle U_{fwd}(msg), \mathcal{A}\rangle(U_s, U_r, \mathsf{pk})$
    $T_{rec}[mid_b] \leftarrow (U_s^{(b)}, U_s, m_b, fd, tid_b)$
$T_{send}[mid] \leftarrow (U_s, U_r, e, m_b, tid)$
return $mid$
```

```
$\mathsf{malSend}(U_s, U_r, e)$
if $U_s \notin \mathcal{U}_{mal} \vee U_r \notin \mathcal{U}_{honest}$ : return $\perp$
$(m, fd) \leftarrow \langle U_{rec}, \mathcal{A}\rangle(U_s, U_r, e, \mathsf{pk})$
if $fd = \perp$ :    return $\perp$
$mid \xleftarrow{R} \{0,1\}^n$
$T_{rec}[mid] \leftarrow (U_s, U_r, m, fd, mid)$
return $mid$
```

```
$\mathsf{malRec}(U_s, U_r, c)$
if $U_s \notin \mathcal{U}_{honest} \vee U_r \notin \mathcal{U}_{mal}$ : return $\perp$
//case 1: c corresponds to a new message
if $c[type] = new$ :    $m \leftarrow c$
    $ad \leftarrow T_{auth}[U_s], msg \leftarrow (m, ad)$
    $(ad', e) \leftarrow \langle U_{auth}(msg), \mathcal{A}\rangle(U_s, U_r, \mathsf{pk})$
    $T_{auth}[U_s] \leftarrow ad'$
//case 2: c corresponds to a forward
else :    $mid \leftarrow c$
    if $mid \notin T_{rec}$ :    return $\perp$
    $(U_s', U_r', m, fd, tid) \leftarrow T_{rec}[mid]$
    if $U_r' \neq U_s \vee tid = \perp$ :    return $\perp$
    $msg \leftarrow (m, fd)$
    $fd', e \leftarrow \langle U_{fwd}(msg), \mathcal{A}\rangle(U_s, U_r, \mathsf{pk})$
    $T_{rec}[mid] \leftarrow (U_s', U_r', m, fd', tid)$
```

Figure 7: Platform confidentiality game and oracles. The adversary is also given access to $\mathsf{send}(\cdot, U, \cdot)$ **and** $\mathsf{receive}(\cdot, \cdot, U)$ **oracles for the underlying encrypted messaging scheme for all adversary-controlled users** $U \in \mathcal{U}_{mal}$**.**

satisfies deniability if there exists an efficient simulation algorithm $\mathsf{Sim}_{\mathcal{E}}(U_s, U_r, m)$ that produces a transcript of U_s sending the message m using $\mathcal{E}$ to U_r without access to the private keys of U_s or U_r such that this simulated transcript is indistinguishable from a real transcript to any efficient distinguisher with access to U_s and U_r's secret keys.

Substituting in a weaker assumption on the messaging scheme would just result in the same weakened guarantee for the overall system, so using this approach means that we can guarantee a source-tracking scheme preserves the offline deniability of any messaging scheme it is built on top of.

Following the approach of [32], we additionally show that in addition to *universal deniability*, where the forger has access to an arbitrary third party user and the discerning party has access to the long-term keys of all involved users, our schemes also satisfy *platform-compromise deniability*, which gives the forger and discerning party additional access to the platform's keys. This second form of deniability ensures that even in the event that the platform was compromised and its secret keys exposed, deniability can be preserved.

While not required in our definition for deniable source-tracking schemes, it is worth noting that our scheme constructions provide strong deniability protections for *reporters* of malicious messages, regardless of the deniability guarantees of the underlying messaging scheme. Neither scheme makes use of the messaging oracles or users' secret keys while making a report, and additionally anyone with knowledge of the forwarding data for a particular message can report it even if they themselves did not receive it, meaning that the pool of users who could have reported a message is not restricted to the users on the forwarding path of the message in question.

We present two security games to address each of these types of deniability: *UnivDEN* and *PlatDen* (Figure 8). Each game accepts the corresponding forgery algorithm that simulates a forwarding path's transcripts as a parameter. The games get access to a challenge oracle $\mathsf{Chal}(\cdot,\cdot,\cdot, type)$, where the value of *type* is either u (universal) or p (platform) depending on the type of game, as well as the ability to create new users of their choice.

The deniability challenge consists of the adversary presenting a query that consists of a message m, a path of users p, a forging user U_D, and a list of metadata *mds*. The challenge will either output the actual transcripts (Tr_r, Tr_f, and Tr_{rep}) and forwarding data resulting from sending the message along the path and then being reported by the last user, or a forged version constructed by inputting the query into the forgery algorithm, which is given access to the sending and receiving capabilities of U_D and a simulator for the underlying messaging scheme, $\mathsf{Sim}_{\mathcal{E}}$. The functions of this oracle are presented in Figure 8 as interactive protocols where $\mathcal{F}$ is the portion of the interaction that the forger can control.

Note that some of our security games are defined with interactive oracle functions. When using these functions as a sub-protocol in our proofs, we write the name of the function with its public parameters, and then a double headed arrow with $\langle P_1(secrets), O_s\rangle$ above it for each interaction, where $P_1(secrets)$ is the protocol that the adversary follows, and O_s represents whatever behavior is defined for the oracle (See Figures 9 and 10).

B SECURITY PROOFS FOR TREE-LINKABLE SOURCE-TRACKING

B.1 Confidentiality – Proof of Theorem 4.1

Proof. The proof of this statement follows immediately from combining the results of Lemmas B.1 and B.2, which prove user and platform confidentiality, respectively. □

Lemma B.1. *Assuming that $\mathcal{P}$ is CPA-secure, and the commitment and signatures schemes $\mathcal{C}$ and $\mathcal{S}$ are correct, the advantage of any efficient adversary against the lUCONF game for Scheme 1 is negligible.*

Proof. We construct a series of hybrid games to show that $lUCONF^1$ is indistinguishable from $lUCONF^0$ to any efficient adversary. This proves that the scheme satisfies tree-linkable user confidentiality.

G0: This game is identical to the standard *UCONF* game when $b = 0$.

G1: We modify $\mathsf{goodAuth}$ (oracle function for authoring a message between two honest users) and $\mathsf{malSend}$ (oracle function for sending a message from a malicious user) so that the *src* the platform computes and includes in the platform data is an encryption of some default user and metadata $U_D||md_D$ rather than the actual author of the message (the default data can, for example, be an all-zero string). Because the adversary does not have the secret key, and this value is never decrypted by the platform, this version of the game is indistinguishable from Game 0 by the CPA-security of the platform's encryption scheme, $\mathcal{P}$.

G2: We modify $\mathsf{goodFwd}$ (oracle function for forwarding a message between two honest users) to immediately abort if the user provides an $mid \in T_{rec}$ such that the signature or commitment included in the forwarding data for the associated message is invalid. By the definition of the scheme, any message stored in T_{rec} that's associated with an honest receiver was received successfully by that honest user who checked the correctly formed signature and commitment, so this will never happen, and this game is indistinguishable from Game 1.

G3: We again modify $\mathsf{goodFwd}$ so that after checking that the users associated with the given *mid* are correct and that $mid \in T_{rec}$, if m, fd, tid is the data associated with the provided *mid*, the oracle doesn't actually forward the message but instead just adds a new entry $T_{rec}[mid'] = (U_s, U_r, m, fd, tid)$ to the table. By definition of the scheme, the resulting entry in T_{rec} is identical to the entry if the message had actually been forwarded, and so this game is indistinguishable from Game 2 as the oracle does not modify any other persistent state outside of the table T_{rec}.

We note that after these modifications, the output of malRec (oracle function for forwarding a message to a malicious receiver, the function gets two potential forwards and uses the value of b to determine which one the adversary sees), which is the only oracle function that uses b to decide what to output, no longer depends on b because the forwarding data of all honestly sent messages have the same contents. This means that applying and identical series of hybrid steps to those described in G0 to G3 starting at $lUCONF^1$ instead of $lUCONF^0$ gets us to a game identical to Game 3 that is indistinguishable from $lUCONF^1$ to an efficient adversary.

Thus we have shown that we can go from $lUCONF^0$ to $lUCONF^1$ through a series of indistinguishable hybrids, and we can conclude that any adversary's advantage against the *lUCONF* game for Scheme 1 must be negligible. □

Lemma B.2. *Assuming that the commitment scheme is hiding and the messaging encryption scheme $\mathcal{E}$ is AE-secure, the advantage of any efficient adversary $\mathcal{A}$ against the PCONF game for Scheme 1 is negligible.*

Note that the lemma requires that the messaging scheme guarantee authenticated encryption. This sort of guarantee is common among most encrypted messaging schemes, such as Signal's Double Ratchet Protocol [23].

Proof. We first present a series of hybrid games that show an adversary cannot gain advantage by tampering with messages between honest users. By definition of the scheme, the platform is tasked with passing on the platform data (σ, src) to the receiving user, and the security game additionally gives the option for the platform to choose the message identifier e that the receiver gets as input. The platform could stray from the correct protocol by calling $\mathsf{goodRec}$ (oracle function for receiving a message between honest users) with an e that wasn't the same as the one outputted by the sending interaction associated with the *mid*, or by signing

