

Transcript Franking for Encrypted Messaging

Armin Namavari and Thomas Ristenpart

Cornell Tech

Abstract. Message franking is an indispensable abuse mitigation tool for end-to-end encrypted (E2EE) messaging platforms. With it, users who receive harmful content can securely report that content to platform moderators. However, while real-world deployments of reporting require the disclosure of multiple messages, existing treatments of message franking only consider the report of a single message. As a result, there is a gap between the security goals achieved by constructions and those needed in practice.

Our work introduces *transcript franking*, a new type of protocol that allows reporting subsets of conversations such that moderators can cryptographically verify message causality and contents. We define syntax, semantics, and security for transcript franking in two-party and group messaging. We then present efficient constructions for transcript franking and prove their security. Looking toward deployment considerations, we provide detailed discussion of how real-world messaging systems can incorporate our protocols.

1 Introduction

End-to-end encrypted (E2EE) messaging is used by billions of people through platforms like Whatsapp, Signal, and iMessage [37]. As a result, users enjoy strong security and privacy protections even in the face of messaging platform compromise by malicious insiders, remote attackers, or government overreach. Abuse, hate, and harassment, however, are not prevented or mitigated by encryption, and encrypted messaging platforms are used to spread misinformation, incitements of violence, and illegal content [22]. As a result, a rapidly growing body of work has sought to provide trust and safety features for encrypted messaging, without diminishing its privacy benefits [31].

One important line of work targets secure reporting of abusive messages (see [31]). When users receive harmful content, they can report it to the platform, which can in turn take appropriate action against the user that sent the problematic content. Such user-driven reporting features are widespread on plaintext platforms and play an instrumental role in content moderation across a wide range of abuse types [28]. In encrypted messaging, however, the platform cannot trivially verify that a report corresponds to a transmitted message.

Facebook’s message franking feature [1] was the first to target cryptographically verifiable abuse reports. Message franking targets not compromising the confidentiality of unreported messages, and preventing attacks that undermine

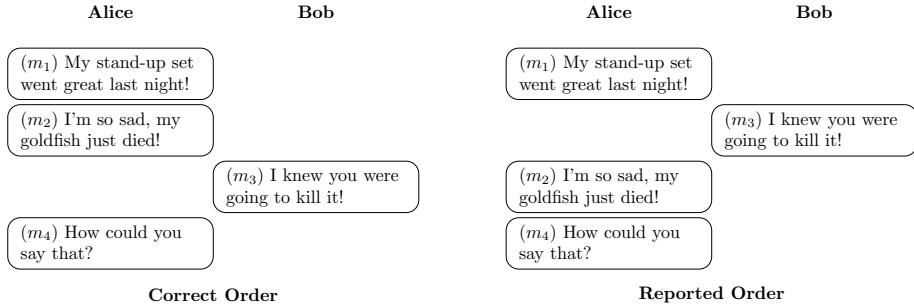


Fig. 1. An example conversation in which message ordering impacts interpretation. A report of this conversation should confirm for moderators the causal ordering.

the trustworthiness of reporting: users should not be able to report messages that were not sent, nor be able to send messages that cannot be reported. These security properties were first formalized in [15] as receiver binding and sender binding, respectively. While Facebook’s first design had a sender binding vulnerability [12], we now have message franking protocols with strong assurance in their security [12, 15, 16]. Subsequent work extended to provide asymmetric message franking schemes (AMFs) [17, 33] for two-party sender-anonymous messaging, group AMFs [18], franking for two-party channels [16], and message franking that allows only revealing parts of messages [21].

All those treatments of message franking only support reporting individual messages. In practice, however, moderators typically need visibility into more of a conversation to make judgements, and indeed existing abuse reporting workflows do report surrounding messages when one is reported [3, 15]. Recent work [36] reports that users would find it useful to have more agency in specifying what portions of their private conversations are disclosed to moderators, which is not something current approaches offer.

Despite this, to date, there has been no attempt to show how to provide cryptographically verifiable reporting of multi-message conversations. Near-at-hand approaches, including those used in practice, do not provide a satisfying level of security. Consider reporting with message franking: each individual message can be verified along with a platform timestamp of when it was sent. But a malicious client can simply undetectably omit messages from a report. For example consider the conversation between Alice and Bob shown in Figure 1. If Alice reports the conversation with omission of m_1 it blocks moderators from interpreting Bob’s message m_3 as replying to m_1 rather than mocking Alice’s loss. A more subtle issue is that even high-precision timestamps do not establish strict causal ordering [19]. Let $m < m'$ represent that m was received before m' was sent. Returning to the figure, it could be that $m_2 < m_3$, or it could be that m_3 was sent before Bob received m_2 . The result in the latter case would be Bob’s view of the conversation being different from Alice’s.

The problem of conversation ordering and moderation has been known to practitioners since at least 2014 [24], but only recently has there been a first effort to address it by Chen and Fischlin [11]. They propose a message franking protocol, MFCh_{cFB} , in which clients report observed message ordering via franking metadata alongside content. While their approach provides a novel augmentation of message franking with causality, they stop short of providing a fleshed out solution for multi-message franking. Their security modeling considers reporting individual received messages. Meanwhile, allowing reporters to disclose sent messages is crucial to multi-message reporting with full context. One could consider a natural extension of MFCh_{cFB} in which the report function is invoked for each message and both parties are involved in disclosing each other’s messages. This, however, is susceptible to denial of service as the abusive party can simply refuse to cooperate and go offline, withholding crucial context.

Moreover, MFCh_{cFB} is susceptible to integrity attacks. Since causality data is client-generated, malicious clients can provide incorrect information about message ordering. Consider the example conversation in Figure 1. We demonstrate an attack in which Bob makes Alice observe the view on the left while making it so that Alice can only report the view on the right. Bob simply sends m_3 after having received m_2 but attaches causality metadata to m_3 indicating he has seen only m_1 . Hence, Bob can force Alice to observe an upsetting message ordering while making seem as if it were due to the network reordering messages.

Our contributions. We suggest a new approach that we call *transcript franking*. This cryptographic protocol goal allows users to report some or all of two-party or group conversations with stronger security guarantees about message ordering and omissions. We define the syntax and semantics of a transcript franking scheme and provide formal security definitions. We go on to detail transcript franking schemes for both the two-party and group messaging settings; our schemes are practical to deploy and avoid issues like those above, clarifying in a single report all relevant information about the messages reported, including causal ordering. To do so, our schemes deviate from Chen-Fischlin’s approach of using client-provided causality metadata, instead taking advantage of the fact that the platform can establish causal ordering over ciphertexts and check that reports are consistent with it. We prove that our new schemes meet our new security goals under standard assumptions.

We treat both two-party (direct) and group messaging settings; we explain further each, starting with the former.

Two-party transcript franking. We start with the two-party case, where only two clients are involved in a conversation (sometimes called direct messaging). Our goal is to enable either participant in the encrypted conversation to report all or part of the conversation to the platform. Our starting point is symmetric message franking (SMF) [15, 16], in which the clients encrypt messages using committing authenticated encryption with associated data (AEAD) and the platform MACs (portions of) the resulting ciphertexts to produce a reporting tag. To report a message, the ciphertext, secret encryption key, and the reporting tag are all sent to the provider.

In this setting where we rely on fast symmetric primitives, the veracity of a ciphertext can only be checked by those who have access to the secret keys—before reporting, just the two clients. Prior work on SMFs took this to mean that one cannot support reporting a sent message, since the reporter could be unreliable. But if we want to allow reporting more of a conversation, we must support this. Intuitively, our approach will be to leverage acknowledgements of received ciphertexts to register their validity for their use in reporting.

To do so, we first enrich the architectural model to surface how the provider manages sending and receiving events separately. This more accurately captures the asynchronous nature of messaging. Formally, the platform is represented by a pair of algorithms **TagSend** and **TagRecv**. We allow the platform to maintain per-conversation state; we also give a way to outsource this state securely to clients. A client sends a message by running an algorithm **Snd** and submitting the resulting ciphertext to the platform by calling **TagSend**. To receive a message, a recipient client runs an algorithm **Rcv** on it, and if it accepts the message, indicates so to the server, which then calls **TagRecv**. The latter readies a cryptographic reception acknowledgement for both the sender and the receiver.

A set of messages can be reported by submitting their ciphertexts, their corresponding secret keys, and platform-provided reporting tags. The moderator can process them via an algorithm **Judge** that verifies them, and produces a causal graph providing the moderator with: (1) a partial ordering over all messages that implies a total ordering over messages sent in either direction; and (2) indication of gaps—unreported messages sent between reported messages.

We require that transcript franking schemes provide confidentiality and integrity of messages that aren’t reported in the face of a malicious provider. More complex is the security of reporting, in which the platform is trusted, but clients are not. Here we formalize two security goals for transcript franking schemes, which can be viewed as strengthening SMF sender and receiver binding. Coming up with definitions that capture transcripts, rather than individual messages, proved challenging. For example, any given conversation can now give rise to all sorts of possible reports: the entire conversation or any subset of that conversation, including possibly just a single message.

Our first definition is *transcript reportability*. Here the security game tasks an adversary controlling one client with interacting with the provider and some other honest client. The adversary attempts to generate a message transcript such that the honest client cannot successfully report some adversarially-chosen subset of messages. In the case where only a single message is reported, this coincides with sender binding, but it goes much beyond since it requires that any subset of the conversation be reportable, including messages sent by the reporter.

The second main security notion is *transcript integrity*. Intuitively we want to not only ensure that no maliciously generated report can frame someone as having sent something when they have not, but, moreover, ensure that the reported transcript does not lie about ordering or omissions. Perhaps counterintuitively, we increase the adversarial power over prior treatments of receiver binding by

allowing the adversary to control *both* clients in a conversation to arbitrarily deviate from the protocol. The adversary has oracles to send ciphertexts to the platform and register having received them. The game keeps track of a ground truth graph of message transmission. This matches the view of the platform in terms of sending and receiving event orderings for each transmitted ciphertext. Then, the adversary’s goal is to come up with either a report that generates a causality graph inconsistent with the ground truth graph, or two reports that are inconsistent with each other. We define consistency relative to our causality graph abstraction.

The quad-counter construction. We now turn to constructions. We want to ensure practicality, using fast cryptographic mechanisms and avoiding unrealistic storage constraints. The main underlying idea is to have the platform carefully attest to the observed causal order of ciphertexts. Since we allow stateful platforms, we could of course just store a log of all sending and reception events that occurred, with their corresponding ciphertexts. But this would be prohibitively expensive in terms of storage. Instead, we can use MAC’d counters of events. Namely, for each party we have a pair of counters, a sending counter incremented each time that party sends a ciphertext, and a reception counter incremented each time a ciphertext is registered as received. The platform generates a cryptographic acknowledgement for each send and receive event: a tuple including the party identifiers, the type of event (send or receive), a binding commitment to the ciphertext, and the current counters, together with a MAC over the tuple using a platform-held secret key. Cryptographic acknowledgements are shared with both parties. The platform need only retain the four counters, hence the name of this protocol as the quad-counter construction (QCC).

A reporter can choose any subset of messages, and submits the identity of the sender, the message itself, the cryptographic commitment to the message, and both the sending and reception cryptographic acknowledgements. The platform can then verify each tuple, and construct a causality graph that indicates the precise causality order of the reported messages plus how many send and receive events were omitted from the report.

We show that, under standard assumptions on the MAC and commitment scheme, we achieve transcript integrity.

Extensions. The quad-counter construction readily extends to work in group messaging settings, by having a sending and reception counter for each party (for a total of $2n$ counters where n is the number of participants). The cryptographic acknowledgements are otherwise similar to the two-party case. We formalize this setting, showing how our definitions and results handle it securely.

One potential downside of the quad-counter construction and its generalization, as compared to in-use (but insecure) approaches like timestamps, is that the platform must now keep per-conversation state. While this state is tiny, it would be better to avoid, due to it complicating large-scale deployments where one would prefer to have platform servers stateless and able to load balance any message event across any server without having to synchronize state. We describe an extension to our approach which outsources the state to the clients,

at the cost of slightly extra data being sent to the platform, and relying on honest users to report replay of cryptographic acknowledgements. See Section 7 for details.

2 Preliminaries

2.1 General Notation and Primitives

Let \mathbb{Z}^* denote the non-negative integers $\{0, 1, \dots\}$. To indicate the range of elements $\{0, 1, \dots, N-1\}$, we use the shorthand $[N]$. The symbol $\lambda \in \mathbb{Z}^*$ denotes the security parameter. For an adversary \mathcal{A} in a game \mathbf{G} , we use $\Pr[\mathbf{G}(\mathcal{A}) \Rightarrow x]$ to denote the probability that the \mathbf{G} outputs x when run with adversary \mathcal{A} . With a tuple, we refer to its elements via 0-indexed positions or names. For instance, $c.x$ or $c[0]$ refer to the first element of the tuple $c = (x, y, z)$. Multiple indexing is also allowed: $c.(x, z)$ extracts the first and third elements of the tuple, while $c[1 : 2]$ extracts the last two elements (for ranges, both the start and end indices are inclusive). We indicate structs as sets of variables, e.g., $s \leftarrow \{x, y, z\}$. When s is in scope, we allow accessing x directly in order to simplify notation. The notation $d \leftarrow D(x)$ indicates obtaining the output d from a deterministic algorithm D , when fed input x . For a randomized algorithm R , we write $r \leftarrow^s R(x)$ to denote obtaining the output r from the input x . We utilize the terms *algorithm*, *routine*, and *procedure* interchangeably.

Bidirectional channels. To model two-party communication, we make use of a bidirectional channel abstraction, borrowing syntax from [11]. A bidirectional channel Ch consists of three procedures $\text{Ch} = (\text{Init}, \text{Snd}, \text{Rcv})$, defined over a key space \mathcal{K} , a message space \mathcal{M} , a party space $\{0, 1\}$, a ciphertext space \mathcal{C} , and a state space \mathcal{S} . The variable P and the labels $\{0, 1\}$ are used to indicate the two parties. Let the notation \bar{P} be shorthand for $1 - P$. We elaborate on these procedures below.

- $st \leftarrow \text{Init}(P, k)$ generates initial channel state $st \in \mathcal{S}$ for party $P \in \{0, 1\}$ with key $k \in \mathcal{K}$.
- $st', c \leftarrow^s \text{Snd}(P, st, m)$ produces a ciphertext $c \in \mathcal{C}$ from party $P \in \{0, 1\}$ for plaintext $m \in \mathcal{M}$ and client state st , yielding updated state st' .
- $st', m, i \leftarrow \text{Rcv}(P, st, c)$ decrypts a ciphertext $c \in \mathcal{C}$, received by the party $P \in \{0, 1\}$, to plaintext $m \in \mathcal{M}$ along with a sending index $i \in \mathbb{Z}^*$.

Correctness of a channel requires that all honestly sent messages can be successfully received with the correct sending index. For security, we expect standard confidentiality and integrity properties. See [11] for more details.

