Hecate: Abuse Reporting in Secure Messengers with Sealed Sender

Rawane Issa, Nicolas Alhaddad, and Mayank Varia
Boston University[*], {ra1issa,nhaddad,varia}@bu.edu

Abstract

End-to-end encryption provides strong privacy protections to billions of people, but it also complicates efforts to moderate content that can seriously harm people. To address this concern, Tyagi et al. [CRYPTO 2019] introduced the concept of *asymmetric message franking* (AMF) so that people can report abusive content to a moderator, while otherwise retaining end-to-end privacy by default and compatibility with anonymous communication systems like Signal's sealed sender.

In this work, we provide a new construction for asymmetric message franking called Hecate that is faster, more secure, and introduces additional functionality compared to Tyagi et al. First, our construction uses fewer invocations of standardized crypto primitives and operates in the plain model. Second, on top of AMF's accountability and deniability requirements, we also add forward and backward secrecy. Third, we combine AMF with source tracing, another approach to content moderation that has previously been considered only in the setting of non-anonymous networks. Source tracing allows for messages to be forwarded, and a report only identifies the original source who created a message. To provide anonymity for senders and forwarders, we introduce a model of *AMF with preprocessing* whereby every client authenticates with the moderator out-of-band to receive a token that they later consume when sending a message anonymously.

1 Introduction

End-to-end encrypted messaging systems like Facebook Messenger, Signal, Telegram, Viber, and WhatsApp are used by billions of people [77] due to their powerful combination of cryptographic protections and ease of use. The security guarantees provided by encrypted messengers are both varied and valuable [74]: confidentiality and integrity from authenticated key exchange [16, 19, 51], deniability from the use of symmetric authenticated encryption [15, 28, 38], and forward and backward security via key evolution (aka ratcheting) [24, 39]. However, these very security guarantees complicate efforts by secure messaging platforms to investigate reports of abuse or disinformation campaigns, which can have serious consequences for individuals and collective society [10, 30, 69, 71, 76].

To address these concerns, the security research community has developed three methods to augment end-to-end messengers with privacy-respecting technologies to assist with content moderation: message franking, source tracing, and automated identification. First, *message franking* [28, 31, 38, 52, 72] allows recipients to manually report abusive messages with assurance that unreported messages retain all guarantees of secure messengers, and reported messages are both accountable (the moderator correctly identifies the message's sender) and deniable (the moderator cannot prove this fact to anybody else). Second, *source tracing* [57, 73] allows the moderator to pinpoint the original source of a viral message rather than the person who forwarded the message to the eventual reporter. Finally, *automated identification* [11, 50, 53] proactively matches messages against a moderator-provided list of messages using a private (approximate) set membership test, with possible interventions like rate-limiting or warning labels in case of a match [70].

This work contributes a new construction called Hecate that simplifies, strengthens, and unifies the first two content moderation techniques: asymmetric message franking and source tracing. We do not consider

[*] This material is based upon work supported by the National Science Foundation under Grants No. 1718135, 1739000, 1801564, 1915763, and 1931714, by the DARPA SIEVE program under Agreement No. HR00112020021, and by DARPA and the Naval Information Warfare Center (NIWC) under Contract No. N66001-15-C-4071.

automated identification, focusing instead on abuse reporting schemes that empower the people who receive messages to choose the action they wish to take [47, 58]. To provide context for our work, we describe the nascent space of message franking and source tracing in more detail before explaining our improvements.

1.1 Prior work

There exists a long line of research into the security of end-to-end encrypted messaging systems (EEMS) at both the protocol design and software implementation layers (e.g., [4, 9, 13, 22, 46]). Our work relies on these analyses in order to treat the underlying messaging protocol in a black-box manner and abstract away its details, so that we may focus on the additions provided by content moderation protocols.

Message franking constructions involve four parties: a sender and receiver of a message, plus the platform providing the secure messaging service and a moderator who acts on abuse reports (see Figure 1). *Symmetric* message franking protocols are limited to the setting in which the platform and moderator are the same entity and have sufficient network-level visibility to pinpoint the sender of each message. At a high level, these constructions operate as follows: when a sender submits a ciphertext corresponding to the message m, the platform signs an attestation binding the sender's identity to a compact commitment $\mathsf{com}(\mathsf{m})$ provided by the sender in the clear. The receiver also sees this commitment (e.g., if it is part of a robust encryption scheme [1, 32, 33]) and can check whether it is correct, dropping the packet if it is malformed. Subsequently, the receiver can report the message as abusive by opening all [28, 31, 38] or part [52] of the commitment so that the platform can determine whether the message is abusive and take appropriate action.