```
$UnivDEN^{\mathcal{A},b}_{ST,\mathcal{E},\mathsf{UForge}}$
$(\mathsf{pk},\mathsf{sk}) \xleftarrow{R} \mathsf{KGen}(params)$
$b' \leftarrow \mathcal{A}^{O_\mathsf{u}}(\mathsf{pk})$
return $b'$
```

```
$PlatDEN^{\mathcal{A},b}_{ST,\mathcal{E},\mathsf{PForge}}$
$(\mathsf{pk},\mathsf{sk}) \xleftarrow{R} \mathsf{KGen}(params)$
$b' \leftarrow \mathcal{A}^{O_\mathsf{p}}(\mathsf{pk},\mathsf{sk})$
return $b'$
```

```
$\mathsf{Chal}(path, U_D, m, mds, type)$
//Construct path
$(U_0,...,U_k) \leftarrow path$
if $U_D, U_0, ..., U_k \notin \mathcal{U}$ :    return $\perp$
$ad \leftarrow T_{auth}[U_0], msg \leftarrow (m, ad)$
$(ad', pd, e) \leftarrow \langle U_{auth}(msg), P_{send}(\mathsf{sk}, mds[0])\rangle(U_0, U_1, \mathsf{pk})$
$(m, fd, Tr_r) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd)\rangle(U_0, U_1, e, \mathsf{pk})$
$output_0.append((Tr_r, fd)), T_{auth}[U_0] \leftarrow ad'$
for $i = 1, ..., k-1$ :
    $msg \leftarrow (m, fd)$
    $(fd', pd, e, Tr_f) \leftarrow \langle U_{fwd}(msg), P_{send}(\mathsf{sk}, mds[i])\rangle(U_i, U_{i+1}, \mathsf{pk})$
    $(m, fd, Tr_r) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd)\rangle(U_i, U_{i+1}, e, \mathsf{pk})$
    $output_0.append(Tr_f, fd', Tr_r, fd)$
$((src, md), Tr_{rep}) \leftarrow \langle U_{rep}(fd), P_{rep}(\mathsf{sk})\rangle(m, \mathsf{pk})$
$output_0.append(Tr_{rep}, (src, md))$
//construct forgery
if $type = \mathsf{u}$ :    $output_1 \leftarrow \mathsf{UForge}^{O(U_D)}(path, m, mds, \mathsf{pk})$
if $type = \mathsf{p}$ :    $output_1 \leftarrow \mathsf{PForge}^{O(U_D)}(path, m, mds, \mathsf{pk}, \mathsf{sk})$
return $output_b$
```

```
$\mathsf{getUser}(U)$
if $U \in \mathcal{U}$ :    return $\perp$
$(ad, \mathcal{U}') \leftarrow \langle U_{new}, P_{new}(\mathcal{U}, \mathsf{sk})\rangle(U, \mathsf{pk})$
$T_{auth}[U] \leftarrow ad, \mathcal{U}.add(U)$
return $ad$
```

```
$\mathsf{ForgeSend}(U_r, md)$
$(pd, e) \leftarrow \langle \mathcal{F}, P_{send}(\mathsf{sk}, md)\rangle(U_D, U_r, \mathsf{pk})$
if $U_r \neq U_D$ : return
$\langle \mathcal{F}, P_{rec}(\mathsf{sk}, pd)\rangle(U_D, U_D, e, \mathsf{pk})$

$\mathsf{Report}(m)$
$\langle \mathcal{F}, P_{rep}(\mathsf{sk})\rangle(m, \mathsf{pk})$