Message authentication code. A message authentication code (MAC) consists of the algorithms $\text{MAC} = (\text{KGen}, \text{Tag}, \text{Verify})$ defined over a key space \mathcal{K} , a message space \mathcal{M} , and a tag space \mathcal{T} . The key generation procedure KGen outputs a random key $k \in \mathcal{K}$. The procedure $\text{Tag}(k, m)$ outputs a tag $t \in \mathcal{T}$ for a message $m \in \mathcal{M}$. To verify a tag t on a message m , one runs the procedure $\text{Verify}(k, m, t)$, which outputs $b \in \{0, 1\}$. An output of 1 indicates a valid tag

on the message while an output of 0 indicates an invalid tag. For a MAC to be considered secure, it has to satisfy existential unforgeability under a chosen message attack (EUF-CMA). This means that, when given oracle access to $\text{Tag}(k, \cdot)$ and $\text{Verify}(k, \cdot, \cdot)$, for $k \leftarrow \text{KGen}()$, an adversary \mathcal{A} has a negligible probability of producing (m, t) , where m is not previously queried to Tag , that passes the verification check. We denote this probability as the advantage $\text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{A})$.

Commitment scheme. A commitment scheme CS consists of a pair of algorithms $(\text{Com}, \text{VerC})$ defined over a message space \mathcal{M} , a key space \mathcal{K} , and a commitment space \mathcal{Q} . The randomized algorithm $\text{Com}(m)$ outputs a pair $(k, c) \in \mathcal{K} \times \mathcal{Q}$, where c is the commitment and k is a key that allows opening the commitment to the original message $m \in \mathcal{M}$. In terms of security, a commitment scheme is often required to be *hiding* and *binding*. To satisfy the hiding property, the commitment c must reveal nothing about the message m . We formalize this via a real-or-random notion that requires that no efficient adversary can distinguish a commitment from a random bit-string of the same length. The binding property requires that an adversary \mathcal{A} has a negligible probability in producing a tuple (m, k, m', k', c) , where $m \neq m'$, $\text{VerC}(m, k, c) = 1$, and $\text{VerC}(m', k', c) = 1$.

Message franking. User-driven reporting for end-to-end encrypted messaging is captured by message franking [1, 12, 15]. For now, we focus our attention on two-party conversations between users. For the sake of notational simplicity, we elide associated metadata that clients or the server may associate with the message (e.g., timestamps). In accordance with our usage of messaging channels, we draw on message franking channels [16]. We opt for the syntax used in Chen and Fischlin’s work [11]. The procedure $\text{Rcv}(P, st, c)$ outputs st', m, k_f, i , where k_f is a key opening the commitment $c.c_f$, which is stored as part of the ciphertext c . There is also a server tagging procedure $\text{SrvTag}(st_S, P, c_f)$, which outputs a tag t on a franking commitment c_f , which corresponds to a ciphertext sent by party P . A reporting procedure $\text{Rprt}(st_S, P, m, k_f, c_f, t)$ verifies for the server that t is a server tag on c_f , which opens to a message m .

2.2 Causality Graphs

In order to model message transmission in a way that accounts for asynchronous networks, we use causality graphs. We define our causality graph object here, which draws heavily on the graph formalism used in [11]. A causality graph G modeling two-party messaging is a directed bipartite graph consisting of two disjoint sets of vertices V_0 and V_1 and an edge set E . An edge is a pair of vertices (v_1, v_2) where $v_1 \in V_P$ and $v_2 \in V_{\bar{P}}$ for some $P \in \{0, 1\}$. We write $V = V_0 \cup V_1$ to refer to the full set of vertices within the graph. Each vertex $v \in V$ contains four pieces of data: an action type $t \in \{S, R\}$, a sending counter cs , a reception counter cr , and a message $m \in \mathcal{M}$. Indeed, these four pieces of data are sufficient to uniquely identify a vertex within a single conversation. We can therefore define a vertex space \mathcal{V} as the direct product $\{S, R\} \times \mathbb{Z}^* \times \mathbb{Z}^* \times \mathcal{M}$, and the edge space \mathcal{E} as $\mathcal{V} \times \mathcal{V}$. The action type t indicates whether the vertex

corresponds to a sending (S) event or a reception (R) event. For any message sent from P to \bar{P} , there is an edge from a sending vertex in P to a receiving vertex in \bar{P} . We define the sub-graph relation as follows: for two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, we write $G_1 \subseteq G_2$ if $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$. We will also allow causality graphs that do not contain messages, which will become useful for our security definitions. Let G be a causality graph; we denote the message-excluded version of G as $\tilde{G} = (\{v[0 : 2] \mid v \in G.V\}, \{(v_1[0 : 2], v_2[0 : 2]) \mid (v_1, v_2) \in G.E\})$.

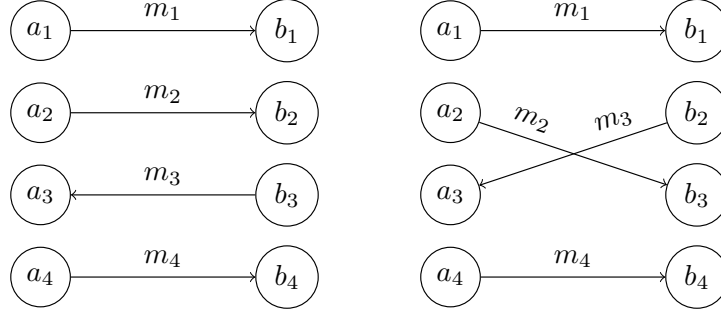


Fig. 2. Examples of causality graphs. Let the a nodes correspond to Alice and the b nodes correspond to Bob. The left graph corresponds to a situation in which both Alice and Bob view the left ordering from Figure 1. The right graph leads to Alice viewing the left ordering and Bob viewing the right ordering from the same figure.

The counters cs and cr are monotonically increasing counters over sending and reception event respectively for each party $P \in \{0, 1\}$. The addition of a sending event to V_P increments the sending counter for V_P while the addition of a reception event increments the reception counter. Consider a sending vertex $v \in V_P$. We have that $v.t = S$, there are $v.cs - 1$ sending events that precede v and $v.cr$ reception events that precede v . The nodes in V_P are totally ordered by the lexicographic ordering over v given by $(v.cs, v.cr)$. For a message m' sent from P to \bar{P} , there are vertices $v_s \in V_P$ and $v_r \in V_{\bar{P}}$, where $v_s.t = S$, $v_r.t = R$, $v_s.m = v_r.m = m'$, and $(v_s, v_r) \in E$.

Now that we have introduced the semantics of the data contained within the graph, we define graph operations associated with sending and receiving messages, and how they modify the data contained within the graph.

Graph initialization. An empty graph G consists of empty vertex sets V_0 and V_1 along with an empty edge set E . Each vertex set V_P has an associated send counter cs and reception counter cr . We use the notation $V_P.cs$ and $V_P.cr$ to refer to these counters for party P . Initially, $V_P.cs = V_P.cr = 0$ for $P \in \{0, 1\}$. The counters will be treated as auxiliary state as opposed to part of the graph. We use the symbol ε to denote the empty graph.

Addition of send operation. We use the notation $G \leftarrow G + (S, P, m)$ to denote the addition of a sending operation for message m' from party P . First,

we increment $V_P.cs$ ($V_P.cs \leftarrow V_P.cs + 1$), then create a new vertex v with $v.t = S$, $v.cs = V_P.cs$, $v.cr = V_P.cr$, and $v.m = m'$. Finally, we add v to V_P . When updating a non-message-inclusive graph, we simply write $G \leftarrow G + (S, P)$.

Addition of reception operation. The addition of a reception for the message with sending index i from party \bar{P} is denoted by $G \leftarrow G + (R, P, i)$. Note that this operation enforces that such a message with sending index i exists in $G.V_{\bar{P}}$. First, we increment $V_P.cr$, then create a new vertex v with $v.t = R$, $v.cs = V_P.cs$, $v.cr = V_P.cr$, $v.m = v_s.m$, where $v_s \in V_{\bar{P}}$, $v_s.t = S$, $v_s.cs = i$ (by construction, there is only one such vertex). We add v to V_P and add (v_s, v) to E .

Graph validity and consistency. A causality graph G is valid if there exists a sequence of send and reception operations that lead to its construction from an empty graph. A sub-graph G' is valid if there exists a valid graph G such that $G' \subseteq G$. Two graphs G and G' are consistent if there exists a valid graph G^* such that $G \subseteq G^*$ and $G' \subseteq G^*$. We write $G \approx G'$ to indicate that G and G' are consistent, and we use $G \not\approx G'$ to indicate that they are inconsistent. Notions of validity and consistency will be important in our security definitions.

Partial ordering over events. As we mentioned before, there is a total ordering over the events within V_0 and V_1 . For $v_1, v_2 \in V_P$, we have that $v_1 < v_2$ if $v_1.cs \leq v_2.cs$, $v_1.cr \leq v_2.cr$, and $(v_1.cs, v_1.cr) \neq (v_2.cs, v_2.cr)$. The causality partial ordering $v_1 \leq v_2$ for $v_1, v_2 \in V$ is defined as the transitive closure of these individual total orders along with the edge relation (i.e., $v_1' \leq v_2'$ if $(v_1', v_2') \in E$). Observe that this is an ordering over the sending and reception of messages as opposed to an ordering of the messages themselves.

Contiguity and gaps. When a moderator views a sub-graph of a full conversation, it will be useful to understand which events are contiguous (in the sense that no other events can be ordered between them) and which events have gaps. This makes clear where there might be missing context, which can later be provided by either party if requested. The inclusion of sending and reception counters within the graph provides rich information that allows interpretation of the contiguity of events. Consider two vertices $v, v' \in V_P$ where $v < v'$. If we have that $\max(v'.cs - v.cs, v'.cr - v.cr) = 1$, then by construction there can be no $v'' \in V_P$ such that $v < v'' < v'$, so v and v' are contiguous with one another, among the vertices in V_P . Taking into consideration the edges in E , we can ascertain whether any nodes in $V_{\bar{P}}$ can be ordered between them. If we have that $(v'.cs - v.cs, v'.cr - v.cr) = (a, b)$ for $v, v' \in V_P$, we know that there are a send events and b reception events that occurred after v , including v' . In this way, these counters provide rich interpretability of gaps within sub-graphs.

3 Two-Party Transcript Franking

Prior work on message franking has focused on reporting single messages that were received by the reporting user. Often, single messages do not contain sufficient context for understanding online harassment. Hence, we aim to extend message franking to enable reporting sequences of messages. We propose a new

cryptographic primitive we call transcript franking that captures this goal. In addition to providing integrity over the contents of messages contained within a sequence, a transcript franking scheme must have cryptographic assurance over the ordering and contiguity of messages reported within a sequence. Due to the inherent asynchrony of messaging, honest users may observe differing but valid views of the message order. We aim for users and platform moderators to be able to see the view of the transcript from the perspective of each party as well the causal dependencies between messages.

Limitations of current approaches. The original Facebook white-paper that introduced message franking suggests including server timestamps within the data tagged by the server [1]. As prior work points out, this is insufficient for capturing causal dependencies between messages since it does not directly reflect client-side views and capture message concurrency [11]. Furthermore, timestamps do not assure integrity over the contiguity of reported messages within a conversation. Recent work suggests incorporating causality metadata into message franking channels, however since this metadata is client-generated, it can deviate from the actual ordering of sending and reception events that clients experience [11]. As we explain in the introduction, this leads to attacks in which a malicious party can force the other party to view an upsetting message sequence while only being able to report a plausibly benign one.

A further limitation is that all prior treatments of message franking allow reporters to disclose only messages they have received from the other party. From a motivational standpoint, this seems reasonable since the goal of reporting is to demonstrate that an abusive party, presumably the non-reporting party, sent harmful content. However, when reporting multiple messages within a conversation, a reporter may have to include their own sent messages to provide sufficient context. In Figure 1, for instance, Alice should be able to show her message that precedes the message Bob sends in order for the moderator to evaluate whether it is abusive. The obvious solution of a reporter disclosing their own sent messages is insufficient since the corresponding ciphertexts may be malformed for the other party, and existing treatments of message franking provide no way for recipients to indicate this to the service provider. An alternative would be to involve both parties in the reporting process. Yet this can lead to a denial of service attack in which the accused party refuses to participate. Even with an honest accused party, requiring both parties to be online is an unfavorable limitation. Hence, we must devise a reporting protocol that is not contingent upon the participation of the non-reporting party. Note that such a protocol may still allow the non-reporting party to continue to disclose more context if they decide to do so.

Our suggested platform model. In line with the model used by the messaging layer security (MLS) standard [5], we conceptualize the platform as providing a delivery service (DS) for message transmission and an authentication service (AS) for managing user identities. Additionally, we consider a platform-managed moderation service (MS) for accepting user reports and taking action against abusive user accounts. Clients issue a **Send** request with a payload containing

the message ciphertext to the DS. To receive messages, a client issues a **Recv** request, to which the server responds with outstanding message ciphertexts. In a real platform, clients may be identified via usernames or phone numbers. Our transcript franking abstraction, on the other hand, simply identifies parties within a conversation via the numeric labels $\{0, 1\}$ for direct messaging.

Our model assumes a client receives a notification that indicates when the DS has successfully received their message and another notification when the recipient client device has successfully received the message. These two events correspond to the first and second checkmarks shown in widely used messaging platforms such as Whatsapp and Signal [2, 4]. Messages may still be dropped or reordered, but we will concern ourselves with reporting only messages that were successfully received. We remark that this differs slightly from traditional platform models for message franking, in which the platform simply tags a message once it is sent and shares this tag with just the recipient. It turns out this model will be crucial to achieving our transcript franking security goals.

Transcript franking syntax and semantics. We draw heavily on [11] as inspiration for our syntax and will later provide a comprehensive comparison between their approach to incorporating causality into message franking and our notion of transcript franking. A transcript franking scheme is a tuple of algorithms $\text{TF} = (\text{SrvInit}, \text{Init}, \text{Snd}, \text{Rcv}, \text{TagSend}, \text{TagRecv}, \text{Judge})$, defined over a server state space \mathcal{S}_S , a client state space \mathcal{S}_C , a key space \mathcal{K} , a message space \mathcal{M} , a commitment space \mathcal{Q} , a franking key space \mathcal{K}_f , a message-sent tag space \mathcal{T}_S , and a reception tag space \mathcal{T}_R . We detail each algorithm below. Input and output variables names are unique, and we indicate the “type” (the set of possible values) and description of a variable only the first time it is introduced in order to reduce verbosity. In general, st will refer to client or server state the before the invocation of a routine, while st' refers to the updated state after the invocation.