The work of Tyagi et al. [72], which is the starting point for this paper and which we will henceforth refer to as TGLMR, introduces the notion of *asymmetric message franking* (AMF) that removes the limitations from above. Specifically, AMF can operate even when using Signal's sealed sender [65] or an anonymous communication system (e.g., [3, 25, 27, 75]) that hides the identity of the sender or receiver from the platform.

Inspired by designated-verifier signatures [44, 62], the TGLMR construction requires the sender to make a Diffie-Hellman tuple $\langle g, g^{\mathsf{sk}_{\mathtt{src}}}, g^{\mathsf{sk}_{\mathtt{mod}}}, g^{\mathsf{sk}_{\mathtt{src}} \cdot \mathsf{sk}_{\mathtt{mod}}} \rangle$ involving the moderator's secret key and her own, as well as a non-interactive zero knowledge proof that the tuple is well-formed. TGLMR achieves accountability and deniability for the sender, but doesn't provide forward and backward security due to the use of long-lived secret keys. Moreover, it is complex and expensive to implement (see §6), and requires a non-falsifiable knowledge of exponent assumption in the random oracle model. Finally, TGLMR does not easily generalize to more complex conversation graphs that allow for forwarding.

Another line of research investigates the ability for the moderator to trace the original source of messages that might have been forwarded several times within an EEMS. Tyagi, Miers, and Ristenpart [73] began this line of study with their Traceback scheme. This protocol reveals to the moderator the entire path from the original source to the reporter while formally guaranteeing notions of confidentiality and accountability to members of that path. However, their scheme imposes computational and storage burdens on the moderator; the required storage is proportional to the number of messages eligible to be traced. Moreover, it may not be desirable to reveal the entire forwarding path.

Two recent works provide *source tracing*, identifying only the original source of a reported message. First, Peale, Eskandarian, and Boneh [57] contribute a source tracing construction that inherits most security properties from the underlying EEMS (see Table 1). Using more expensive crypto operations, the stronger variant of their construction is the only one to date to achieve tree unlinkability — namely, that a receiver who gets the same message twice cannot tell if they originate from the same or different sources. Second, the FACTS scheme by Liu et al. [53] provides source tracing along with a threshold reporting scheme so that the moderator only learns when sufficiently many complaints have been lodged against an abusive source client. However, none of these traceback or source tracing schemes [53, 57, 73] considers backward security as part of their security model. Also, none of them provides full anonymity of senders and receivers from the EEMS platform or moderator; FACTS is compatible with a network that provides one-sided anonymity, but it requires senders to identify themselves and request tokens from the moderator on the fly whenever they wish to send a message.

This leaves the following open question:

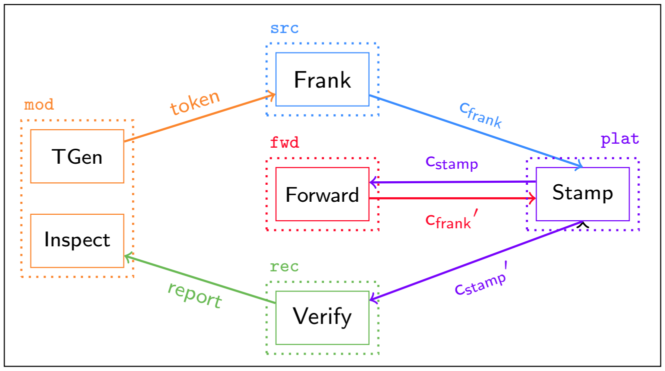


Figure 1: Diagram of Hecate's data flow for a message m, from the source (top) to the forwarder (middle) and then to a reporting receiver (bottom). The commands match our definition of AMF with preprocessing (Def. 11), and the variables token, c_{frank}, c_{stamp}, and report are defined in Fig. 2.