$e \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_s, U_r, m)$
```

Oracle functions $O(\cdot)$ for forgery algorithms. $\mathsf{Sim}_{\mathcal{E}}$ **simulates a transcript of the messaging scheme using the simulator guaranteed to exist by the deniability of the messaging scheme.**

Figure 8: Deniability games and oracle functions. Oracles O_{type} for $type = \mathsf{u}$ or p give the adversary access to the functions $\mathsf{getUser}(\cdot,\cdot)$, $\mathsf{Chal}(\cdot,\cdot,\cdot,\cdot,type)$**, and** $\mathsf{send}(\cdot, U, \cdot)$ **and** $\mathsf{receive}(\cdot,\cdot,U)$ **oracles for the underlying messaging scheme for all users $U \in \mathcal{U}$. Oracle $O(U_D)$ gives the forgery algorithms access to the** Send **and** Report **functions.**

a value other than the *src* value chosen by the platform and the commitment presented by the sender.

- G0: This initial game is identical to $PCONF^{\mathcal{A},0}_{ST,\mathcal{E}}$, the game when $b = 0$.
- G1: We add the additional condition to goodRec to abort immediately if the inputted e value has never been outputted by the sender U_s as part of a message to the receiver U_r at any point in the game. By definition, this can only happen during a call to goodSend because U_s and U_r are honest. This is indistinguishable from G0 because finding a valid e that was not sent from U_s to U_r contradicts the authenticated encryption properties of the messaging scheme, specifically its ciphertext integrity, and can therefore happen with only negligible probability.
  We note that if the adversary presents a different e but the game does not abort due to the above condition, e must have resulted from a call to goodSend between U_s and U_r earlier in the game, and so there exists some mid'' in T_{send} that stores e. The only difference between calling goodRec with the correct pair (mid'', e) instead of (mid, e) is that the tid will always be $\perp$, which only weakens the adversary's power because it can no longer see the message contents by having the message reported and so we can assume that $e = e'$ in all other cases.
- G2: In this game, if during goodRec the platform presents an honest user with platform data (σ, src) and a message identifier e such that σ is a signature on the pair (c_m, src), but (c_m, e) was never sent by the sending user, the game aborts. By the properties of Game 1, we know that the e must have been created by the sending user and so contains an encryption of the commitment c_m that the sender linked to e, and therefore the receipt of the message will fail when the receiving user compares the commitment that is signed by the platform to the value encrypted in e. Therefore, this game is indistinguishable from Game 3 because an adversary cannot create an otherwise valid message that fails for this reason. Similarly, the *src* value is chosen by the platform and sent over in the clear, so the platform gains no advantage by signing a value other than the correct *src*, because it already knows it will cause the signature verification to fail.

We conclude that a platform gains no advantage by passing a receiver something other than the platform data and message identifier outputted by following the honest protocol. We now present two hybrids that show forwarded messages are indistinguishable from authored messages.

G3: Game 3 is identical to Game 2, except that during goodSend, a forwarded message m with forwarding data σ, src, c_m, r that would normally be forwarded as $e, c_\perp$ where $c_\perp$ is a commitment to $\perp$ is instead forwarded with e, c'_m where c'_m is a new commitment to m. The honest receiver evaluates the message as if c'_m was the original commitment to $\perp$ (tracked by the oracle). By the hiding properties of the commitment, this is indistinguishable to an efficient adversary.

G4: Game 4 is identical to Game 3, but during goodSend, e is encrypted as $(m_0, \perp, c'_{m_0}, r)$ rather than $(m_0, fd, c_\perp, r)$. This is indistinguishable to an efficient adversary because of the authenticated encryption properties of $\mathcal{E}$.

The messages and associated data of Game 4 are all identically distributed to authored messages. We now show that message contents are hidden from the adversary as well.

G5: In this game, modify Game 4 so that when messages are sent via goodSend, the associated message identifier corresponds to an encryption of $(m_1, \perp, c_{m_1}, r)$ and the commitment is still the original commitment to m_0. The oracle keeps a table of the corresponding $(m_0, \perp, c_{m_0}, r)$ that should have been sent and evaluates the results based on that value. By the authenticated encryption properties of $\mathcal{E}$, this is indistinguishable from Game 4 because all that changes is the decryption of e.

G6: In this game, we switch the commitment sent in the clear, c_{m_0}, to instead be c_{m_1}, the commitment contained in e that commits to m_1. The message is evaluated in the standard way using the decryption of e. This is indistinguishable from Game 5 by the hiding properties of the commitment.

We now note that applying the same game hops 1 - 4 starting from the game where $b = 1$ instead of 0 are each indistinguishable by the same arguments, and result in a game identically distributed to Game 6. Therefore we have shown that both $PCONF^1_{ST,\mathcal{E}}$ and $PCONF^0_{ST,\mathcal{E}}$ are indistinguishable from Game 7 to an efficient adversary, and so they are also indistinguishable from each other.

This means an efficient adversary can gain at most negligible advantage in the PCONF game, and so Scheme 1 satisfies platform confidentiality. □

B.2 Accountability – Proof of Theorem 4.2

Proof. Suppose for purposes of contradiction we have an adversary that can win the *srcBIND* game against Scheme 1 with non-negligible probability. This means that an honest user successfully received a message, but then the report of that message failed.

First, we consider the conditions that would cause a report to fail, and then show that each can happen with only negligible probability assuming that the receipt was successful:

(1) The opening for the provided commitment was invalid.
This is checked by the receive protocol, and the protocol fails if the opening is invalid, so this will happen with probability zero on a message that was received successfully.
(2) The signature verification on the source and commitment fails.
This exact check is also performed by the receive protocol, so again there is zero probability that this check will fail on a report of a validly received message.
(3) The decryption of the source ciphertext fails.
By definition of the scheme, the platform only ever signs ciphertexts that it created, and can therefore be sure of their validity. Therefore, the probability that this occurs is upper bounded by the probability that an adversary could create a valid signature without the keys of the platform. By the unforgeability properties of the signature scheme, this happens with at most negligible probability.

We therefore conclude that the scheme satisfies accountability because if a message is received successfully, all three possibilities for a report failure happen with negligible probability. □

B.3 Unforgeability – Proof of Theorem 4.3

Proof. First, we remind ourselves of the conditions that must be met for an adversary to win the *unFORGE* game. After sending some amount of messages between honest and malicious users, the adversary must present forwarding data (σ, src, c_m, r) and a message m to the platform such that:

(1) c_m is a valid commitment with opening (m, r)
(2) σ is a valid signature on src and c_m
(3) src decrypts to an honest user U and metadata md such that U never sent m with metadata md during the adversary's initial queries.

We show that the probability an adversary can construct such an output is negligible via a series of hybrid games.

G0: This is identical to the original *unFORGE* game.

G1: We modify G0 to abort immediately and return 0 if σ was not created by the platform oracle during the query portion of the game. By the unforgeability of the signature, the probability that the adversary can create a new valid signature without the platform's secret keys is negligible, and so this game is indistinguishable from G0 to efficient adversaries.

G2: Suppose that the src value decrypts to U, md. We modify G1 to abort immediately if U did not send some message m' with commitment c_m and metadata md. By definition of the scheme, the platform would only create a valid signature linking U and md to the commitment c_m that U provided. By G1, the signature must have been created by the platform, so this additional condition happens with zero probability, and we conclude G2 is indistinguishable from G1.

G3: We modify G2 to immediately abort if m was not the plaintext of the message sent when U sent c_m and md. This can only happen if an adversary can find a second opening (m', r') for c_m that is different from the original opening created by U. By the binding property of the commitment scheme, this can happen with only negligible probability, and so G3 is indistinguishable from G2 to an efficient adversary.

In a series of three hops, we have reached a game where if the first two requirements described above are met and the U revealed is honest, then U must have sent m with metadata md at some point during the adversary's initial queries. This violates the third requirement, and so we conclude that there is no way to win Game

3. We have shown that *unFORGE* is indistinguishable from Game 3 to any efficient adversary, and so we can conclude that any efficient adversary can gain at most negligible advantage against the *unFORGE* game for Scheme 1, and therefore the scheme is unforgeable. □

B.4 Deniability – Proof of Theorem 4.4

Proof. The *UForge* and *PForge* algorithms for this scheme are presented in Figure 9.

Universal Deniability We note that the forwarding data and transcript contents created during any particular query to the challenge are discarded by definition of the scheme, so the output of any particular challenge query is independent of the queries that came before it. An efficient adversary can make at most a polynomial number of challenge queries, and so it suffices to show that no efficient adversary can gain a non-negligible advantage against the *UnivDen* game after seeing the output of a single query to the challenge. First, we review the form of the forged and unforged outputs of a query consisting of the path $path = U_0, ..., U_k$, forging user U_D, metadata $mds = md_0, ..., md_{k-1}$, and a message m. The output to this query will have the form

$$output = (Tr_{r,0}, fd_0, Tr_{s,0}, fd'_0, ..., Tr_{r,k-1}, fd_{k-1}, Tr_{rep}, (src, md)).$$

In the unforged version, we have that

$$Tr_{r,0}, fd_0 = (\sigma_0, src_0), (\sigma_0, src_0, c_{m,0}, r_0)$$

where $src_0 \leftarrow \mathsf{Enc}(\mathsf{k}, (U_0, md_0))$, $c_{m,0}, r_0 \leftarrow \mathsf{Commit}(m)$, and $\sigma_0 \leftarrow \mathsf{Sig}(\mathsf{sk}_s, (c_{m,0}, src_0))$.

All the forwarding data is the same for our scheme, so we have

$$fd_0 = fd'_0 = ... = fd_{k-1} = (\sigma_0, src_0, c_{m,0}, r_0)$$

Then, for each $i = 1, ..., k-1$, we have

$$Tr_{s,i-1}, Tr_{r,i} = (c_{m,i}, e_i), (\sigma_i, src_i)$$

Where $c_{m,i} \leftarrow \mathsf{Commit}(\perp)$, $src_i \leftarrow \mathsf{Enc}(\mathsf{k}, (U_i, md_i))$, $\sigma_i \leftarrow \mathsf{Sig}(\mathsf{sk}_s, (c_{m,i}, src_i))$, and $e_i \leftarrow \mathsf{send}((m, fd_0, c_{m,i}, r_i), U_i, U_{i+1})$.

Finally, the report consists of $(Tr_{rep}, (src, md))$ where Tr_{rep} is just the forwarding data, fd_0, and $(src, md) = (U_0, md_0)$.

The forged version is constructed in the exact same way, but each src_i for $i = 0, ..., k$ is instead computed from $\mathsf{Enc}(\mathsf{k}, (U_D, md_i))$, and $e_i \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_i, U_{i+1}, (m, fd_0, c_{m,i}, r_i))$ (where fd_0 is replaced with $\perp$ for $i = 0$).

We note that this means that except for the values of src_i and e_i for $i = 0, ..., k-1$, the forged and unforged outputs are identical.

We now present a hybrid argument to show that the distributions of the forged and unforged src_is and e_is are indistinguishable.

G0: This is identical to $UnivDen^0$, the unforged version of the game.

G1: We modify each e_i to be outputted by $\mathsf{Sim}_{\mathcal{E}}(U_i, U_{i+1}, \cdot)$ instead of the output of an actual send from U_i to U_{i+1}. By the deniability of the messaging scheme $\mathcal{E}$ (Definition 3.8), this is indistinguishable to any efficient adversary.

G2: We change the platform's protocol so that each src_i is an encryption of U_D, md_i rather than U_i, md_i. By the CPA-security of the platform's encryption scheme $\mathcal{P}$, these two ciphertexts are indistinguishable to an efficient adversary, and therefore G1 is indistinguishable from G2. We further note that this game is identical to $UnivDEN^1$, the forged version of the game.

We can therefore conclude that Game 0, or $UnivDen^0$ is indistinguishable from Game 2, or $UnivDen^1$, to any efficient adversary, and therefore any efficient adversary can have at most negligible advantage against the *UnivDen* game for Scheme 1.

Platform Compromise Deniability Similar to universal deniability, we note that it suffices to show that the advantage of an efficient adversary who makes a single query to the challenge oracle must be negligible.

We again consider a query of m, $path = (U_0, ..., U_k)$, forger U_D, and $mds = md_0, ..., md_{k-1}$ that results in the output $Tr_{r,0}$, $fd_0, Tr_{s,0}, fd'_0, ..., Tr_{r,k-1}, fd_{k-1}, Tr_{rep}, (src, md)$.

The output of the unforged version of the game is the same as described in the Universal Deniability proof.

Intuitively, the scheme should satisfy platform compromise deniability because the behavior of the platform can be perfectly imitated by anyone who has access to a leaked platform key. In this case, by definition of the PForge algorithm (Figure 9) the output follows the exact same distribution as the unforged version except that each e_i is the output of the simulator $\mathsf{Sim}_{\mathcal{E}}(U_i, U_{i+1}, \cdot)$ rather than actually being sent by the messaging scheme from U_i to U_{i+1}. By the deniability of the messaging scheme, this difference is indistinguishable to any efficient adversary, and so we conclude that for any efficient adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{platden}}_{ST,\mathcal{E},\mathsf{PForge}}(\mathcal{A}) \leq \mathsf{negl}(\lambda).$$

Therefore, the scheme is deniable because it satisfies universal and platform-compromise deniability.

□

C SECURITY PROOFS FOR TREE-UNLINKABLE SOURCE-TRACKING

C.1 Confidentiality – Proof of Theorem 5.1

Proof. The proof follows immediately from the results of Lemmas C.1 and C.2. □

C.1.1 User Confidentiality.

Lemma C.1. *Assuming the CPA-security of the platform's El Gamal scheme, any efficient adversary has negligible tree-unlinkable advantage against the UCONF game.*

Proof. We note that conditioned on the *src* ciphertext values (E_1, E_2) used in the forwarding data of messages sent via malRec, the adversary's view in $UCONF^0$ and $UCONF^1$ is identical.

It therefore suffices to show that the difference in ciphertexts doesn't provide the adversary with a non-negligible advantage. We do this below by constructing a series of hybrid games to show that $UCONF^0$ and $UCONF^1$ are indistinguishable to an efficient adversary.

G0: This game is identical to the standard $UCONF^0$ game (when $b = 0$).

G1: We modify malRec (oracle function that sends one of two messages from an honest sender to a malicious receiver) so that if the message would normally be sent with forwarding data that includes the source ciphertext (E_1, E_2), which is an encryption of the source information $S_0 \in \mathbb{G}$, we instead