- $st_S \leftarrow \text{SrvInit}()$ outputs an initial server state $st_S \in \mathcal{S}_S$.
- $st \leftarrow \text{Init}(P, k)$ outputs initial client state $st \in \mathcal{S}_C$ for party $P \in \{0, 1\}$ and a secret shared channel key $k \in \mathcal{K}$.
- $st', c \leftarrow \text{Snd}(P, st, m)$ is a client procedure that produces a ciphertext $c \in \mathcal{C}$ and updated client state $st' \in \mathcal{S}_C$ for party P , with original client state $st \in \mathcal{S}_C$, corresponding to an input message $m \in \mathcal{M}$. The ciphertext c contains a franking tag $c_f \in \mathcal{Q}$, which is a commitment to m , and a sending index $i \in \mathbb{Z}^*$.
- $st'_S, t_s \leftarrow \text{TagSend}(st_S, P, c_f)$ is a server procedure that produces a tag $t_s \in \mathcal{T}_S$ for a message sending event, where P is the sending party, and c_f is the franking tag for the message.
- $st_S, t_r \leftarrow \text{TagRecv}(st_S, P, c_f)$ is a server procedure that produces a tag $t_r \in \mathcal{T}_R$ for a message reception event by receiving party P . This procedure is invoked only when the receiving client indicates that the message was successfully received and valid.
- $st', m, k_f, i \leftarrow \text{Rcv}(P, st, c)$ is a client procedure that processes a received ciphertext $c \in \mathcal{C}$ and decrypts it to a message $m \in \mathcal{M} \cup \{\perp\}$ with sending

index $i \in \mathbb{Z}^*$ and a franking key $k_f \in \mathcal{K}_f$. The message m is \perp if and only if decryption fails.

- $G \leftarrow \text{Judge}(st_S, \rho)$ takes as input the server state st_S as well as a client-provided report $\rho \in \{(\{0, 1\} \times \mathcal{M} \times \mathcal{K}_f \times \mathcal{Q} \times \mathcal{T}_S \times \mathcal{T}_R)\}$, which is a set of tuples of the form $(P, m, k_f, c_f, t_s, t_r)$. This procedure verifies the report and, if the report is valid, produces a causality graph $G \in (\mathcal{V} \times \mathcal{E}) \cup \{\perp\}$ for the messages contained within the report. If the reporting information is invalid, the procedure outputs \perp . Note, we enforce here that only messages that have been sent and received can be reported.

To illustrate the semantics of a transcript franking scheme, we provide a brief example of how these procedures are invoked. At initialization time, the server runs SrvInit to produce an initial state. When two clients initiate a conversation, both clients run $\text{Init}(P, k)$ over a shared key k to obtain initial client states. When client P sends a message m , it obtains c from the output of $\text{Snd}(P, st, m)$. The ciphertext c is sent to a platform server, which then invokes $\text{TagSend}(st_S, P, c_f)$ to produce a tag t_s , which is returned to P . Eventually, \bar{P} contacts the server to retrieve outstanding messages. Upon doing so, the server transmits the ciphertext c , along with t_s , to \bar{P} , which decrypts it by invoking Rcv . Upon indicating valid reception to the platform, the server runs TagRecv to produce t_r , which is sent to both P and \bar{P} . When party P wishes to report a set of messages, they compile $(P, m, k_f, c_f, t_s, t_r)$ for each message in ρ . The moderator then runs Judge to produce a causality graph G .

Correctness. Informally, correctness requires that all honestly generated and tagged messages can be successfully received and that any sub-graph of the full causality graph can be reported. We make this precise with the game shown in Figure 3, and define the correctness advantage of an adversary \mathcal{A} as

$$\text{Adv}_{\text{TF}}^{\text{corr}}(\mathcal{A}) = \Pr[\mathbf{G}_{\text{TF}}^{\text{corr}}(\mathcal{A}) = 1] .$$

Formally, a transcript franking scheme TF achieves correctness if $\text{Adv}_{\text{TF}}^{\text{corr}}(\mathcal{A}) = 0$ for all adversaries \mathcal{A} .

Tagging reception events. We discuss in detail what it means for the platform to tag a successful reception event. Recall that our goal is to enable reporters to include their own sent messages within a report without interaction from the other party. The reception tag serves as a way to achieve this goal, acting as an acknowledgement from the other party that the reported message was received. However, in order for this acknowledgement to be meaningful, we must carefully consider at which point the platform generates the reception tag. One option is to generate the tag once the recipient sends a message to the service provider indicating that their verification check passed, meaning the franking tag corresponds to the received plaintext. This would require two round-trips, the first for to retrieve the message, and the second to explicitly tell the server that it was well-formed.

Another option is to do this in one round trip: immediately tag the reception event and send the reception tag along with the ciphertext. If the ciphertext

<p>$G_{\text{TF}}^{\text{corr}}(\mathcal{A})$:</p> <p>$k_{\text{Srv}} \leftarrow \mathcal{K}, st_{\mathcal{A}}, k_{\text{Ch}} \leftarrow \mathcal{A}(), \text{win} \leftarrow 0$ $st_S \leftarrow \text{SrvInit}(k_{\text{Srv}})$ $st_0 \leftarrow \text{Init}(0, k_{\text{Ch}}), st_1 \leftarrow \text{Init}(1, k_{\text{Ch}})$ $\mathcal{R}_t, \mathcal{R}_r, \mathcal{R} \leftarrow \{\}, \{\}, \{\}$ $\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{\text{Ch}})$ return win</p> <p>$\mathcal{O}.\text{SendTag}(P, m)$:</p> <p>$(st_P, c) \leftarrow \text{Snd}(P, st_P, m)$ $st_S, t_s \leftarrow \text{TagSend}(st_S, P, c.c_f)$ $G \leftarrow G + (\mathcal{S}, P, m)$ Add (P, c, t_s) to \mathcal{R}_t return c, t_s</p>	<p>$\mathcal{O}.\text{RecvTag}(P, c, t_s)$:</p> <p>Assert $(\bar{P}, c.c_f, t_s) \in \mathcal{R}_t$ Assert $(P, c, t_s) \notin \mathcal{R}$ Add (P, c, t_s) to \mathcal{R} $st_P, m, k_f, i \leftarrow \text{Rcv}(P, st_P, c)$ if $m \neq \perp$ then $st_S, t_r \leftarrow \text{TagRecv}(st_S, P, c_f)$ $G \leftarrow G + (\mathcal{R}, P, c.i)$ Add $(P, m, k_f, c_f, t_s, t_r)$ to \mathcal{R}_r else win $\leftarrow 1$ return m, k_f, t_s, t_r</p> <p>$\mathcal{O}.\text{Rep}(\rho)$:</p> <p>Assert $\rho > 0$ $G' = \text{Judge}(st_S, \rho)$ if $\rho \subseteq \mathcal{R}_r \wedge ((G' = \perp) \vee (G' \not\subseteq G))$: win $\leftarrow 1$</p>
--	---

Fig. 3. The security game for transcript franking correctness.

is malformed, the recipient can issue a complaint to the service provider, nullifying the reception tag in question. Therefore, two round trips are made only if the ciphertext is malformed. An implementation may also enforce a reasonable time window within which to make such a complaint. We discuss receiver acknowledgement further in Section 7.

Comparison to Chen-Fischlin. Observe that our formalism, unlike that of [11] includes two server-side tagging procedures as opposed to one. This makes possible acknowledgement of message receipt by the platform and message delivery to the recipient client device. As a result, the server, as opposed to client devices, becomes the authority on message ordering, leading to additional security benefits as we discuss next. Instead of having a single `Init` procedure shared by the client and server, we specify `SrvInit`, and `Init`. The syntax and semantics of the message franking channel in Chen-Fischlin does not enable parties to report their own sent messages while our formalism does. While Chen and Fischlin focus on two-party channels, we show how to enable transcript franking for group channels in Section 6. We provide a more in-depth comparison in Appendix A.

4 Security Definitions for Two-Party Transcript Franking

In this section, we introduce security notions for transcript franking. Our setting requires that the platform is the same entity that handles moderation reports. Recall that we defined the transcript franking syntax and semantics in Section 3. Our security definitions formalize notions of confidentiality and accountability

for the reporting process. Accountability consists of two properties, reportability and transcript integrity, which we further explain in the remainder of the section.

Threat model, informally. As the platform is trusted for handling user reports, we trust it to serve as source of ground truth for the ordering of messages. This does not mean that we trust the platform with the contents of messages or that we assume a malicious platform will not attempt to maul ciphertexts. We do not explicitly model the public key infrastructure used to authenticate users, though we note that solutions such as key transparency [10, 20, 23, 26, 32] enable PKIs without placing full trust in the service provider.

Transcript reportability. When a client accepts a message as valid, it should be the case that this message can be successfully reported to the moderator as well. Transcript reportability, which is formally specified by a security game in Figure 4, is a security property we define that captures this goal. The adversary \mathcal{A} attempts to craft a report, containing messages accepted by an honest recipient, that does not verify for the moderator. We define the reportability advantage of an adversary \mathcal{A} for a transcript franking scheme TF as follows:

$$\text{Adv}_{\text{TF}}^{\text{tr-rep}}(\mathcal{A}) = \Pr[\mathbf{G}_{\text{TF}}^{\text{tr-rep}}(\mathcal{A}) = 1] .$$

Transcript integrity. To model malicious reporters that attempt to trick moderators into accepting incorrect causality graphs, we define a security notion called transcript integrity, which is captured by the game in Figure 5. The adversary \mathcal{A} controls both parties and has access to three oracles: **SendTag**, **RecvTag** and **Rep**. A ground truth causality graph G is maintained by the game and updated by **SendTag** and **RecvTag**. The adversary wins if it can produce two valid reports, where at least one is not a sub-graph of the ground truth causality graph, or where the generated sub-graphs are inconsistent.

To elaborate, we recall some definitions regarding causality graphs that were given in Section 2. First, recall that \tilde{G} refers to the graph with message labels removed, allowing us to compare with the ground-truth causality graph G maintained by the game. Second, two causal sub-graphs are consistent if there exists a valid causality graph G' of which they are both sub-graphs. This final consistency condition means that there is a unique ground truth causality graph from which sub-graphs can be reported. We view this as a natural lifting of the receiver binding notion proposed in [15] to the multi-message setting. The advantage of an adversary \mathcal{A} in the transcript integrity game is defined as follows:

$$\text{Adv}_{\text{TF}}^{\text{tr-int}}(\mathcal{A}) = \Pr[\mathbf{G}_{\text{TF}}^{\text{tr-int}}(\mathcal{A}) = 1] .$$

Confidentiality. In order for a transcript franking scheme to achieve confidentiality, the reporting process must not reveal any information about unreported messages. Of course, the underlying messaging channel itself must provide confidentiality as well. We formalize this property in a security game in Figure 6, inspired by the real-or-random multi-opening confidentiality notion proposed in [15]. Our definition uses the function $\text{clen} : \mathcal{M} \rightarrow \mathbb{Z}^*$, which outputs the

$G_{TF}^{tr-rep}(\mathcal{A})$:	$\mathcal{O}.\text{RecvTag}(P, c, t_s)$:	$\mathcal{O}.\text{TagSend}(P, c_f)$:
$k_{Srv} \leftarrow \$\mathcal{K}$	Assert $(\bar{P}, c, c_f, t_s) \in \mathcal{R}_t$	$G \leftarrow G + (\mathbb{S}, P)$
$st_{\mathcal{A}}, k_{Ch} \leftarrow \$\mathcal{A}()$	Assert $(P, c, t_s) \notin \mathcal{R}$	$st_S, t_s \leftarrow \$\text{TagSend}(st_S, P, c_f)$
$win \leftarrow 0; \mathcal{R} \leftarrow \{\}$	Add (P, c, t_s) to \mathcal{R}	Add (P, c_f, t_s) to \mathcal{R}_t
$st_S \leftarrow \$\text{SrvInit}(k_{Srv})$	$st_P, m, k_f, i \leftarrow \text{Rcv}(P, st_P, c)$	return t_s
$st_0 \leftarrow \$\text{Init}(0, k_{Ch})$	if $m \neq \perp$ then	$\mathcal{O}.\text{Rep}(\rho)$:
$st_1 \leftarrow \$\text{Init}(1, k_{Ch})$	$st_S, t_r \leftarrow \$\text{TagRecv}(st_S, P, c_f)$	Assert $ \rho > 0$
$G, \mathcal{R}_t, \mathcal{R}_r \leftarrow \varepsilon, \{\}, \{\}$	Add $(P, m, k_f, c_f, t_s, t_r)$ to \mathcal{R}_r	$G' \leftarrow \text{Judge}(st_S, \rho)$
$\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{Ch})$	$G \leftarrow G + (\mathbb{R}, P, c, i)$	if $G' \not\subseteq G \wedge \rho \subseteq \mathcal{R}_r$:
return win	return m, k_f, t_s, t_r	win $\leftarrow 1$
	$\mathcal{O}.\text{Send}(P, m)$:	
	$(st_P, c) \leftarrow \$\text{Snd}(P, st_P, m)$	
	return c	

Fig. 4. The security game for transcript reportability.

$G_{TF}^{tr-int}(\mathcal{A})$:	$\mathcal{O}.\text{SendTag}(P, c)$:	$\mathcal{O}.\text{RecvTag}(P, c, t_s)$:
$k_{Srv} \leftarrow \$\mathcal{K}; win \leftarrow 0$	$G \leftarrow G + (\mathbb{S}, P)$	Assert $(\bar{P}, c, t_s) \in \mathcal{R}_t$
$st_{\mathcal{A}}, k_{Ch} \leftarrow \$\mathcal{A}()$	$st_S, t_s \leftarrow \text{TagSend}(st_S, P, c, c_f)$	Assert $(P, c, t_s) \notin \mathcal{R}$
$st_S \leftarrow \$\text{SrvInit}(k_{Srv})$	Add (P, c, t_s) to \mathcal{R}_t	Add (P, c, t_s) to \mathcal{R}
$st_0 \leftarrow \$\text{Init}(0, k_{Ch})$	return t_s	$st_S, t_r \leftarrow \text{TagRecv}(st_S, P, c, c_f)$
$st_1 \leftarrow \$\text{Init}(1, k_{Ch})$		$G \leftarrow G + (\mathbb{R}, P, c, i)$
$G, \mathcal{R}_t, \mathcal{R} \leftarrow \varepsilon, \{\}, \{\}$		return t_r
$\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{Ch})$		$\mathcal{O}.\text{Rep}(\rho_1, \rho_2)$:
return win		Assert $ \rho_1 > 0$ and $ \rho_2 > 0$
		$G_1 \leftarrow \text{Judge}(st_S, \rho_1)$
		$G_2 \leftarrow \text{Judge}(st_S, \rho_2)$
		if $G_1 \neq \perp \wedge G_2 \neq \perp \wedge$
		$((G_1 \not\subseteq G) \vee (G_2 \not\subseteq G))$
		$\vee (G_1 \not\approx G_2)$:
		win $\leftarrow 1$

Fig. 5. The security game for transcript integrity.

length of a ciphertext for plaintext m . The ROR-advantage against the confidentiality of a transcript franking scheme TF for an adversary \mathcal{A} is:

$$\text{Adv}_{TF}^{\text{conf}}(\mathcal{A}) = |\Pr[G_{TF,0}^{\text{conf}}(\mathcal{A})] - \Pr[G_{TF,1}^{\text{conf}}(\mathcal{A})]|.$$

5 Our Construction

The key idea of our construction is to report platform-tagged acknowledgements of message sending and reception. These acknowledgements contain counters, maintained as part of the server state, that allow the moderator to reliably reconstruct a portion of the causality graph corresponding to the platform's view of message transmission. Since our construction uses four counters per

$\mathbf{G}_{\text{TF},b}^{\text{conf}}(\mathcal{A}):$ $k_{\text{Srv}} \leftarrow \$ \mathcal{K}_S; k_{\text{Ch}} \leftarrow \$ \mathcal{K}_C$ $st_{\mathcal{A}} \leftarrow \$ \mathcal{A}(), \mathcal{Y} \leftarrow \{\}$ $st_S \leftarrow \$ \text{SrvInit}(k_{\text{Srv}})$ $st_0 \leftarrow \$ \text{Init}(0, k_{\text{Ch}}), st_1 \leftarrow \$ \text{Init}(1, k_{\text{Ch}})$ $\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}})$ return \hat{b} $\mathcal{O}.\text{Send}(P, m):$ $(st_P, c) \leftarrow \$ \text{Snd}(P, st_P, m)$ $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{c\}$ return c	$\mathcal{O}.\text{Recv}(P, c, t_s):$ Assert $c \in \mathcal{Y}$ $st_P, m, k_f, i \leftarrow \text{Rcv}(P, st_P, c)$ return m, k_f $\mathcal{O}.\text{ChalSend}(P, m):$ $(st_P, c_0) \leftarrow \$ \text{Snd}(P, st_P, m)$ $c_1 \leftarrow \$ \{0, 1\}^{\text{clen}(m)}$ return c_b
--	---

Fig. 6. The security game for transcript franking confidentiality.

conversation, we call it the quad-counter construction (QCC). We present the pseudocode specification of our construction in Figure 7, which specifies how the service provider handles state for a single conversation between two parties. Parallelizing this for multiple conversations can be done in a straightforward manner, as we further discuss in Section 7.