```
UForge($path, path, m, mds$, pk)
$(U_0, ..., U_k) \leftarrow path$
Send($U_D, mds[0]$) :
    $Tr_s \quad \underset{\longleftrightarrow}{\langle U_{auth}(m, \perp), O_s)\rangle}$
    $fd, Tr_r \quad \underset{\longleftrightarrow}{\langle U_{rec}, O_s\rangle}$
$output.append(Tr_r, fd)$
for $j = 1, ..., k-1$ :
    Send($U_D, mds[j]$) :
        $Tr_s \quad \underset{\longleftrightarrow}{\langle \mathsf{Fwd}(U_j, U_{j+1}, m, fd), O_s)\rangle}$
        $Tr_r \quad \underset{\longleftrightarrow}{\langle \mathsf{Rec}, O_s\rangle}$
    $output.append(Tr_s, fd, Tr_r, fd)$
Report($m$) :
    $Tr_{rep} \quad \underset{\longleftrightarrow}{\langle U_{rep}(fd), O_s\rangle}$
$output.append(Tr_{rep}, (U_0, mds[0]))$
return $output$
```

```
Rec
// Does nothing other than add pd to the transcript.
    $\xleftarrow{\quad pd \quad}$
```

```
Fwd($U_s, U_r, m, fd$)
$(c_m, r) \leftarrow$ Commit($\perp$)
$e \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_s, U_r, (m, fd, c_m, r))$
    $\xrightarrow{\quad (c_m, e) \quad}$
```

Fwd and Rec are helper functions defining how the forger interacts with the Send oracle during UForge.

```
PForge($path, m, mds$, pk, sk)
$(U_0, ..., U_k) \leftarrow path$
$src \leftarrow$ Enc(k, $(U_0, mds[0])$)
$(c_m, r) \leftarrow$ Commit($m$)
$\sigma \leftarrow$ Sig(sk$_s$, $(c_m, src)$)
$e \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_0, U_1, (m, \perp, c_m, r))$
$Tr_r \leftarrow (\sigma, src)$
$fd \leftarrow (\sigma, src, c_m, r)$
$output.append(Tr_r, fd)$
for $i = 1, ..., k-1$ :
    $src' \leftarrow$ Enc(k, $(U_i, mds[i])$)
    $(c'_m, r') \leftarrow$ Commit($\perp$)
    $\sigma' \leftarrow$ Sig(sk$_s$, $(c'_m, src')$)
    $e \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_i, U_{i+1}, (m, fd, c'_m, r'))$
    $Tr_s \leftarrow (c'_m, e), Tr_r \leftarrow (\sigma', src')$
    $output.append(Tr_s, fd, Tr_r, fd)$
$output.append(fd, (U_0, mds[0]))$
return $output$
```

Figure 9: Forgery Algorithms for Scheme 1.

replace (E_1, E_2) with a fresh encryption of S_0, (E'_1, E'_2), before sending.
This means that the only difference between G0 and G1 is that instead of seeing a re-randomization of the original source ciphertext upon receipt, the adversary sees a re-randomization of a fresh encryption of the same value. These two ciphertexts follow the same distribution, and are therefore indistinguishable to the adversary.

G2: We further modify malRec so that instead of a fresh encryption of the original source S_0, we make E'_1, E'_2 a fresh encryption of S_1, the user/metadata pair that would have been used in the case that $b = 1$. Because the adversary does not know the platform's secret key, this is indistinguishable from Game 1 by the CPA security of El Gamal encryption.

G3: This is the standard $UCONF^1$ game. The only difference in the adversary's view between Games 2 and 3 is that the source stored in the forwarding data used to send the message that the adversary receives during a call to malRec is a fresh encryption of S_1 in the case of Game 2, and a re-randomized encryption of S_1 in the case of Game 3. By the same argument as the hop from Game 0 to Game 1, this difference is indistinguishable to an efficient adversary.

We have shown that in each hop, the two consecutive games are indistinguishable to an efficient adversary. We can therefore conclude that $UCONF^0$ is indistinguishable from $UCONF^1$ to an efficient adversary, and therefore Scheme 2 satisfies tree-unlinkable user confidentiality. □

C.1.2 Platform Confidentiality.

Lemma C.2. *Assuming that the messaging system's encryption scheme $\mathcal{E}$ is a secure authenticated encryption scheme, the proof system used to prove attribute values on messages is zero-knowledge, and the algebraic MAC presented in [5] satisfies blind issuance and anonymous presentation properties, then the advantage of any efficient platform-confidentiality adversary for Scheme 2 is negligible.*

Note that the proof relies on the blind issuance and anonymity properties of the algebraic MAC presented in [5]. These are formally defined in [4] and discussed in [5]. Briefly, blind issuance guarantees that when the platform and user interact to issue a MAC on attributes known only to the user, the interaction is a secure two-party computation against malicious adversaries in which the user gets a valid MAC on their attributes and the platform learns nothing about the user's hidden attributes or the MAC that they receive. Anonymity guarantees that a user can prove to the platform they hold a valid MAC on particular attributes without revealing the MAC or hidden attributes to the platform.

Proof. We show that the two $PCONF$ games are indistinguishable to an efficient adversary via a series of game hybrids.

The first few hybrids show that the adversary can't gain a non-negligible advantage by not following the correct protocol. We begin by using the same approach as in the proof of Lemma B.2 to show that the adversary gains no advantage from calling goodRec (oracle function to make an honest user recieve a message from another honest user) with a message identifier e that is not associated with the inputted mid or correct platform data pd.

G0: This is identical to the standard $PCONF^0$ game for $b = 0$.

G1: We add the additional condition to goodRec to abort immediately if the inputted e value has never been outputted by the sender U_s as part of a message to the receiver U_r at any point in the game. This is indistinguishable from G0 because finding a valid e that was not sent from U_s to U_r contradicts the authenticated encryption properties of the messaging scheme, and can therefore happen with only negligible probability.

We note that otherwise, e must have resulted from a call to goodSend (oracle function to make an honest user send a message to another honest user) between U_s and U_r earlier in the game, and so there exists some mid'' in T_{send} that stores e. The only difference between calling goodRec with the correct pair (mid'', e) instead of (mid, e) is that the tid will always be $\perp$, which only weakens the adversary's power because it can no longer see the message contents by having the message reported, so we can assume that $e = e'$ in all other cases.

G2: Consider the contents of the platform data (C_A, src, C'_F) that the platform passes to the receiver. In the correct protocol, the platform would pass over identical C_A and C'_F values to the ones that were presented by the sender (The src ciphertext is chosen by the platform and is allowed to be unique for every new message, so it doesn't matter what this is chosen to be). We modify G1 to immediately abort if C_A and C'_F are not identical to the commitments presented by the sender. This game is indistinguishable from Game 1, because the openings of C_A and C'_F are encrypted with the message plaintext in e, and so the receive function will always fail when the receiver checks the openings against the commitments that the platform provides.

We have ensured that the platform must act honestly and pass the correct information between protocols. We now show that it cannot act dishonestly *during* either of the interactive protocols getUser or RecMsg.

G3: We modify G2 to immediately abort if the platform gives the user an invalid MAC for the expected attributes in goodRec, malSend, or getUser.

By the blind issuance properties of the MAC (with no hidden attributes in the case of NewUser), the adversary can achieve this and still create a correct issuance proof with only negligible probability, and so this game is indistinguishable from Game 2.

We've now ensured that the platform interacts with the user according to the expected protocol.

G4: We modify Game 3 so that each e sent between honest users is just an encryption of a default value of all zeroes, and the oracle passes the actual information between the two users. Since the adversary has non of the secrete keys for these interactions, this game is indistinguishable from Game 3 by the authenticated encryption properties of $\mathcal{E}$.

G5: We modify G4 so that during calls to goodSend and goodRec on an unrevealable message, i.e. one with tid set to $\perp$, we replace the receiving proof output by redeem and the re-randomization proof π_r output by present with the outputs of the zero-knowledge simulator for the proofs. By the zero-knowledge properties of the proof system, this is indistinguishable from a real proof, and so the game is indistinguishable from G4.

G6: We alter each call to goodSend with $tid = \perp$ to use the authoring data for the currently sending user as the MAC that gets presented to the platform when the message is sent rather than the correct forwarding data for the message (if it is a forward). The commitments C_A and C'_F still commit to the same values that would have been used in the standard game. By the anonymity of the MAC, the platform cannot distinguish between a presentation of the authoring data and the forwarding data, because both are valid MACs. Therefore this game is indistinguishable from G5.

G7: We modify the ciphertexts (A_1, A_2), (B_1, B_2), (C_1, C_2) to just be encryptions of the default values $\perp, G^{\perp}, G^{\perp}$ when goodRec is called on a message with $tid = \perp$. By the blind-issuance property of the keyed-verification anonymous credentials scheme [5], this is indistinguishable to the platform from issuing a MAC on the expected values.

G8: Finally, we change C_A and C'_F in calls to goodSend with $tid = \perp$ so that they commit to the default message $\perp$ and default source $G^{\perp}, G^{\perp}$. By the hiding properties of the commitment, this is indistinguishable from commitments for the actual message and source values.

We now note that applying the same hybrids described in Games 1-8 starting from the $PCONF$ game where $b = 1$ instead of 0 are each indistinguishable by the same arguments, and result in a game identically distributed to Game 8, because all message interactions with $tid \neq \perp$ are identically distributed in both games by definition, and we have altered the $tid = \perp$ case to use the same authoring data, commit to the same attributes, and receive a MAC on the same values. Therefore we have shown that both $PCONF^1_{ST,\mathcal{E}}$ and $PCONF^0_{ST,\mathcal{E}}$ are indistinguishable from Game 8 to an efficient adversary, and so they are also indistinguishable from each other.

This means an efficient adversary can gain at most negligible advantage in the PCONF game, and so Scheme 2 satisfies platform confidentiality. □

C.2 Accountability – Proof of Theorem 5.2

Intuitively, because the receiver of a message has complete control over what commitments and attributes it receives a MAC on, our scheme satisfies perfect accountability.

Proof. As a reminder, the $srcBIND$ game challenges an adversary with the ability to send messages with content of their choice between any two users and completely control some set of malicious users to create a message and associated content that is successfully received by an honest user, but then cannot be reported.

The tree-unlinkable scheme's definition of Report provides five opportunities for failure. We list these in order of occurrence:

(1) The platform's call to blindVf fails to verify the validity of π_p, the proof that the reporter has a valid MAC for the message that they are reporting.
(2) The platform's call to Vf fails to verify the validity of π_r, the proof that the commitments provided by the reporter contain correct re-randomizations of the original credentials.
(3) The first equality check fails ($C'_f \neq (G^z_{y_3} G^d_d, G^z_{y_1} E_1, G^z_{y_2} E_2)$. In other words, the opening o_f is not a valid opening for the commitment C'_f.
(4) The second equality check fails ($d \neq d'$).
(5) The third and final equality check fails ($d = \perp$).

We'll show that assuming the platform and reporter honestly received the message being reported, each of these failures can happen with zero probability.

π_p fails to verify. Because the platform and user honestly followed the receipt protocol, a successful receipt guarantees that the reporter has a valid MAC on the attributes being reported. If a user was unable to create a valid presentation proof for this MAC, it would contradict the correctness of the MAC presentation of [5]. We conclude that this failure happens with zero probability.

π_r fails to verify. All of the relationships proved in π_r are constructed by the user during the present protocol. Because the user is honest and follows the protocol, they ensure that all of these relationships are valid, and so an incorrect proof would violate the completeness of the proof scheme, so this failure also happens with zero probability.

The opening to the commitment C'_f is invalid. By definition of the scheme, the reporting user must have received a valid opening (z_F, d, E_1, E_2) for C_F, meaning that

$$C_F = (C_d, C_{E_1}, C_{E_2}) = (G_{y3}^{z_F} G_d^d, G_{y_1}^{z_F} E_1, G_{y2}^{z_F} E_2).$$

When the honest reporter follows the present protocol, C'_F and o_F are computed as

$$\begin{aligned} C'_F &= (C_d G_{y3}^{z'}, C_{E_1} G_{y_1}^{z'} G^{rnd}, C_{E_2} G_{y_2}^{z'} Y^{rnd}) \\ &= (G_{y3}^{z_F} G_d^d G_{y3}^{z'}, G_{y_1}^{z_F} E_1 G_{y_1}^{z'} G^{rnd}, G_{y2}^{z_F} E_2 G_{y_2}^{z'} Y^{rnd}) \\ &= (G_{y3}^{z_F+z'} G_d^d, G_{y_1}^{z_F+z'} E_1 G^{rnd}, G_{y2}^{z_F+z'} E_2 Y^{rnd}) \end{aligned}$$

and $o_F = (z_F + z', d, (E_1 G^{rnd}, E_2 Y^{rnd}))$, so o_F is guaranteed to be the correct opening for C'_F, so this check fails with zero probability because the reporter honestly follows the protocol and successfully received the original commitment.

$d \neq d'$. This condition only occurs if the reporter presents a credential for a message other than the one being reported. This will never happen because the reporter is honest.

$d = \perp$. During a message receipt, the user checks that the message being received is not equal to $\perp$, and aborts otherwise, so this will never happen because the message being reported was successfully received.

In conclusion, we've shown that assuming a message was received successfully by an honest user, a report of that message can fail with zero probability, so the scheme satisfies perfect accountability. □

C.3 Unforgeability – Proof of Theorem 5.2

Proof. The unforgeability of our scheme will depend on the extractability of the proofs used. Note that the blind issuance properties and unforgeability of the anonymous credentials used in [5] assume that the system used for proofs of knowledge satisfies a strong extractability property, in particular, that we can extract the opening (z, d, E_1, E_2) of the commitments C_d, C_{E_1}, C_{E_2} used in the MAC presentation proof $\pi_{present}$, which can be used to re-compute the attributes of the MAC being presented. Since we use their scheme, whose security relies on the random oracle and generic group models, we inherit their use of these models in our scheme. Other possibilities for instantiating such a proof system are discussed in Appendix D of [4].

The proof is a bit long, so we will first lay out the high-level intuition. Suppose an adversary wins the game and presents a report to the platform on an (m, U, md) tuple that was never actually sent by the honest user U.

We have two possibilities. On one hand, the adversary could have achieved this by acting dishonestly during the last report interaction in order to report a message that it didn't actually have forwarding credentials for. On the other hand, the adversary may have correctly followed the standard report protocol, but was able to succeed because it had valid credentials that were created dishonestly earlier in the game.