Client logic. The client procedures `Init`, `Snd`, and `Rcv` comprise a secure messaging channel with reportable franking tags c_f , committing to plaintext content m , with the opening k_f . Indeed, these three procedures form a message franking channel [11, 16].

Server logic. Upon server initialization, sending and reception counters for each party are initialized to 0 and the server samples a MAC key. When a party P sends a message, the server increments the send counter for P and tags the send event. Similarly, it increments the reception counter for P when P successfully receives a message, and then produces a tag for this event. In the pseudocode, the symbols S and R are labels that denote sending and reception events respectively.

Reporting. To report a set of messages, a client compiles the message contents m , the franking key k_f , the franking tag c_f , the send tag `TagSend`, and the reception tag t_r for each message. The client then forwards this information to the platform within a single report object ρ . The platform verifies the commitments for each message along with its own MAC tags. Then, it uses the indexes within the tags to construct and order the vertices for the sub-graph, and it adds edges between vertices that correspond to the same message. A moderator can interpret the contiguity of vertices as explained in Section 2.

Security proofs. We now demonstrate the security of our transcript franking construction, TF. We begin by proving transcript integrity. The following lemma will be helpful for our proof.

Lemma 1. *Let G_1 and G_2 be two valid two-party causality sub-graphs. Suppose $G_1 \approx G_2$ but $G_1 \not\approx G_2$. Then there must be $v_1 \in G_1.V$ and $v_2 \in G_2.V$ such that $v_1[0 : 2] = v_2[0 : 2]$ but $v_1.m \neq v_2.m$.*

<p><u>SrvInit():</u> $k_{\text{mac}} \leftarrow \\$\mathcal{K}; cs_0, cr_0, cs_1, cr_1 \leftarrow (0, 0, 0, 0)$ return $\{k_{\text{mac}}, cs_0, cr_0, cs_1, cr_1\}$</p> <p><u>Init($P, k$):</u> return Ch.Init(P, k)</p> <p><u>Snd(P, st, m):</u> $(k_f, c_f) \leftarrow \text{Com}(m)$ $(st.st_{\text{Ch}}, c_e) \leftarrow \\$\text{Ch.Snd}(P, st.st_{\text{Ch}}, (m, k_f))$ return $st, (c_e, c_f)$</p> <p><u>TagSend(st_S, P, c_f):</u> $cs_P \leftarrow cs_P + 1, \text{ack} \leftarrow (S, P, \bar{P}, c_f, cs_P, cr_P)$ $t_s \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_s</p> <p><u>TagRecv(st_S, P, c_f):</u> $cr_P \leftarrow cr_P + 1, \text{ack} \leftarrow (R, \bar{P}, P, c_f, cs_P, cr_P)$ $t_r \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_r</p>	<p><u>Rcv(P, st, c, t_s, t_r):</u> $(st.st_{\text{Ch}}, m, k_f, i) \leftarrow \text{Ch.Rcv}(P, st_{\text{Ch}}, c)$ if $m = \perp \vee \text{VerC}(m, k_f, c.c_f) = 0$: return \perp return st, m, k_f, i</p> <p><u>Judge(st_S, ρ):</u> Initialize empty graph G For $(P, m, k_f, c_f, t_s, t_r)$ in ρ $b \leftarrow \text{Ver}(k_{\text{mac}}, t_s.\text{ack}, t_s.\text{tag}) \wedge$ $\text{Ver}(k_{\text{mac}}, t_r.\text{ack}, t_r.\text{tag}) \wedge$ $\text{VerC}(m, k_f, c.c_f) \wedge t_s[0] = S \wedge$ $t_r[0] = R \wedge t_s.c_f = t_r.c_f$ if $b = 0$: return \perp $cs_P, cr_P = t_s.\text{ack}.(cs, cr)$ $cs_{\bar{P}}, cr_{\bar{P}} = t_r.\text{ack}.(cs, cr)$ $v_s = (S, cs_P, cr_P, m)$ $v_r = (R, cs_{\bar{P}}, cr_{\bar{P}}, m)$ Add v_s to $G.V_P$, add v_r to $G.V_{\bar{P}}$ Add (v_s, v_r) to $G.E$ return G</p>
---	--

Fig. 7. Pseudocode for our two-party transcript franking construction.

Proof. Assume for the sake of contradiction that for all $v_1 \in \widetilde{G}_1.V$ and $v_2 \in \widetilde{G}_2.V$, that if $v_1[0 : 2] = v_2[0 : 2]$, then $v_1.m = v_2.m$. Since $\widetilde{G}_1 \approx \widetilde{G}_2$, there exists some valid causality graph G such that $\widetilde{G}_1, \widetilde{G}_2 \subseteq G$. This means that there exists a sequence of send and receive operations that constructs G . We can use the same sequence of operations to generate a valid message-inclusive graph G^* , such that $G_1, G_2 \subseteq G^*$, contradicting the assumption that $G_1 \not\approx G_2$. In each send operation, we simply include the message corresponding to the vertex $v \in G_1.V \cup G_2.V$, if the counters for that send operation correspond to a vertex in the union of the two vertex sets. By our initial assumption a unique such vertex exists. For all other send operations, we can include an arbitrary message, the empty string for instance. This completes the proof. \square

Theorem 1. Let TF be the transcript franking scheme given in Figure 7. Let \mathcal{A} be a transcript integrity adversary against TF. Then we give an EUF-CMA adversary \mathcal{B} and V-Bind adversary \mathcal{C} such that

$$\text{Adv}_{\text{TF}}^{\text{tr-int}}(\mathcal{A}) \leq \text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) + \text{Adv}_{\text{CS}}^{\text{v-bind}}(\mathcal{C}).$$

Adversaries \mathcal{B} and \mathcal{C} run in time that of \mathcal{A} plus a small overhead.

Proof. We proceed via a sequence of game hops with Game \mathbf{G}_0 equivalent to $\mathbf{G}_{\text{TF}}^{\text{tr-int}}$, defined in Figure 5. To aid with future game definitions, we provide some additional bookkeeping, initializing an empty set \mathcal{R}' at the start of the game. Game \mathbf{G}_0 adds $(S, P, \bar{P}, c.c_f, G.(cs_P, cr_P))$ to \mathcal{R}' before the **return** statement

of **SendTag**. Similarly, it adds $(R, \bar{P}, P, c.c_f, G.(cs_P, cr_P))$ to \mathcal{R}' before the **return** statement of **RecvTag**.

The adversary \mathcal{A} can only win if it is able to produce ρ_1, ρ_2 such that $\widetilde{G}_1 \not\subseteq G$ or $\widetilde{G}_2 \not\subseteq G$ or $G_1 \not\approx G_2$, where $G_1 \leftarrow \text{Judge}(st_S, \rho_1)$, $G_2 \leftarrow \text{Judge}(st_S, \rho_2)$, and G is the ground truth graph maintained by the game. From (ρ_1, ρ_2) , we will show that we can break either the unforgeability of the MAC or the binding property of the commitment.

We consider two cases: (1) the adversary wins with \widetilde{G}_1 or \widetilde{G}_2 not a subgraph of G or (2) the adversary wins with $G_1 \not\approx G_2$, but $\widetilde{G}_1, \widetilde{G}_2 \subseteq G$. The first case will allow us to reduce to the EUF-CMA security of the MAC while the second allows us to reduce to the binding security of the commitment. Each case will correspond to a distinct failure event. Let \mathbf{G}_1 be the same as \mathbf{G}_0 , except we abort and output 0, right before setting the win flag, if \mathcal{A} produces a valid (ρ_1, ρ_2) in case (1). We denote the event that this abort occurs as F_1 . Let \mathbf{G}_2 be the same as \mathbf{G}_1 except we abort and output 0 at the same location if \mathcal{A} produces a valid (ρ_1, ρ_2) in case (2). We denote the event that this abort occurs as F_2 . Note that $|\Pr[\mathbf{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathbf{G}_1(\mathcal{A}) \Rightarrow 1]| \leq \Pr[F_1]$ and $|\Pr[\mathbf{G}_1(\mathcal{A}) \Rightarrow 1] - \Pr[\mathbf{G}_2(\mathcal{A}) \Rightarrow 1]| \leq \Pr[F_2]$. Furthermore, $\Pr[\mathbf{G}_2(\mathcal{A}) \Rightarrow 1] = 0$, so we have $\text{Adv}^{\text{tr-int}}(\mathcal{A}) = \Pr[\mathbf{G}_0(\mathcal{A}) \Rightarrow 1] \leq \Pr[F_1] + \Pr[F_2]$.

We now demonstrate an adversary \mathcal{B} where $\text{Adv}^{\text{euf-cma}}(\mathcal{B}) = \Pr[F_1]$. The adversary \mathcal{B} perfectly simulates \mathbf{G}_0 to \mathcal{A} , while routing **Tag** and **Verify** calls to its challenger oracles. If F_1 occurs, then we have that $\widetilde{G}_i \not\subseteq G$ for some $i \in \{1, 2\}$. For r an element of ρ , define $f(r) = \{r.t_s.\text{ack}, r.t_r.\text{ack}\}$. Observe that $\widetilde{G}_i \not\subseteq G$ implies there is some $r^* = (P^*, m^*, k_f^*, c_f^*, t_s^*, t_r^*) \in \rho_i$, such that $f(r^*) \not\subseteq \mathcal{R}'$.

To see why this is, observe that the construction mirrors the updates of the causality graph perfectly. Put formally, if $\bigcup_{r' \in \rho} f(r') \subseteq \mathcal{R}'$ and $G' \leftarrow \text{Judge}(\rho)$, then $\widetilde{G}' \subseteq G$. We have that $G.(cs_P, cr_P, cs_{\bar{P}}, cr_{\bar{P}}) = (0, 0, 0, 0)$ and the server counters $(cs_P, cr_P, cs_{\bar{P}}, cr_{\bar{P}}) = (0, 0, 0, 0)$ at the start of the game. When **SendTag** is called, we increment $G.cs_P$ and $st_S.cs_P$. Similarly, when **RecvTag** is invoked, we increment $G.cr_P$ and $st_S.cr_P$. A simple proof by induction on the number of oracle calls shows that $G.(cs_P, cr_P) = st_S.(cs_P, cr_P)$ for $P \in \{0, 1\}$ by the end of each call to **SendTag** and **RecvTag**. This means that $v \in \widetilde{G}'.V$ implies $v \in G.V$ and $e \in \widetilde{G}'.E$ implies $e \in G.E$.

If F_1 occurs, then we retrieve the $r^* = (P^*, m^*, k_f^*, c_f^*, t_s^*, t_r^*)$ in question, and observe that $\text{Verify}(k_{\text{mac}}, t_s^*.\text{ack}, t_s^*.\text{tag}) = 1$ and $\text{Verify}(k_{\text{mac}}, t_r^*.\text{ack}, t_r^*.\text{tag}) = 1$, because the output of **Judge** is not \perp . We must have that at least one of $t_r^*.\text{ack}$ or $t_s^*.\text{ack}$ was not queried to the MAC challenger oracle, otherwise both would be in \mathcal{R}' . Let t^* denote this tag. We output $(t^*.\text{ack}, t^*.\text{tag})$ as a forgery.

Now, we construct adversary \mathcal{C} where $\text{Adv}_{\text{CS}}^{\text{v-bind}}(\mathcal{C}) = \Pr[F_2]$. If F_2 occurs, we have that the adversary \mathcal{A} was able to trigger $G_1 \not\approx G_2$ while $\widetilde{G}_1, \widetilde{G}_2 \subseteq G$. By Lemma 1 there exists $v_1 \in G_1$ and $v_2 \in G_2$ such that $v_1[0 : 2] = v_2[0 : 2]$ but $v_1.m \neq v_2.m$. Note that there must also be a single c_f and $k_f^{(1)}, k_f^{(2)}$ such that $\text{VerC}(v_1.m, k_f^{(1)}, c_f) = 1$ and $\text{VerC}(v_2.m, k_f^{(2)}, c_f) = 1$. This breaks the binding

property of the commitment, and so \mathcal{C} outputs $(v_1.m, k_f^{(1)}, v_2.m, k_f^{(2)}, c_f)$ to win with probability $\Pr[F_2]$. This completes the proof. \square

We now show that our scheme also achieves perfect reportability.

Theorem 2. *For all adversaries \mathcal{A} , we have $\text{Adv}_{\text{TF}}^{\text{tr-rep}}(\mathcal{A}) = 0$.*

Proof. Observe that the check that an honest recipient performs in **Rcv**, namely, that $\text{VerC}(m, k_f, c.c_f) = 0$, is replicated in **Judge**. The only way **Judge** can return \perp is if this check fails, if any of the MAC checks fail, or if the input ρ is malformed. Since we are dealing with an honest reporter, this cannot be the case, so **Judge** must always return a non- \perp value. \square

Our scheme achieves correctness via the correctness of the underlying channel **Ch**, the correctness of the MAC scheme (**Tag**, **Verify**), the correctness of the commitment scheme (**Com**, **VerC**), and the fact that the counters in our construction perfectly mirror those of the ground truth graph (see the proof of Theorem 1). Observe that our construction boils down to a commit-then-encrypt scheme, which was proven secure for the multi-opening real-or-random confidentiality notion in [15], hence we omit the proof of confidentiality here.

6 Multi-party Transcript Franking

Up to this point, our constructions have considered transcript franking in the two-party direct messaging context. We now discuss how our approach generalizes to an arbitrary number of parties. A group consists of a set of N parties $\{0, \dots, N-1\}$. The goal of a group transcript franking construction is to be able to reconstruct a causality graph like that shown in Figure 8. Note that, unlike the two-party setting, one send event can correspond to multiple reception events, as there are now multiple recipients. Each edge corresponds to a single copy of the broadcast message. We build on the multi-party channel communication graph formalism proposed in [25], adapting it to our causality graph abstraction.