In this second case, we can then look at the point in time when the adversary received these credentials earlier in the game. Once again, these could have been obtained by the adversary acting dishonestly to create credentials that it didn't actually have, or this receipt could have been performed honestly because the necessary credentials were created earlier in the game.

In this way, we can trace back through the adversary's interactions with the oracle to identify the first interaction that resulted in a set of forwarding credentials with an honest source user that didn't originate from a call to goodAuth.

We consider all the places where this could have happened, and then in the actual proof, remove these opportunities in a series of game hops:

(1) *While the adversary was sending a message.*
If the incorrect credentials are created when an adversary is sending a message (i.e., during a call to malSend), this could happen because either the adversary is able to prove ownership of a MAC that was never created by the platform (addressed in G1 in the proof below), fake a proof that the new commitments stored in C'_F are correct re-randomizations of the original attributes (G3), or find an alternate opening of the commitments it used to prove the validity of its forwarding credentials, C_F and C'_F (G4).
(2) *While the adversary was receiving a message.*
If the incorrect credentials are created when an adversary receives a message, this means that either the adversary found alternate openings to the commitments of the attributes it can receive credentials from, C_F and C_A (Games G4 and G5, respectively), it faked a proof that the attribute ciphertexts were a valid re-randomization of C_A or C_F (G2), or it received a valid MAC on different values by acting dishonestly during the blind issuance protocol for the MAC (G1).
(3) *While the adversary was reporting the message.*
A valid report follows the same approach as a send, so it has the same opportunities for dishonest action, which we cover above.
(4) *While the adversary was creating a new adversary-controlled user.*
The last potential place where incorrect credentials could have been created is during a call to getUser. However, the newUser function is non-interactive, so the adversary cannot act dishonestly in order to alter the credentials they receive.

In each game, we remove one of these opportunities for dishonest action and show that the resulting game is indistinguishable from the previous game to an efficient adversary. This means that in the final game, the adversary has no opportunity to create fake credentials, and so cannot win the game.

We present the games below.

G0: This game is identical to the standard *unFORGE* game.

G1: This game is identical to the *unFORGE* game, but we add a set $\mathcal{M}_{MAC}$, that keeps track of all attributes that the platform has issued a MAC on. To do this, we modify the receiving interaction so that when a proof π provided by the receiver is verified by the platform, we run the knowledge extractor to extract witnesses $(h, r_1, r_2, r_3, rnd, d_A, z_A, d_F, z_F)$ and abort if extraction fails. We use these witnesses to decrypt (A_1, A_2), (B_1, B_2), (C_1, C_2) to get attributes M, E_1, E_2, i.e. $A_2/A_1^h, B_2/B_1^h, C_2/C_1^h$, respectively. We then add (M, E_1, E_2) to $\mathcal{M}_{MAC}$. The extractor fails with negligible probability, so this game is indistinguishable from Game 0.

G2: We further modify so that every time a message is sent or reported, if the proof output by present verifies, we again run the proof of knowledge extractor on the MAC presentation proof $\pi_{present}$ to extract attributes M, E_1, and E_2 and abort if extraction fails. Once again, failure happens with negligible probability and this is indistinguishable from G1.

G3: After extracting the attributes M, E_1, E_2 as described in G2, we immediately abort if $(M, E_1, E_2) \notin \mathcal{M}_{MAC}$.
If the game aborts due to the above reason, it means that the user proved ownership of a MAC that the platform never created. (Note that we're not requiring that the attributes be in any way correct at this stage, we are just looking directly at the exact attributes of the MACs that the platform created and comparing them to the attributes that the adversary tries to prove it has. This can happen with negligible probability due to the unforgeability of the anonymous credentials of [5], and therefore G3 is indistinguishable from G2.

G4: In this next game, every time an individual presents a valid receiving proof to the platform, we run the extractor to get witnesses $h, r_1, r_2, r_3, rnd, d_A, z_A, d_F, z_F$. We immediately abort if the extractor fails. By the extractability of the proof system, a valid proof fails extraction with negligible probability and so this game is indistinguishable to the forgery adversary. On success, we are guaranteed that the properties included in the proof are guaranteed to hold, i.e. that $P_1 \wedge P_A$ or $P_1 \wedge P_F$ as described in Figure 5 is satisfied.
We also note that by this check, if $P_1 \wedge P_A$ holds, then we are guaranteed that:

$$\begin{aligned} A_1, A_2 &= G^{r_1}, H^{r_1} G_d^{d_A} \\ B_1, B_2 &= G^{r_2}, H^{r_2} G^{rnd} E_1^{(A)} \\ C_1, C_2 &= G^{r_3}, H^{r_3} Y^{rnd} E_2^{(A)} \\ C_A &= G_d^{d_A} G_{y_3}^{z_A} \\ C_d &= G_d^{d_F} G_{y_3}^{z_F} \end{aligned}$$

So, the encryptions are valid encryptions of d_A and a re-randomization of (E_1, E_2) under public key H.

Similarly, if $P_1 \wedge P_F$ holds, then we are guaranteed that

$$\begin{aligned} A_1, A_2 &= G^{r_1}, H^{r_1} G_d^{d_F} \\ B_1, B_2 &= G^{r_2}, H^{r_2} G^{rnd} C_{E_1} / G_{y_1}^{z_F} \\ &= H^{r_2} G^{rnd} G_{y_1}^{z_F} E_1 / G_{y_1}^{z_F} \\ &= H^{r_2} G^{rnd} E_1 \\ C_1, C_2 &= G^{r_3}, H^{r_3} Y^{rnd} C_{E_2} / G_{y_2}^{z_F} \\ &= H^{r_3} Y^{rnd} G_{y_2}^{z_F} E_2 / G_{y_2}^{z_F} \\ &= H^{r_3} Y^{rnd} E_2 \\ C_d &= G_d^{d_F} G_{y_3}^{z_F} \\ C_A &= G_d^{d_A} G_{y_3}^{z_A} \end{aligned}$$

So, this guarantees that in the forwarded case, the encrypted values must be an encryption of d_F and a re-randomization of (E_1, E_2), where (z_F, d_F, E_1, E_2) is a valid opening for the commitment C_F.

G5: In this game, we check the sending/reporting proofs in the interaction. When given valid $\pi_{present}$ and π_{rerand} proofs associated with commitments $C_F = (C_d, C_{E_1}, C_{E_2})$ and $C_F' = (C_d', C_{E_1}', C_{E_2}')$, we apply the proof extractor to extract the witnesses (z', rnd) from π_{rerand}, aborting on failure. By the strong extractability property, this extraction fails with only negligible probability and otherwise we are guaranteed that the statement in π_{rerand} holds true for the provided values of C_F and C_F'.
Moreover, this guarantees that if (z, d, E_1, E_2) is the opening of C_F extracted from the associated presentation proof $\pi_{present}$, then we must have

$$C_d' = C_d G_{y_3}^{z'} = G_{y_3}^{z+z'} G_d^d$$

$$C_{E_1}' = G_{y_1}^{z'} G^{rnd} C_{E_1} = G_{y_1}^{z+z'} G^{rnd} E_1$$

$$C_{E_2}' = G_{y_2}^{z'} Y^{rnd} C_{E_2} = G_{y_2}^{z+z'} Y^{rnd} E_2$$

and so $(z + z', d, E_1 G^{rnd}, E_2 Y^{rnd})$ must be a valid opening for C_F'.
By the requirements of Game 3, the sending user must have had a valid MAC on (d, E_1, E_2), and so this ensures that the opening C_F' must be a re-randomization of attributes that the user has a valid MAC for.

G6: We have shown that the attributes used to prove and create MACs must be internally consistent (within a send/receive operation), but we now show that they need to consistent across transactions too. Let G4 be identical to G3 except for the modification that every time we have a valid pair of sending proofs π_p and π_r, we extract the opening $(z + z', d, E_1 G^{rnd}, E_2 Y^{rnd})$ of C_F' from the sending proof In the same manner as G3. Then, during the corresponding receiving proof that uses the same C_F' and some arbitrary C_A, we once again extract the witnesses $(h, r_1, r_2, r_3, rnd, d_A, z_A, d_F, z_F)$, aborting on failure. By the extractability of the proof system, extraction fails with negligible probability, making this game indistinguishable from G5.

G7: We modify G6 so that after extracting the above witnesses, we use them to compute the new opening $(z_F, d_F, E_1' = C_{E_1}/G_{y_1}^{z_F}, E_2' = C_{E_2}/G_{y_2}^{z_F}$.
We abort if these openings are not the same, i.e. if $(z + z', d, E_1 G^{rnd}, E_2 Y^{rnd}) \neq (z_F, d_F, E_1', E_2')$.
We first note that E_1' and E_2' are completely determined by the value of z_F, so it suffices to show that we must have $(z+z', d) = (z_F, d_F)$. If this is not the case, this means that the adversary was able to find distinct pairs (z_1, d_1) and (z_2, d_2) such that $G_{y_3}^{z_1} G_d^{d_1} = G_{y_3}^{z_2} G_d^{d_2}$ however, this implies that the adversary knows the discrete log of G_{y_3} with respect to G_d or vice versa, both of which we assume are unknown, so we conclude that finding two such pairs will happen with only negligible probability, and so the game is indistinguishable from G6.

G8: We just showed that the openings of the forwarding commitment created by the sender can't be changed by the receiver. We will now do the same for authoring commitments. We note that for authoring commitments, the source encryption is chosen by the platform and verified in the clear, so we need only ensure that the message stored in C_A created by an honest sender stays constant.
We modify G7 by adding the requirement that each time goodAuth or goodFwd is called, we save the message m (which could be $\perp$) that is associated with the authoring commitment C_A that gets presented to the platform. Then, if the receiving proof verifies, we extract witnesses z_A and d_A from the proof, aborting if the extraction fails. This happens with negligible probability, so this game is indistinguishable from G7.

G9: After extracting witnesses z_A and d_A in G8, we abort if $H(m) \neq d_A$.
As before, if this is the case, then after being provided a valid opening (z, d) by the honest sender, the adversary found some new (z_A, d_A) such that $G_{y_3}^{z} G_d^{d} = G_{y_3}^{z_A} G_d^{d_A}$. By the same reasoning as the previous game, this can be achieved only with negligible probability, and therefore this new game is indistinguishable from G4 to an efficient adversary.

By the definition of G9, we know that if a message is successfully sent or reported with attributes d, E_1, E_2, then either the adversary must have previously received a message with attributes $d, E_1 G^r, E_2 Y^r$ for some random r, or in the case of sending, (E_1, E_2) must decrypt to the sender's identity. (If the attributes came from an unused forwarding or authoring commitment, we would have $d = \perp$ and the protocol would automatically fail).

Because the decryption $E_2 Y^r/(E_1 G^r)^y$ will be constant for any value of r, we can trace back through these extracted receipts to the first time a MAC on attributes (d, E_1, E_2) was created. This must either have been the result of a call to goodAuth or malSend. If it was from malSend, then the sender wasn't an honest user, and so such attributes cannot be used to win the forgeability game.

Otherwise, if it was in goodAuth, then the tuple must have been added to $\mathcal{M}_{sent}$, and so similarly can't be used to win the game.

Therefore, we have shown that there is no way for the adversary to win Game 9. Because we showed that Game 9 was indistinguishable from the *unFORGE* game. We can conclude that any efficient adversary will have negligible advantage against *unFORGE*, and therefore Scheme 2 is unforgeable. □

C.4 Deniability – Proof of Theorem 5.4

Proof. We present the forgery protocols in Figure 10.

Universal Deniability

As in the proof of the first scheme, we note that the forged and unforged outputs have identical distributions conditioned on the values of each *src*, the source user's authoring data *ad*, and the message scheme ciphertexts *e*.

G0: This game is identical to the standard $UnivDen^0$ game when $b = 0$, and the output is unforged.

G1: We modify G0 so that each *src* value created by the platform is an encryption of U_D and the relevant metadata, rather than the true unforged user (This includes the authoring data *ad* used to send the first message). These ciphertexts are never decrypted by the adversary, who doesn't have access to the platform's secret keys, and so this game is indistinguishable from G1 due to the CPA-security of ElGamal encryption [31].

G2: We modify each e_i to be the result of $\mathsf{Sim}_{\mathcal{E}}(U_i, U_{i+1}, \cdot)$ instead of using the actual messaging scheme. This is indistinguishable from G1 by the deniability of the messaging scheme.

G3: We change G2 to use the authoring data *ad* of the forging user U_D instead of the authoring data of U_0 to author the first message on the path. Due to the changes made in G1, both *ad*s have the form $(E_1, E_2), \sigma$, where (E_1, E_2) is an ElGamal encryption of U_D, and σ is a MAC on $\perp$. The ciphertext portion of the authoring data appears in re-randomized form in the commitment C_F', contained in the platform data. However, because both are re-randomized before appearing in the transcript, they have the exact same distribution, and so this game is identical to G2.

We now observe that because we have switched the *src* values, authoring data, and message scheme ciphertexts, the output of G3 is identically distributed to the forged version of the game, $UnivDen^1$. We conclude that the two games are indistinguishable to any efficient adversary, and so Scheme 2 satisfies universal deniability.

□

Platform Compromise Deniability We note that the only differences between the forged and unforged versions of the *PlatDen* game is that a fresh version of the authoring data on the same values is used to send the first message, and the *e*s are created using the simulator $\mathsf{Sim}_{\mathcal{E}}$ in the forged version, but are created using the keys of users along the path in the real version.

By the same arguments as for the universal deniability case, changing $UnivDen^0$ by swapping out the true authoring data *ad* of U_0 for a fresh version of the authoring data results in an identical game because the source ciphertexts are re-randomized before appearing in the transcript, and by the deniability of the underlying messaging scheme, replacing the *e*s resulting from calls to send with simulated values output from $\mathsf{Sim}_{\mathcal{E}}$ results in a game that is indistinguishable to an efficient adversary, and also identical to the forged version of the game, $UnivDen^1$. We conclude that any efficient adversary can achieve at most a negligible advantage

```
$\mathsf{UForge}(path, path, m, mds, \mathsf{pk})$
$(U_0, ..., U_k) \leftarrow path$
$ad \leftarrow T_{auth}[U_D]$
$\mathsf{Send}(U_D, mds[0])$ :
    $(e, Tr_s) \xleftrightarrow{\langle U_{auth}(m, ad), O_s \rangle}$
    $(m, fd_0, Tr_r) \xleftrightarrow{\langle U_{rec}, O_s \rangle}$
$output.append(Tr_r, fd_0)$
**for** $j = 1, ..., k-1$ :
    $\mathsf{Send}(U_D, mds[j])$ :
        $(e, Tr_s) \xleftrightarrow{\langle U_{fwd}(m, fd_{j-1}), O_s \rangle}$
        $(m, fd_j, Tr_r) \xleftrightarrow{\langle U_{rec}, O_s \rangle}$
    $m' \leftarrow \mathsf{receive}(U_D, U_D, e)$
    $e' \leftarrow \mathsf{Sim}_{\mathcal{E}}(U_j, U_{j+1}, m')$
    $(e, info) \leftarrow Tr_s$
    $Tr_s \leftarrow (e', info)$
    $output.append(Tr_s, fd_{j-1}, Tr_r, fd_j)$
$\mathsf{Report}(m)$ :
    $Tr_{rep} \xleftrightarrow{\langle U_{rep}(fd_{k-1}), O_s \rangle}$
$output.append(Tr_{rep}, (U_0, mds[0]))$
**return** $output$
```