Causality graphs for group messaging. For N -party communication, a causality graph is an N -partite graph $G = (V, E)$, where $V = \bigcup_{i \in [N]} V_i$. All vertex sets V_i for $i \in [N]$ are disjoint. The edge set E consists of pairs (v, v') where $v \in V_i$ and $v' \in V_j$, where $i, j \in [N]$ and $i \neq j$. The vertex space, as before, is $\{S, R\} \times \mathbb{Z}^* \times \mathbb{Z}^* \times \mathcal{M}$. The notation and updates for adding a send event is the same as the two party version. For adding a reception event to party P_R from party P_S , we write $G \leftarrow G + (\text{R}, P_S, P_R, c.i)$, incrementing the reception counter for P_R before adding the vertex. We then add an edge between the sending vertex and the new reception vertex, as before. The partial ordering over events is given by the transitive closure over the total orders for each vertex set and the edge relation. The total order within each vertex set is given by the lexicographic ordering over $v.(cs, cr)$ for $v \in V_P$. Event contiguity and gaps can be interpreted similarly as the two-party case, as described in Section 2.

Group messaging channels. A group messaging channel **Ch** is defined as the tuple $\text{Ch} = (\text{Init}, \text{Snd}, \text{Rcv})$. The main difference here is that instead of $P \in \{0, 1\}$,

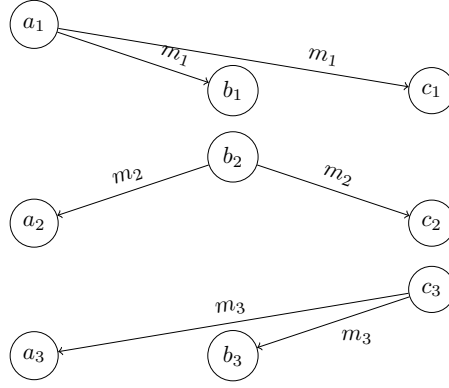


Fig. 8. Example of group causality graph

we have that $P \in [N]$, and our reception procedure accepts the identity of the sending party P_S associated with the ciphertext c .

- $st \leftarrow \text{Init}(P, k)$ outputs initial client state $st \in \mathcal{S}_C$ for a new channel for Party $P \in [N]$ and a key $k \in \mathcal{K}$.
- $st', c \leftarrow \text{Snd}(P, st, m)$ is a client procedure that produces a ciphertext $c \in \mathcal{C}$ corresponding to an input message $m \in \mathcal{M}$, and an updated client state $st' \in \mathcal{S}_C$.
- $st', m, i \leftarrow \text{Rcv}(P_R, st, P_S, c)$ is a client procedure that processes a received ciphertext c from party $P_S \in [N]$ to party $P_R \in [N]$ (where $P_S \neq P_R$) and decrypts it to a message $m \in \mathcal{M} \cup \{\perp\}$ with sending index $i \in \mathbb{Z}^*$. The message m can be \perp if decryption fails.

Group transcript franking syntax and semantics. The Init , Snd , and Rcv procedures inherit the syntactic changes discussed above, and our TagRcv procedure accepts an additional argument for the sending party $P_S \in [N]$. This additional argument is required in the group case since a client can receive a message from more than one party. We inherit notational conventions from Section 4.

- $st_S \leftarrow \text{SrvInit}(N)$ outputs initial server state $st_S \in \mathcal{S}_S$ for an N -party group.
- $st \leftarrow \text{Init}(P, k)$ outputs initial client state $st \in \mathcal{S}_C$ for a new channel for Party $P \in [N]$ and a key $k \in \mathcal{K}$.
- $st', c \leftarrow \text{Snd}(P, st, m)$ is a client procedure that produces a ciphertext $c \in \mathcal{C}$ corresponding to an input message $m \in \mathcal{M}$, and an updated client state $st' \in \mathcal{S}_C$.
- $st'_S, t_s \leftarrow \text{TagSend}(st_S, P_S, c_f)$ is a server procedure that produces a tag $t_s \in \mathcal{T}_S$ for a message sending event, where $P_S \in [N]$ is the sending party, $c_f \in \mathcal{Q}$ is the franking tag for the message, and $st'_S \in \mathcal{S}_S$ is the updated server state.

$G_{\text{TF},N}^{\text{tr-rep}}(\mathcal{A})$:	$\mathcal{O}.\text{RecvTag}(P_R, P_S, c, t_s)$:	$\mathcal{O}.\text{TagSend}(P, c_f)$:
$k_{\text{Srv}} \leftarrow \\mathcal{K}	Assert $(P_S, c, c_f, t_s) \in \mathcal{R}_t$	$st_S, t_s \leftarrow \text{TagSend}(st_S, P, c_f)$
$st_A, k_{\text{Ch}} \leftarrow \$\mathcal{A}()$	Assert $(P, c, t_s) \notin \mathcal{R}$	Add (P, c_f, t_s) to \mathcal{R}_t
$\text{win} \leftarrow 0; \mathcal{R} \leftarrow \{\}$	Add (P, c, t_s) to \mathcal{R}	return t_s
$st_S \leftarrow \$\text{SrvInit}(k_{\text{Srv}})$	$st_P, m, k_f, i \leftarrow \text{Rcv}(P, st_P, c)$	
For $i \in [N]$	if $m \neq \perp$ then	$\mathcal{O}.\text{Rep}(\rho)$:
$st_i \leftarrow \$\text{Init}(i, k_{\text{Ch}})$	$st_S, t_r \leftarrow \$\text{TagRecv}(st_S, P_R, P_S, c_f)$	Assert $ \rho > 0$
$\mathcal{R}_t, \mathcal{R}_r \leftarrow \{\}, \{\}$	Add $(P_S, P_R, m, k_f, c_f, t_s, t_r)$	$G' = \text{Judge}(st_S, \rho)$
$\mathcal{A}^{\mathcal{O}}(st_A, k_{\text{Ch}})$	to \mathcal{R}_r	if $G' = \perp \wedge \rho \subseteq \mathcal{R}_r$:
return win	return m, k_f, t_s, t_r	win $\leftarrow 1$

Fig. 9. The security game for transcript reportability for N -party messaging.

- $st_S, t_r \leftarrow \text{TagRecv}(st_S, P_S, P_R, c_f)$ is a server procedure that produces a tag $t_r \in \mathcal{T}_R$ for a message reception event by receiving party $P_R \in [N]$ for a message sent by party $P_S \in [N]$. This procedure is invoked only when the receiving client indicates that the message was successfully received and valid.
- $st', m, k_f, i \leftarrow \text{Rcv}(P_R, st, P_S, c)$ is a client procedure that processes a received ciphertext $c \in \mathcal{C}$ and decrypts it to a message $m \in \mathcal{M} \cup \{\perp\}$ and a franking key $k_f \in \mathcal{K}_f$. The message m can be \perp if decryption fails.
- $G \leftarrow \text{Judge}(st_S, \rho)$ takes as input the server state $st_S \in \mathcal{S}_S$ as well as a client-provided report ρ , which is a set of tuples of the form $(P_S, P_R, m, k_f, c_f, t_s, t_r)$. This procedure verifies the report and, if the report is valid, produces a causality graph $G \in (\mathcal{V} \times \mathcal{E}) \cup \{\perp\}$ for the messages contained within the report. If the reporting information is invalid, the procedure outputs \perp .

Security definitions. Our security notions in the group setting are a natural extension of those for the two-party setting. The main difference is that N parties are initialized and any of these parties can send and receive messages within the same channel. Correctness and confidentiality definitions generalize in a straightforward manner, hence we omit full descriptions of them for brevity. We present our group transcript reportability definition in Figure 9 and our group transcript integrity definition in Figure 10. To denote the advantage of an adversary \mathcal{A} in the N -party reportability game, we write $\text{Adv}_{\text{TF},N}^{\text{tr-rep}}(\mathcal{A})$. Similarly, $\text{Adv}_{\text{TF},N}^{\text{tr-int}}(\mathcal{A})$ is the advantage of \mathcal{A} in the N -party transcript integrity game.

Our construction. The group messaging transcript franking construction generalizes naturally from the two-party version. We provide a pseudocode specification of our group transcript franking protocol in Figure 11. Counter updates happen nearly identically in **TagSend** and **TagRecv**, except now N pairs of counters are maintained, one for each party.

Security analysis. The security analysis of our group construction closely mirrors that of the two-party construction. As with the two-party construction, our group construction achieves perfect reportability because the **Rcv** procedure per-

$\mathbf{G}_{\text{TF},N}^{\text{tr-int}}(\mathcal{A})$: $k_{\text{Srv}} \leftarrow \$ \mathcal{K}$; $\text{win} \leftarrow 0$ $st_{\mathcal{A}}, k_{\text{Ch}} \leftarrow \$ \mathcal{A}()$ $st_S \leftarrow \$ \text{SrvInit}(k_{\text{Srv}})$ For $i \in [N]$ $st_i \leftarrow \$ \text{Init}(i, k_{\text{Ch}})$ $G, \mathcal{R}_t, \mathcal{R} \leftarrow \varepsilon, \{\}, \{\}$ $\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{\text{Ch}})$ return win	$\mathcal{O}.\text{SendTag}(P, m, c, k_f)$: $G \leftarrow G + (\text{S}, P, m)$ $st_S, t_s \leftarrow \text{TagSend}(st_S, P, c, c_f)$ Add (P, c, t_s) to \mathcal{R}_t return t_s	$\mathcal{O}.\text{RecvTag}(P_R, P_S, c, t_s)$: Assert $(\bar{P}, c, t_s) \in \mathcal{R}_t$ Assert $(P, c, t_s) \notin \mathcal{R}$ Add (P, c, t_s) to \mathcal{R} $st_S, t_r \leftarrow \text{TagRecv}(st_S, P_R, P_S, c, c_f)$ $G \leftarrow G + (\text{R}, P_S, P_R, c, i)$ return t_r $\mathcal{O}.\text{Rep}(\rho_1, \rho_2)$: Assert $ \rho_1 > 0$ and $ \rho_2 > 0$ $G_1 \leftarrow \text{Judge}(st_S, \rho_1)$ $G_2 \leftarrow \text{Judge}(st_S, \rho_2)$ if $G_1 \neq \perp \wedge G_2 \neq \perp \wedge$ $((G_1 \not\subseteq G) \vee (G_2 \not\subseteq G) \vee (G_1 \not\approx G_2))$: win $\leftarrow 1$
---	--	---

Fig. 10. The security game for group transcript integrity for N -party messaging.

forms the same commitment checks as the **Judge** procedure. Below, we prove the transcript integrity of our scheme.

Theorem 3. *Let TF be the group transcript franking scheme given in Figure 11. Let \mathcal{A} be an N -party group transcript integrity adversary against TF. Then we give an EUF-CMA adversary \mathcal{B} and V-Bind adversary \mathcal{C} such that*

$$\text{Adv}_{\text{TF},N}^{\text{tr-int}}(\mathcal{A}) \leq \text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) + \text{Adv}_{\text{CS}}^{\text{v-bind}}(\mathcal{C}).$$

Adversaries \mathcal{B} and \mathcal{C} run in time that of \mathcal{A} plus a small overhead.

Proof. We proceed as with the proof of the two-party construction. The games \mathbf{G}_0 , \mathbf{G}_1 , and \mathbf{G}_2 are defined as before, with a similar security argument. Here, we highlight notable differences in the group case. The additional bookkeeping \mathcal{R}' is initialized to $\{\}$, as before, at the start of the game. Game \mathbf{G}_0 adds $(S, P_S, P_R, c, c_f, G.(cs_P, cr_P))$ to \mathcal{R}' before the **return** statement of **SendTag**. Similarly, it adds $(R, P_S, c, c_f, G.(cs_P, cr_P))$ to \mathcal{R}' before the **return** statement of **RecvTag**. The failure events F_1 and F_2 are defined as in the proof of Theorem 1.

We now demonstrate an adversary \mathcal{B} where $\text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) = \Pr[F_1]$. The adversary \mathcal{B} perfectly simulates \mathbf{G}_0 to \mathcal{A} , while routing **Tag** and **Verify** calls to its challenger oracles. If F_1 occurs, then we have that $\widetilde{G}_i \not\subseteq G$ for some $i \in \{1, 2\}$. For r an element of ρ , define $f(r) = \{r.t_s.\text{ack}, r.t_r.\text{ack}\}$. Observe that $\widetilde{G}_i \not\subseteq G$ implies there is some $r^* = (P_S^*, P_R^*, m^*, k_f^*, c_f^*, t_s^*, t_r^*) \in \rho_i$, such that $f(r^*) \not\subseteq \mathcal{R}'$.

This results from the fact that $\bigcup_{r' \in \rho} f(r') \subseteq \mathcal{R}'$ and $G' \leftarrow \text{Judge}(\rho)$ implies $\widetilde{G}' \subseteq G$. We have that $G.(cs_P, cr_P) = (0, 0)$ for $P \in [N]$ and the server counters $st_S.(cs_P, cr_P) = (0, 0)$ for $P \in [N]$ at the start of the game. When **SendTag** is called, we increment $G.cs_P$ and $st_S.cs_P$. Similarly, when **RecvTag** is invoked, we increment $G.cr_P$ and $st_S.cr_P$. This means that $v \in \widetilde{G}'.V$ implies $v \in G.V$

<p>SrvInit(N):</p> $k_{\text{mac}} \leftarrow \\mathcal{K} For $i \in [N]$ do $cs_i, cr_i \leftarrow 0, 0$ return $\{k_{\text{mac}}\} \cup \{cs_i, cr_i\}_{i \in [N]}$	<p>Rcv(P_R, st, P_S, c):</p> $(st.st_{\text{Ch}}, m, k_f, i) \leftarrow \text{Ch.Rcv}(P_R, st_{\text{Ch}}, P_S, c)$ if $m = \perp \vee \text{VerC}(m, k_f, c.c_f) = 0$: return \perp return st, m, k_f, i
<p>Init(P, k):</p> return $\text{Ch.Init}(P, k)$	<p>Judge(st_S, ρ):</p> Initialize empty graph G For $(P_S, P_R, m, k_f, c_f, t_s, t_r)$ in ρ : $b \leftarrow \text{Ver}(k_{\text{mac}}, t_s.\text{ack}, t_s.\text{tag}) \wedge$ $\text{Ver}(k_{\text{mac}}, t_r.\text{ack}, t_r.\text{tag}) \wedge$ $\text{VerC}(m, k_f, c.c_f) \wedge$ $t_s[0] = S \wedge t_r[0] = R \wedge$ $t_s.c_f = t_r.c_f$ if $b = 0$: return \perp $cs_P, cr_P = t_s.\text{ack}.(cs, cr)$ $cs_{P_R}, cr_{P_R} = t_r.\text{ack}.(cs, cr)$ $v_s = (S, cs_P, cr_P, m)$ $v_r = (R, cs_{P_R}, cr_{P_R}, m)$ if $v_s \notin G.V_P$ Add v_s to $G.V_P$ Add v_r to $G.V_{P_R}$, add (v_s, v_r) to $G.E$ return G
<p>Snd(P, st, m):</p> $(k_f, c_f) \leftarrow \text{Com}(m)$ $(st.st_{\text{Ch}}, c_e) \leftarrow \$\text{Ch.Snd}(P, st.st_{\text{Ch}}, (m, k_f))$ return $st, (c_e, c_f)$	
<p>TagSend(st_S, P, c_f):</p> $cs_P \leftarrow cs_P + 1, \text{ack} \leftarrow (S, P, c_f, cs_P, cr_P)$ $t_s \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_s	
<p>TagRecv(st_S, P_R, P_S, c_f):</p> $cr_{P_R} \leftarrow cr_{P_R} + 1$ $\text{ack} \leftarrow (R, P_S, P_R, c_f, cs_{P_R}, cr_{P_R})$ $t_r \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_r	

Fig. 11. Pseudocode for our N -party transcript franking construction.

and $e \in \widetilde{G}'.E$ implies $e \in G.E$. This allows us to produce a forgery as shown in the proof for the two-party case.

Now, we construct adversary \mathcal{C} where $\text{Adv}_{\text{CS}}^{\text{v-bind}}(\mathcal{C}) = \Pr[F_2]$. If F_2 occurs, we have that the adversary \mathcal{A} was able to trigger $G_1 \not\approx G_2$ while $\widetilde{G}_1, \widetilde{G}_2 \subseteq G$. It is straightforward to see that the group version of Lemma 1 holds, so there exists $v_1 \in G_1$ and $v_2 \in G_2$ such that $v_1[0 : 2] = v_2[0 : 2]$ but $v_1.m \neq v_2.m$. We proceed as in the proof of Theorem 1 to produce a binding violation. \square

7 Discussion and Extensions

Composing causality preservation with transcript franking. Our work is primarily concerned with how malicious parties can interfere with the reporting process. Causality preservation as proposed in [11] aims to model how parties in a messaging channel can obtain consistent causality graphs in the presence of a malicious service provider. Indeed, transcript franking can be instantiated with a channel that achieves strong causality preservation to reap these benefits.

Reports with redacted messages. We remark that our construction allows clients to report transmission patterns of messages, via causality sub-graphs, without having to disclose every message within the causality sub-graph. Doing so simply requires omitting the opening key k_f for the messages that a client

wishes to redact within a report. In Chen and Fischlin’s construction of a message franking channel with causality, this would not be possible as the causality metadata is committed to alongside the plaintext message [11].

Malicious clients refusing acknowledgement. In our protocol, clients indicate reception of a well-formed ciphertext to the server before a reception event tag is created. Malicious clients may refuse to make this acknowledgement to the server, preventing the sender of that message from being able to report it. We can mitigate this via notifying senders of message delivery and recipient validation separately, thereby flagging malicious behavior in-band. Upon detecting this behavior, a client may refuse to further communicate with the misbehaving client, but we do not yet support cryptographically reporting this misbehavior.

To explain in more detail, when P sends a message to \bar{P} , there are three key events in the course of message transmission: (1) the reception of the message sent from P by the platform server, (2) the reception of the message by \bar{P} , and (3) platform reception of a valid message acknowledgement from \bar{P} .

If \bar{P} is malicious, it could refuse to indicate the validity of the message, omitting step (3) as described above. When (an honest) P notices that only events (1) and (2) occurred for a particular message, while \bar{P} continues to send and acknowledge subsequent messages, it knows that \bar{P} is acting in an aberrant manner and will halt interactions with \bar{P} (tear down the conversation and alert the user). So detection of deviation is built into our protocol. But our protocol does not enable P to cryptographically prove to a moderator that \bar{P} misbehaved in the way described above. A messaging system might trust client software to report such misbehavior, but cryptographically secure reporting of this class of misbehavior is an open question.

The above issue is analogous to an honest client receiving a message with a malformed franking tag in the standard single-message franking setting. Here the recipient should drop the message, but cannot cryptographically prove to the moderator that the sender sent a malformed ciphertext.

One might wonder if the key-committing aspect of the encryption scheme can come to the rescue here: if a ciphertext can be decrypted only under one key, then the sender needs to simply reveal the key in order to show the ciphertext is well-formed. In messaging protocols, these keys are often intended for one-time use and revealing them does not compromise forward-secrecy or post compromise security. The issue is that there is no straightforward way to prove that the recipient should have been able to derive a particular key from the ratcheting protocol. Hence, proving that a message is decryptable is not sufficient to show that the recipient should have been able to decrypt it. Designing protocols that allow for proving such statements is an interesting future direction.

Deployment considerations. Our definitions consider single conversations, however our constructions can be parallelized in a straightforward manner for multiple conversations. Indeed, the same server MAC key can be used, and as [1] suggests, periodically rotated. Tags will additionally have to include conversation specific identifiers in order to ensure that messages cannot be falsely reported across conversations. This would amount to appending the identifier *cid* to *ack*

before tagging it in `TagSend` and `TagRecv`. Instead of using numerical indices to identify parties, one might use unique user identifiers. This is especially important for groups as membership can change over time, hence so can the mapping between party indexes and user identities within a group. Presenting causality information in a user-friendly way to messaging parties and content moderators is an open question, which was also raised in [11]. Concretely, both the MAC and the commitment scheme can be instantiated with HMAC-SHA-256 [7]. A drawback of our proposed construction is that the server must maintain counters for each party in each channel. At scale, keeping track of this state can be prohibitive. In the remainder of this section, we describe a modification of our scheme that mitigates this issue.

Outsourced-storage transcript franking. We now propose mechanisms for allowing clients to store the counters while the server verifies how it is updated. In our no-server-storage solution, clients send these counters in the associated data of their messages. We present a solution for two-party transcript franking with outsourced storage in the remainder of this section. In Appendix B, we provide a security analysis of this solution and outline its generalization to group transcript franking. We present pseudocode for our outsourced construction in Figure 12, highlighting the procedures that differ from the server-storage version.

Formally, an outsourced two-party transcript franking scheme is a tuple of algorithms $\text{OTF} = (\text{SrvInit}, \text{Init}, \text{Snd}, \text{Rcv}, \text{TagSend}, \text{TagRecv}, \text{Judge}, \text{JudgeReplay})$, defined over a server state space \mathcal{S}_S , a client state space \mathcal{S}_C , a key space \mathcal{K} , a message space \mathcal{M} , a commitment space \mathcal{Q} , a franking key space \mathcal{K}_f , an initialization tag space \mathcal{T}_I , a message-sent tag space \mathcal{T}_S , and a reception tag space \mathcal{T}_R . We detail these algorithms below:

- $st_S, t_0^{(0)}, t_0^{(1)} \leftarrow \text{SrvInit}()$ outputs an initial server state $st_S \in \mathcal{S}_S$, along with initialization tags $t^{(0)}, t^{(1)} \in \mathcal{T}_I$ for each party.
- $st \leftarrow \text{Init}(P, k)$ is defined as in Section 4.
- $st', c \leftarrow \text{Snd}(P, st, m)$ is defined as in Section 4.
- $st'_S, t_s \leftarrow \text{TagSend}(st_S, P, c_f, t)$ is a server procedure that produces a tag $t_s \in \mathcal{T}_S$ for a message sending event, where P is the sending party, c_f is the franking tag, and $t \in \mathcal{T}_S \cup \mathcal{T}_R \cup \mathcal{T}_I$ is the last tag issued for P .
- $st_S, t_r \leftarrow \text{TagRecv}(st_S, P, c_f, t)$ is a server procedure that produces a tag $t_r \in \mathcal{T}_R$ for a message reception event by receiving party P . As with `TagSend`, $t \in \mathcal{T}_S \cup \mathcal{T}_R \cup \mathcal{T}_I$ is the last tag issued for P .
- $st', m, k_f, i \leftarrow \text{Rcv}(P, st, c)$ is defined as in Section 4.
- $G \leftarrow \text{Judge}(st_S, \rho)$ is defined as in Section 4.
- $P \leftarrow \text{JudgeReplay}(st_S, t, t')$ is a server procedure that takes as input two tags $t, t' \in \mathcal{T}_S \cup \mathcal{T}_R \cup \mathcal{T}_I$. It outputs a party $P \in \{0, 1\}$ if it determines that the tags constitute a replay attempt by P , or \perp if no replay attempt is detected.

Preventing fast-forwards. When sending a message, a client increments its send counter and appends to the causality data a server tag for the previous counter. This prevents the client from incrementing the counter too far into

SrvInit(): $k_{\text{mac}} \leftarrow \mathcal{K}; cs_0, cr_0, cs_1, cr_1 \leftarrow (0, 0, 0, 0)$ For $P \in \{0, 1\}$ $t_0^{(P)}.ack = (\text{Init}, P, \perp, 0, 0)$ $t_0^{(P)}.tag = \text{Tag}(k_{\text{mac}}, t_0^{(P)}.ack)$ return $\{k_{\text{mac}}, cs_0, cr_0, cs_1, cr_1\}, t_0^{(0)}, t_0^{(1)}$	TagRecv(st_S, P, c_f, t): if $\Pi(t) \neq P \vee \text{Verify}(k_{\text{mac}}, t.ack, t.tag) = 0$: return st_S, \perp $(cs_P, cr_P) \leftarrow t.ack.(cs, cr)$ $cr_P \leftarrow cr_P + 1, ack \leftarrow (R, \bar{P}, P, c_f, cs_P, cr_P)$ $t_r \leftarrow (ack, \text{Tag}(k_{\text{mac}}, ack))$ return st_S, t_r
TagSend(st_S, P, c_f, t): if $\Pi(t) \neq P \vee \text{Verify}(k_{\text{mac}}, t.ack, t.tag) = 0$: return st_S, \perp $(cs_P, cr_P) \leftarrow t.ack.(cs, cr)$ $cs_P \leftarrow cs_P + 1, ack \leftarrow (S, P, \bar{P}, c_f, cs_P, cr_P)$ $t_s \leftarrow (ack, \text{Tag}(k_{\text{mac}}, ack))$ return st_S, t_s	JudgeReplay(st_S, t, t'): if $\Pi(t) \neq \Pi(t')$ then return \perp $b \leftarrow (\text{Verify}(k_{\text{mac}}, t.ack, t.tag) = 1) \wedge$ $(\text{Verify}(k_{\text{mac}}, t'.ack, t'.tag) = 1) \wedge$ $((t.ack.cs + t.ack.cr) =$ $(t'.ack.cs + t'.ack.cr))$ if $b = 1$ then return $\Pi(t)$ else return \perp

Fig. 12. Pseudocode for our two-party transcript franking construction with out-sourced storage. Let $\Pi(t)$ be the sending party if t is a sending tag and the receiving party if t is a reception tag. The routines **Init**, **Snd**, **Rcv**, and **Judge** remain unchanged relative to the pseudocode given in Figure 7.

the future, resulting in what we term a *fast-forward* attack. Doing so would require the client to forge a MAC, since it would have to produce a valid server tag on a message the server had not tagged before. Syntactically, this means we modify **TagRecv** and **TagSend** to take in an additional input t , the latest tag issued by the server to party P . At the initialization of a conversation, the server provides each party P with a special starting tag $t_0^{(P)}$, which are additional outputs from **SrvInit**, where $t_0^{(P)}.ack = (\text{Init}, P, \perp, 0, 0)$. We write **TagSend**(st_S, P, c_f, t) and **TagRecv**(st_S, P, c_f, t). In our modified construction, the server first checks that t is a valid tag and obtains the initial values of the counters as $(cs_P, cr_P) \leftarrow t.ack.(cs, cr)$ instead of retrieving them from its own storage, for both **TagSend** and **TagRecv**. If the check on t fails, **TagRecv** and **TagSend** output \perp .

Preventing replays. If a client attempts to send a message with a repeated past counter, an honest recipient can report the repeated counters to the server as proof of sender misbehavior. Such a report provides resistance against rollback attacks for counters. For message reception tags, we follow the same exact approach as it applies to receive counters. We add a new procedure to the construction: $P \leftarrow \text{JudgeReplay}(st_S, t, t')$, where P is the party that attempted the replay. If the provided tags do not constitute proof of a replay, then the output is \perp . In our modified construction, the server returns P if $(\text{Verify}(k_{\text{mac}}, t.ack, t.tag) = 1)$, $(\text{Verify}(k_{\text{mac}}, t'.ack, t'.tag) = 1)$, $((t.ack.cs + t.ack.cr) = (t'.ack.cs + t'.ack.cr))$, and $(t.ack \neq t'.ack)$, where P is the sending party within a send acknowledgement or the receiving party of a reception acknowledgement – if these are not the same between t and t' , then we output \perp . To submit a false replay report

framing an honest party would require a client to forge a MAC. We term the submission of a false replay a *replay framing attack*, for which we provide a game-based definition in Appendix B.

Replay reportability. In addition to ensuring that honest clients cannot be falsely framed for attempting a replay, we must guarantee that actual replays by malicious clients are reportable. For a party P that has been issued a server tag $t \in \mathcal{T}_S \cup \mathcal{T}_R \cup \mathcal{T}_I$, that then generates t' and t'' by invoking **TagRecv** and/or **TagSend** with the same t given as the previous tag, **JudgeReplay**(st_S, t', t'') must output P . This holds for the construction in Figure 12 by the checks performed in **JudgeReplay**.

8 Related Work

Message franking. Message franking has been studied in various settings. Symmetric message franking provides a reporting solution when the platform houses the moderation endpoint for receiving reports, and when sender and recipient identities are known [1, 12, 15, 16]. Our own work is situated within this setting. Asymmetric message franking (AMF) generalizes to metadata private platforms and allows for third-party moderation [17, 33]. Recent work has also generalized AMFs to group messaging [18]. There are also proposed reporting mechanisms built from secret sharing [13]. All of these works consider message franking at the single-message level.

Causality in cryptographic protocols. Prior work has investigated incorporating causality in cryptographic channels [14, 25]. Notably, recent work by Chen and Fischlin has introduced stronger causality notions and shown how to combine them with message franking protocols [11]. However, as we have discussed, their message franking formalism does not meet our goals for transcript franking, due to its reliance on client-reported causality information and the inability for reporters to disclose their own sent messages. See Appendix A for more details. The distributed systems literature has long considered the problem of ordering events over communication networks via devising notions of logical time [19] and distributed snapshots [9].

Cryptographic Abuse Mitigation. In addition to enabling user-driven reporting, other cryptographic solutions have been proposed for targeting abuse in encrypted messaging. For messaging platforms that allow forwarding content, message trace-back is a cryptographic primitive that allows a platform to determine the origin of harmful content [17, 27, 35]. Message franking concerns user-driven content reporting. Recent work has also considered automated reporting for messages that match a list of known harmful content [8]. Such proposals have been met with strong criticism from privacy advocates. Follow-on work has attempted to navigate the privacy vs. moderation trade-off through added transparency and placing limitations on what content can be traced [6, 30]. Another line of work explores how cryptography can be used to aid with user-blocking [29, 34].