```
$\mathsf{PForge}(path, m, mds, \mathsf{pk} = Y, \mathsf{sk})$
$(U_0, ..., U_k) \leftarrow path$
$r \xleftarrow{R} \mathbb{Z}_q, (E_1, E_2) \leftarrow (G^r, U_0 Y^r)$
$(\sigma, \pi_{issue}) \leftarrow \mathsf{issue}(\perp, E_1, E_2, \mathsf{sk}_{MAC})$
$ad \leftarrow (\sigma, (E_1, E_2))$
$(e, Tr_s, pd) \leftarrow \langle U_{auth}(m, ad), \mathsf{Proc}(U_0, mds[0]) \rangle (U_D, U_D, \mathsf{pk})$
$(Tr_r, fd_0) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_D, U_D, e, \mathsf{pk})$
$output.append(Tr_r, fd)$
**for** $j = 1, ..., k-1$ :
    $(e, pd, Tr_s) \leftarrow \langle U_{fwd}(m, fd_{j-1}), \mathsf{Proc}(U_j, mds[j]) \rangle (U_D, U_D, \mathsf{pk})$
    $(Tr_r, fd_j) \leftarrow \langle U_{rec}, P_{rec}(\mathsf{sk}, pd) \rangle (U_D, U_D, e, \mathsf{pk})$
    $(e, info) \leftarrow Tr_s, m' \leftarrow \mathsf{receive}(U_D, U_D, e)$
    $Tr_s \leftarrow (\mathsf{Sim}_{\mathcal{E}}(U_j, U_{j+1}, m'), info)$
    $output.append(Tr_s, fd_{j-1}, Tr_r, fd_j)$
$\mathsf{Report}(m)$ :
    $Tr_{rep} \xleftrightarrow{\langle U_{rep}(fd_{k-1}), O_s \rangle}$
$output.append(Tr_{rep}, (U_0, mds[0]))$
**return** $output$

$\mathsf{Proc}(U_s, md, \mathsf{pk} = Y)$
$\xleftarrow{e, C_A, C_F, C'_F, \pi_p, \pi_r}$
$r \xleftarrow{R} \mathbb{Z}_q, S \leftarrow (U_s, md)$
$src \leftarrow (G^r, SY^r)$
**return** $((C_A, src, C'_F), e)$
```

Figure 10: Forgery Algorithms for Scheme 2.

against the *UnivDen* game, and so Scheme 2 satisfies platform-compromise deniability.

We have therefore shown that the scheme satisfies universal and platform-compromise deniability, and is therefore deniable.