9 Conclusion

Existing treatments of message franking only consider how reporters can disclose individual messages that they receive. This is insufficient for including necessary context within reports. Our work provides definitions and constructions for transcript franking, an extension of message franking protocols that allows reporting sequences of messages with strong guarantees over message ordering and contiguity. We generalize our results to multi-party messaging and show how to securely outsource state to clients, allowing for more practical deployment. How our techniques can be generalized to asymmetric message franking, in order to be applicable to metadata-private and third-party moderation settings, remains an interesting open problem.

References

1. Messenger secret conversations technical whitepaper (2017), <https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf>
2. How do i know if my message was delivered or read? (2024), <https://support.signal.org/hc/en-us/articles/360007320751-How-do-I-know-if-my-message-was-delivered-or-read>
3. How to block and report someone (2024), https://faq.whatsapp.com/1142481766359885/?helpref=hc_fnav&cms_platform=web
4. How to check read receipts (2024), https://faq.whatsapp.com/665923838265756/?helpref=uf_share
5. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The messaging layer security (mls) protocol (2024), <https://datatracker.ietf.org/doc/rfc9420/>
6. Bartusek, J., Garg, S., Jain, A., Policharla, G.V.: End-to-end secure messaging with traceability only for illegal content. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology – EUROCRYPT 2023*. pp. 35–66. Springer Nature Switzerland, Cham (2023)
7. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) *Advances in Cryptology — CRYPTO ’96*. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
8. Bhowmick, A., Boneh, D., Myers, S., Talwar, K., Tarbe, K.: The apple psi system (2021), https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf
9. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.* **3**(1), 63–75 (Feb 1985). <https://doi.org/10.1145/214451.214456>, <https://doi.org/10.1145/214451.214456>
10. Chen, B., Dodis, Y., Ghosh, E., Goldin, E., Kesavan, B., Marcedone, A., Mou, M.E.: Rotatable zero knowledge sets. In: Agrawal, S., Lin, D. (eds.) *Advances in Cryptology – ASIACRYPT 2022*. pp. 547–580. Springer Nature Switzerland, Cham (2022)
11. Chen, S., Fischlin, M.: Integrating causality in messaging channels. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology – EUROCRYPT 2024*. pp. 251–282. Springer Nature Switzerland, Cham (2024)
12. Dodis, Y., Grubbs, P., Ristenpart, T., Woodage, J.: Fast message franking: From invisible salamanders to encryption. In: Shacham, H., Boldyreva, A. (eds.) *Advances in Cryptology – CRYPTO 2018*. pp. 155–186. Springer International Publishing, Cham (2018)
13. Eskandarian, S.: Abuse reporting for Metadata-Hiding communication based on secret sharing. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 3205–3221. USENIX Association, Philadelphia, PA (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/eskandarian>
14. Eugster, P., Marson, G.A., Poettering, B.: A cryptographic look at multi-party channels. In: 2018 IEEE 31st Computer Security Foundations Symposium (CSF). pp. 31–45 (2018). <https://doi.org/10.1109/CSF.2018.00010>
15. Grubbs, P., Lu, J., Ristenpart, T.: Message franking via committing authenticated encryption. In: Katz, J., Shacham, H. (eds.) *Advances in Cryptology – CRYPTO 2017*. pp. 66–97. Springer International Publishing, Cham (2017)

16. Huguenin-Dumittan, L., Leontiadis, I.: A message franking channel. In: Yu, Y., Yung, M. (eds.) *Information Security and Cryptology*. pp. 111–128. Springer International Publishing, Cham (2021)
17. Issa, R., Alhaddad, N., Varia, M.: Hecate: Abuse reporting in secure messengers with sealed sender. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2335–2352. USENIX Association, Boston, MA (Aug 2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/issa>
18. Lai, J., Zeng, G., Huang, Z., Yiu, S.M., Mu, X., Weng, J.: Asymmetric group message franking: Definitions and constructions. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology – EUROCRYPT 2023*. pp. 67–97. Springer Nature Switzerland, Cham (2023)
19. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* **21**(7), 558–565 (jul 1978). <https://doi.org/10.1145/359545.359563>, <https://doi.org/10.1145/359545.359563>
20. Len, J., Chase, M., Ghosh, E., Laine, K., Moreno, R.C.: OPTIKS: An optimized key transparency system. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 4355–4372. USENIX Association, Philadelphia, PA (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/len>
21. Leontiadis, I., Vaudenay, S.: Private message franking with after opening privacy. In: *International Conference on Information and Communications Security*. pp. 197–214. Springer (2023)
22. Machado, C., Kira, B., Narayanan, V., Kollanyi, B., Howard, P.: A study of misinformation in whatsapp groups with a focus on the brazilian presidential elections. In: *Companion Proceedings of The 2019 World Wide Web Conference*. p. 1013–1019. WWW '19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3308560.3316738>, <https://doi.org/10.1145/3308560.3316738>
23. Malvai, H., Kokoris-Kogias, L., Sonnino, A., Ghosh, E., Oztürk, E., Lewi, K., Lawlor, S.: Parakeet: Practical key transparency for end-to-end encrypted messaging. *Cryptology ePrint Archive*, Paper 2023/081 (2023). <https://doi.org/10.14722/ndss.2023.24545>, <https://eprint.iacr.org/2023/081>
24. Marlinspike, M.: Private group messaging (2014), <https://signal.org/blog/private-groups/>
25. Marson, G.A.: Real-World Aspects of Secure Channels: Fragmentation, Causality, and Forward Security. Ph.D. thesis, Technische Universität Darmstadt, Darmstadt (2017), <http://tuprints.ulb.tu-darmstadt.de/6021/>
26. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing key transparency to end users. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 383–398. USENIX Association, Washington, D.C. (Aug 2015), <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara>
27. Peale, C., Eskandarian, S., Boneh, D.: Secure complaint-enabled source-tracking for encrypted messaging. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. p. 1484–1506. CCS '21, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3460120.3484539>, <https://doi.org/10.1145/3460120.3484539>
28. Pfefferkorn, R.: Content-oblivious trust and safety techniques: Results from a survey of online service providers. *Journal of Online Trust and Safety* **1**(2) (Feb 2022). <https://doi.org/10.54501/jots.v1i2.14>, <https://tsjournal.org/index.php/jots/article/view/14>

29. Rosenberg, M., Maller, M., Miers, I.: Snarkblock: Federated anonymous block-listing from hidden common input aggregate proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 948–965 (2022). <https://doi.org/10.1109/SP46214.2022.9833656>
30. Scheffler, S., Kulshrestha, A., Mayer, J.: Public verification for private hash matching. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 253–273. IEEE Computer Society, Los Alamitos, CA, USA (may 2023). <https://doi.org/10.1109/SP46215.2023.10179349>, <https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179349>
31. Scheffler, S., Mayer, J.: SoK: Content Moderation for End-to-End Encryption. *Proceedings on Privacy Enhancing Technologies* **2023**(2), 403–429 (2023). <https://doi.org/10.56553/popets-2023-0060>
32. Tyagi, N., Fisch, B., Zitek, A., Bonneau, J., Tessaro, S.: Versa: Verifiable registries with efficient client audits from rsa authenticated dictionaries. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. p. 2793–2807. CCS '22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3548606.3560605>, <https://doi.org/10.1145/3548606.3560605>
33. Tyagi, N., Grubbs, P., Len, J., Miers, I., Ristenpart, T.: Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In: *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 18–22, 2019, *Proceedings, Part III*. p. 222–250. Springer-Verlag, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-030-26954-8_8, https://doi.org/10.1007/978-3-030-26954-8_8
34. Tyagi, N., Len, J., Miers, I., Ristenpart, T.: Orca: Blocklisting in Sender-Anonymous messaging. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 2299–2316. USENIX Association, Boston, MA (Aug 2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/tyagi>
35. Tyagi, N., Miers, I., Ristenpart, T.: Traceback for end-to-end encrypted messaging. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. p. 413–430. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). <https://doi.org/10.1145/3319535.3354243>, <https://doi.org/10.1145/3319535.3354243>
36. Wang, L., Wang, R., Williams-Ceci, S., Menda, S., Zhang, A.X.: "is reporting worth the sacrifice of revealing what i've sent?": Privacy considerations when reporting on End-to-End encrypted platforms. In: *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*. pp. 491–508. USENIX Association, Anaheim, CA (Aug 2023), <https://www.usenix.org/conference/soups2023/presentation/wang>
37. WhatsApp: Two billion users – connecting the world privately. WhatsApp Blog (2020), <https://blog.whatsapp.com/two-billion-users-connecting-the-world-privately>

A Comparison with Causality Preservation

Recent work by Chen and Fischlin considers the problem of assuring ordering integrity for messages within cryptographic channels, as well as extending this integrity to message franking [11]. They introduce a security notion called *causality preservation*, which captures the ability for two parties to recover a consistent causal dependency graph over the messages they exchange, even in the presence of a malicious network provider. To achieve causality preservation, clients self-report the order in which messages were sent and received via additional metadata. As a result, clients obtain a shared view of the partial ordering in which messages were sent and received. This partial ordering is captured by a causality graph, which we describe in Section 2. Our work additionally considers the problem of multi-message franking in the group setting while Chen and Fischlin focus on two-party messaging.

Overview of MFC with causality preservation. The causality metadata is incorporated into a message franking scheme, enabling reporting of this order in addition to the contents of the messages. Such context is valuable as the ordering and contiguity of messages sent within a conversation can heavily influence a moderator’s interpretation of the reported messages. To illustrate what this metadata looks like, we briefly recall the causal message franking channel MFCh_{cFB} presented in the Chen-Fischlin paper.

A sender attaches causal metadata consisting of a queue Q and an index i_R to each sent message. The queue Q contains the actions performed by the sender that have not yet been acknowledged by the recipient. Messages can be uniquely identified by their sending index and party. An index with a bar \bar{i} indicates a received message with sending index i . Actions recorded in Q simply consist of these indices. Once the recipient indicates the latest message it has received, the sender removes all elements from Q up to and including the one associated with that latest message. To communicate this, the index i_R , sent alongside Q , indicates the largest message index received by the sender from the other party, and the value i_S keeps track of the largest \bar{i}_R received from the other party. Intuitively, the message with sending index \bar{i}_R , now confirmed to have been received by the other party, contains all elements of Q up to and including the sending action \bar{i}_R , allowing those actions to be safely removed from Q . Despite the optimization that i_S and i_R enable, Q can grow arbitrarily large, and the overall bandwidth can increase in a quadratic manner if messages are not acknowledged. In single-message franking, a user reports only messages they have received. In causality-preserving message franking, the same is true, except these messages contain metadata about the order in which other messages have been sent and received.

Reporting self-sent message contents. Recall that the message franking channel formalism allows you to only report the messages and content received from the other party. This poses an issue for transcript franking since we may have to report messages sent by the reporter themselves. Currently, The Chen-Fischlin construction does not have a solution to this problem. Their reporting

formalism only allows reporting messages received from the other party. One workaround would be to require an interactive reporting process in which each party reports the messages of the other party, filling out the causality graph. Of course, this is less than desirable, especially in the case where an abusive party refuses to participate. In our solution, we propose a way in which clients can explicitly acknowledge reception of well-formed messages, thereby allowing the senders of those messages to independently report them.

Misbehavior by malicious clients. Via Q , clients self-report orderings of message sending and reception events. An issue here is that clients can self-report arbitrary such orderings, even ones that do not align with how messages were transmitted through the service provider. Figure 1 illustrates an example for which arbitrary self-reported messages orderings can enable malicious behavior.

Beyond the ability to deviate from the actual interleaving of messages, a malicious sender or reporter can lie about messages having been dropped or delivered out of order. For these reasons, relying on client-reported orderings is insufficient for reporting sequences of messages.

Attack on prior construction. The augmented Facebook message franking channel construction presented in [11], MFCh_{cFB} , allows clients to report message orders that do not align with the ground truth. We illustrate this via a simple attack that mirrors the idea presented in Figure 1. In order to provide a fair comparison, we first describe a natural lifting from the Chen-Fischlin notion for message franking channels to our setting. To instantiate a Chen-Fischlin-style message franking channel in our setting, we define a **TagRecv** function that simply returns \perp for the tag. The **Judge** function is defined in the natural way, by running **Extr** repeatedly for all messages involved in the report in order to construct the full graph.

The adversary \mathcal{A} chooses Party 0 as the malicious party (this is opposite of the attack presented in Section 1, but it applies in the same manner) and issues the following oracle calls. We specify the exact causal metadata that Party 0 embeds within the messages it sends to achieve this goal within each send call.

Sequence 1:

1. $c_1 \leftarrow \text{SendTag}(0, m_1; Q = (), i_R = -1)$
2. **RecvTag**(1, c_1)
3. $c_2 \leftarrow \text{SendTag}(1, m_2; Q = (\bar{1}), i_R = 1)$
4. **RecvTag**(0, c_2)
5. $c_3 \leftarrow \text{SendTag}(0, m_3; Q = (1), i_R = -1)$
6. **RecvTag**(1, c_3)
7. $c_4 \leftarrow \text{SendTag}(0, m_4; Q = (1, 2, \bar{1}), i_R = 1)$
8. **RecvTag**(1, c_4)

The adversary \mathcal{A} embeds causality metadata that suggests Party 0 having observed the ordering (m_1, m_3, m_2, m_4) . Meanwhile, Party 1 observes the ordering (m_1, m_2, m_3, m_4) , which also aligns with what Party 0 should have observed had it not deviated from the protocol. The ordering of **SendTag** and **RecvTag** calls differs from the ordering specified in the Q sent along with m_4 . Below, we

show the causality metadata Party 0 would have attached had it followed the protocol honestly:

Sequence 2:

1. $c_1 \leftarrow \text{SendTag}(0, m_1; Q = (), i_R = -1)$
2. $\text{RecvTag}(1, c_1)$
3. $c_2 \leftarrow \text{SendTag}(1, m_2; Q = (\bar{1}), i_R = 1)$
4. $\text{RecvTag}(0, c_2)$
5. $c_3 \leftarrow \text{SendTag}(0, m_3; Q = (\bar{1}), i_R = 1)$
6. $\text{RecvTag}(1, c_3)$
7. $c_4 \leftarrow \text{SendTag}(0, m_4; Q = (\bar{1}, 2), i_R = 1)$
8. $\text{RecvTag}(1, c_4)$

As a result, the graph recovered from `Extr` differs from the graph maintained by the security game. The adversary \mathcal{A} can issue a report, from either party, indicating the wrong causal ordering, winning with probability 1.

Impossibility result. The attack we just presented works because the server has no way to tag reception events. Since the message franking channel presented in [11] does not enable such tagging, it is impossible for any scheme within their model to satisfy our transcript integrity security notion. Intuitively, any scheme that doesn't enable the server to certify when messages are received requires the recipient party to self-report when reception events occur. In particular, the attack we just presented generalizes to any scheme that does not enable the server to tag reception events. Hence, we extend the message franking model to allow the server to tag reception events, via that `TagRecv` procedure.

Theorem 4. *Any message franking channel MFCh that does not tag message reception events does not satisfy transcript integrity.*

Proof. We consider Sequence 1 and Sequence 2 as defined above and note that they provide a scenario in which the sending events occur in the same order, but the reception events occur in a different order. Given that client input cannot be trusted to report the ordering of reception events, the `Judge` routine needs to somehow recover the causality graph given just information about the order in which messages were sent, which it can record upon invocation of `TagSend`. Since Sequences 1 and 2 have the same ordering of send operations ordering, but correspond to different causality graphs due to the difference in reception ordering, we see that it is impossible for `Judge` to distinguish between these two scenarios in general. In the transcript integrity game, the adversary can randomly choose to execute Sequence 1 or Sequence 2 with probability $1/2$. For any fixed `Judge` routine, the probability of outputting the correct causality graph is at most $1/2$. Hence, no scheme that fails to consider reception events can achieve our notion of transcript integrity. \square

$\mathbf{G}_{\text{TF}}^{\text{corr}}(\mathcal{A}):$ $k_{\text{Srv}} \leftarrow \$ \mathcal{K}, st_{\mathcal{A}}, k_{\text{Ch}} \leftarrow \$ \mathcal{A}(), \text{win} \leftarrow 0$ $st_S, t_0^{(0)}, t_0^{(1)} \leftarrow \$ \text{SrvInit}(k_{\text{Srv}})$ $st_0 \leftarrow \$ \text{Init}(0, k_{\text{Ch}}), st_1 \leftarrow \$ \text{Init}(1, k_{\text{Ch}})$ $t_0, t_1 \leftarrow t_0^{(0)}, t_0^{(1)}$ $\mathcal{R}_t, \mathcal{R}_r \leftarrow \{\}, \{\}$ $\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{\text{Ch}})$ return win $\mathcal{O}.\text{SendTag}(P, m):$ $(st_P, c) \leftarrow \$ \text{Snd}(P, st_P, m)$ $st_S, t_P \leftarrow \$ \text{TagSend}(st_S, P, c, c_f, t_P)$ $G \leftarrow G + (\mathbf{s}, P, m)$ Add (P, c, t_P) to \mathcal{R}_t return c, t_P	$\mathcal{O}.\text{RecvTag}(P, c, t_s):$ Assert $(\bar{P}, c, c, t_s) \in \mathcal{R}_t$ $st_P, m, k_f, i \leftarrow \text{Rcv}(P, st_P, c)$ if $m \neq \perp$ then $st_S, t_P \leftarrow \$ \text{TagRecv}(st_S, P, c_f, t_P)$ $G \leftarrow G + (\mathbf{R}, P, c, i)$ Add $(P, m, k_f, c_f, t_s, t_r)$ to \mathcal{R}_r else win $\leftarrow 1$ return m, k_f, t_s, t_P $\mathcal{O}.\text{Rep}(\rho):$ Assert $ \rho > 0 \ G' = \text{Judge}(st_S, \rho)$ if $\rho \subseteq \mathcal{R}_r \wedge ((G' = \perp) \vee (G' \not\subseteq G))$: win $\leftarrow 1$
--	---

Fig. 13. The security game for outsourced-storage transcript franking correctness.

B Security for Outsourced-storage Transcript Franking

In this section, we elaborate on the security analysis of our outsourced transcript franking scheme, which we introduced in Section 7. Correctness for outsourced transcript franking is captured by the game in Figure 13. We adapt the security notions for server-side-storage transcript franking to the outsourced setting and prove that our outsourced scheme achieves security.

Transcript integrity. In Figure 14, we present our transcript integrity definition for outsourced transcript franking. The goal of the adversary is the same as in the non-outsourced transcript integrity game, except now the adversary must submit valid server tags to generate new ones. Moreover, the adversary cannot replay tags in a way that is undetected by `JudgeReplay`. The advantage of an adversary \mathcal{A} in the outsourced storage transcript integrity game is

$$\text{Adv}_{\text{TF}}^{\text{o-tr-int}}(\mathcal{A}) = \Pr[\mathbf{G}_{\text{TF}}^{\text{o-tr-int}}(\mathcal{A}) = 1] .$$

Replay framing. In Figure 15, we present a security game for replay framing. Given two parties that honestly interact in the protocol, the adversary attempts to generate a replay that is detected by `JudgeReplay`, in effect framing an honest party for attempting to replay a server tag. The advantage of an adversary \mathcal{A} in the outsourced storage replay framing game is

$$\text{Adv}_{\text{TF}}^{\text{o-fr}}(\mathcal{A}) = \Pr[\mathbf{G}_{\text{TF}}^{\text{o-fr}}(\mathcal{A}) = 1] .$$

Security proofs. We now provide proofs for the transcript integrity and replay framing security of our outsourced storage transcript franking scheme. The following theorems establish the security of our outsourced construction.

$\mathbf{G}_{\text{TF}}^{\text{o-tr-int}}(\mathcal{A})$:	$\mathcal{O}.\text{SendTag}(P, c, t)$:	$\mathcal{O}.\text{RecvTag}(P, c, t_s, t)$:
$k_{\text{Srv}} \leftarrow \$ \mathcal{K}$; win $\leftarrow 0$	Assert for all $t_1, t_2 \in \mathcal{R}$,	Assert $(\bar{P}, c, t_s) \in \mathcal{R}_t$
$st_{\mathcal{A}}, k_{\text{Ch}} \leftarrow \$ \mathcal{A}()$	JudgeReplay(st_S, t_1, t_2) = 0	Assert for all $t_1, t_2 \in \mathcal{R}$,
$st_S, t_0^{(0)}, t_0^{(1)} \leftarrow \$$	$st_S, t_s \leftarrow \text{TagSend}(st_S, P, c, c_f, t)$	JudgeReplay(st_S, t_1, t_2) = 0
SrvInit(k_{Srv})	if $t_s = \perp$ then return \perp	$st_S, t_r \leftarrow \text{TagRecv}(st_S, P, c, c_f, t)$
$st_0 \leftarrow \$ \text{Init}(0, k_{\text{Ch}})$	$G \leftarrow G + (\text{S}, P)$	if $t_r = \perp$ then return \perp
$st_1 \leftarrow \$ \text{Init}(1, k_{\text{Ch}})$	Add (P, c, t_s) to \mathcal{R}_t , add t_s to \mathcal{R}	$G \leftarrow G + (\text{R}, P, c.i)$
$G, \mathcal{R}_r, \mathcal{R} \leftarrow \varepsilon, \{\}, \{\}$	return t_s	Add t_r to \mathcal{R}
$\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{\text{Ch}}, t_0^{(0)}, t_0^{(1)})$		return t_r
return win		
		$\mathcal{O}.\text{Rep}(\rho_1, \rho_2)$:
		Assert $ \rho_1 > 0$ and $ \rho_2 > 0$
		$G_1 \leftarrow \text{Judge}(st_S, \rho_1)$
		$G_2 \leftarrow \text{Judge}(st_S, \rho_2)$
		if $G_1 \neq \perp \wedge G_2 \neq \perp \wedge$
		$((G_1 \not\subseteq G) \vee (G_2 \not\subseteq G))$
		$\vee (G_1 \not\approx G_2)$:
		win $\leftarrow 1$

Fig. 14. The security game for outsourced-storage transcript integrity.

$\mathbf{G}_{\text{TF}}^{\text{o-fr}}(\mathcal{A})$:	$\mathcal{O}.\text{RecvTag}(P, c, t_s)$:	$\mathcal{O}.\text{SendTag}(P, c_f)$:
$k_{\text{Srv}} \leftarrow \$ \mathcal{K}$; win $\leftarrow 0$	Assert $(\bar{P}, c, t_s) \in \mathcal{R}_t$	$st_S, t_P \leftarrow \text{TagSend}(st_S, P, c, c_f, t_P)$
$st_{\mathcal{A}}, k_{\text{Ch}} \leftarrow \$ \mathcal{A}()$	Assert $c \notin \mathcal{R}$	if $t_P = \perp$ then return \perp
$st_S, t_0^{(0)}, t_0^{(1)} \leftarrow \$$	$st_S, t_P \leftarrow \text{TagRecv}(st_S, P, c, c_f, t_P)$	Add (P, c, t_P) to \mathcal{R}_t return t_P
SrvInit(k_{Srv})	if $t_P = \perp$ then return \perp	
$st_0 \leftarrow \$ \text{Init}(0, k_{\text{Ch}})$	Add c to \mathcal{R}	$\mathcal{O}.\text{RepReplay}(t, t')$:
$st_1 \leftarrow \$ \text{Init}(1, k_{\text{Ch}})$	return t_P	$P \leftarrow \text{JudgeReplay}(st_S, t, t')$
$t_0, t_1 \leftarrow t_0^{(0)}, t_0^{(1)}$		if $P \neq \perp$:
$\mathcal{R}_r, \mathcal{R}, \mathcal{R}_t \leftarrow \{\}, \{\}, \{\}$		win $\leftarrow 1$
$\mathcal{A}^{\mathcal{O}}(st_{\mathcal{A}}, k_{\text{Ch}}, t_0^{(0)}, t_0^{(1)})$		
return win		

Fig. 15. The security game for outsourced storage replay framing.

Theorem 5. Let TF be our outsourced-storage transcript franking scheme in Figure 12. Let \mathcal{A} be a transcript integrity adversary against TF . Then we give EUF-CMA adversaries \mathcal{B} and \mathcal{C} , and a V-Bind adversary \mathcal{D} , such that

$$\text{Adv}_{\text{TF}}^{\text{o-tr-int}}(\mathcal{A}) \leq \text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) + \text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{C}) + \text{Adv}_{\text{CS}}^{\text{v-bind}}(\mathcal{D}).$$

Adversaries \mathcal{B} , \mathcal{C} , and \mathcal{D} run in time that of \mathcal{A} plus a small overhead.

Proof. We proceed via a sequence of game hops. Define \mathbf{G}_0 to be the same as $\mathbf{G}_{\text{TF}}^{\text{o-tr-int}}$ with the additional bookkeeping as defined in the proof of Theorem 1. Let \mathbf{G}_1 be the same, except we abort if at any point $G.(cs_P, cr_P) \neq st_S.(cs_P, cr_P)$. Let F_1 denote this event. We have $|\Pr[\mathbf{G}_0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathbf{G}_1(\mathcal{A}) \Rightarrow 1]| \leq \Pr[F_1]$, and we will construct \mathcal{B} such that $\text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) = \Pr[F_1]$. We have that \mathcal{B} simulates \mathbf{G}_0 to \mathcal{A} while routing Tag and Verify calls to its challenger

oracles. Observe that the only way for F_1 to occur is for \mathcal{A} to produce a $t^* \notin \mathcal{R}$, which means that t^* was never queried to the **Tag** oracle, hence \mathcal{B} outputs it as a forgery.

The rest of the proof proceeds similarly to the proof of Theorem 1. \square

Theorem 6. *Let **TF** be our outsourced-storage transcript franking scheme in Figure 12. Let \mathcal{A} be a replay framing adversary against **TF**. Then we give an EUF-CMA adversary \mathcal{B} such that*

$$\text{Adv}_{\text{TF}}^{\text{o-fr}}(\mathcal{A}) \leq \text{Adv}_{\text{MAC}}^{\text{euf-cma}}(\mathcal{B}) .$$

Adversary \mathcal{B} runs in time that of \mathcal{A} plus a small overhead.

Proof. Our adversary \mathcal{B} perfectly simulates $\mathbf{G}_{\text{TF}}^{\text{o-fr}}$ to \mathcal{A} while routing calls to **Tag** and **Verify** to its own challenger oracles. For any two tags t_1, t_2 output by the **Tag** oracle, we must have that $\text{JudgeReplay}(st_S, t_1, t_2) = \perp$, since these tags are the output of the honest **TagRecv** and **TagSend** procedures. Therefore, if \mathcal{A} wins, it must have produced a pair (t, t') where at least one of these tags was not output by the **Tag** challenger oracle. Let t^* be that tag. The adversary \mathcal{B} outputs t^* and wins with the same probability that adversary \mathcal{A} wins. \square

Group outsourced transcript franking. The construction and analysis we provide for outsourced two-party transcript franking generalizes naturally to the N -party group messaging setting. We sketch the necessary modifications here. As with the two-party outsourced setting, we have that **SrvInit**, in addition to st_S , outputs a list of initial tags $t_0^{(0)}, \dots, t_0^{(N-1)}$. The server-side tagging procedures **TagSend** and **TagRecv** accept an additional argument t for the previous tag issued to a party. We include an additional procedure $\text{JudgeReplay}(st_S, t, t')$, which outputs a party P if (t, t') constitute a replay attack by P , or \perp otherwise. We outline our outsourced group transcript franking construction in Figure 16.

<p><u>SrvInit(N):</u> $k_{\text{mac}} \leftarrow \\\mathcal{K} For $P \in [N]$ $t_0^{(P)}. \text{ack} = (\text{Init}, P, \perp, 0, 0)$ $t_0^{(P)}. \text{tag} = \text{Tag}(k_{\text{mac}}, t_0^{(P)}. \text{ack})$ $cs_i, cr_i \leftarrow 0, 0$ return $\{k_{\text{mac}}\} \cup \{cs_i, cr_i\}_{i \in [N]}, \{t_0^{(i)}\}_{i \in [N]}$</p> <p><u>TagSend($st_S, P, c_f, t$):</u> if $\Pi(t) \neq P \vee \text{Verify}(k_{\text{mac}}, t. \text{ack}, t. \text{tag}) = 0$: return st_S, \perp $(cs_P, cr_P) \leftarrow t. \text{ack}.(cs, cr)$ $cs_P \leftarrow cs_P + 1, \text{ack} \leftarrow (S, P, c_f, cs_P, cr_P)$ $t_s \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_s</p>	<p><u>TagRecv(st_S, P_R, P_S, c_f, t):</u> if $\Pi(t) \neq P_R \vee \text{Verify}(k_{\text{mac}}, t. \text{ack}, t. \text{tag}) = 0$: return st_S, \perp $(cs_{P_R}, cr_{P_R}) \leftarrow t. \text{ack}.(cs, cr)$ $cr_{P_R} \leftarrow cr_{P_R} + 1$ $\text{ack} \leftarrow (R, P_S, P_R, c_f, cs_{P_R}, cr_{P_R})$ $t_r \leftarrow (\text{ack}, \text{Tag}(k_{\text{mac}}, \text{ack}))$ return st_S, t_r</p> <p><u>JudgeReplay(st_S, t, t'):</u> if $\Pi(t) \neq \Pi(t')$ then return \perp $b \leftarrow (\text{Verify}(k_{\text{mac}}, t. \text{ack}, t. \text{tag}) = 1) \wedge$ $(\text{Verify}(k_{\text{mac}}, t'. \text{ack}, t'. \text{tag}) = 1) \wedge$ $((t. \text{ack}.cs + t. \text{ack}.cr) =$ $(t'. \text{ack}.cs + t'. \text{ack}.cr))$ if $b = 1$ then return $\Pi(t)$ else return \perp</p>
--	--

Fig. 16. Pseudocode for our N -party transcript franking construction with outsourced storage. Let $\Pi(t)$ be the sending party if t is a sending tag and the receiving party if t is a reception tag.