> *Can we design a protocol that simultaneously provides asymmetric message franking (AMF) and source tracing, achieves forward and backward security, maintains anonymity of senders and receivers to the extent provided by the underlying EEMS network, and only makes black-box use of standardized cryptography in the plain model?*

In this work, we answer the question in the affirmative.

1.2 Our contributions

In this work, we provide a new definition and construction for asymmetric message franking (AMF) that is more general, more secure, and faster than previous work. To achieve this goal, we revisit the decision by TGLMR [72] to "restrict[] attention to non-interactive schemes for which franking, verification, and judging requires sending just a single message." On its face, this restriction seems natural because end-to-end encrypted messengers are designed to work asynchronously in situations with limited network connectivity, so one-round (online) protocols are desirable. However, this restriction also appears to direct the solution space toward expensive crypto tools like designated-verifier signatures and zero knowledge proofs.

Our core insight is to introduce an *AMF with preprocessing* model as shown in Fig. 1. As before, the online work of message franking and transmission requires only one round of communication from the source to platform to receiver. Beforehand, we allow the source and moderator to engage in a single data-independent preprocessing interaction to produce *tokens* that can be consumed during the online phase. Preprocessing can be batched to produce many tokens at once, it can be performed during off-peak hours when the source's device is connected to power and wifi, and it should should be performed in advance rather than on the fly in order to avoid network-level traffic linking attacks [55]. As with MPC [7] or PIR [8], we show that adding a preprocessing round to AMF allows for more efficient protocols, and in particular allows us to answer the open question from above.

Concretely, we contribute an AMF scheme called Hecate. Our construction leverages the fact that, with preprocessing, the communication path of reported messages begins and ends with the content moderator. Ergo, we can use techniques from (faster) symmetric message franking whereby the moderator can prepare a token (e.g., a symmetric encryption of the source's identity) that is only intelligible to its future self. The token is passed through the sender → platform → receiver communication flow of an EEMS; that said, end-to-end encryption prevents the platform from viewing the token.

Hecate also supports *source tracing*, in which receivers can forward messages along with their corresponding tokens. Any recipient can choose to report an abusive message; this only requires sending one communication to the moderator.

A big challenge in our construction is to combine message forwarding with our AMF *backward security*

Construction	Features					Security Guarantees							
	Abuse reporting	Message forwarding	Source tracing	Trace info	Threshold report	Confidentiality	Anonymity	Tree unlinkability	Deniability	Forward security	Backward security	Unforgeability	Accountability
Signal	○	●	×	×	×	●	◐	●	●	●	●	●	×
Tyagi et al. [72]	●	○	×	src	○	●	●	×	●	○	○	●	●
Traceback [73]	●	●	●	path	○	●	○	○	◕	○	○	●	●
FACTS [53]	●	●	●	src	●	●	◐	○	◕	◕	○	●	●
Peale et al. [57]	●	●	●	src	○	●	○	●	●	◕	○	●	●
Hecate (*this work*)	●	●	●	src	○	●	●	○	●	●	●	●	●

●: fully provided, ◕ : provided but not proven, ◐: partially provided, ○: not provided, ×: not applicable

Table 1: A comparison of features and security properties provided by the Signal EEMS protocol as well as several abuse reporting constructions. Security properties are described in §2.2 and §5. For the anonymity column, ◐ refers to providing anonymity at the level of Signal's sealed sender [65].

requirement, which states that an attacker who previously (but no longer) controlled a source's device cannot blame the source for new messages. To our knowledge, this work is the first one to consider and formalize backward security within AMF. As we will discuss in more detail in §2, the challenge in combining AMF with backward security stems from the fact that immediate receivers of the message from the original source can rely on the backward security of the underlying encrypted messaging protocol to know that they're speaking with the source rather than the attacker, whereas indirect receivers cannot.

In summary, we make four contributions in this work.

- We rigorously define AMF with preprocessing (§3). We generalize the definition from TGLMR, formalize forward and backward security, and add source tracing.
- We provide a construction called Hecate (§4). It requires only a few black-box calls to standard crypto primitives.
- We formalize and prove (§5) that Hecate achieves all of the security guarantees shown in Table 1.
- We implement Hecate (§6) and integrate it into a Signal client. We show that Hecate's performance compares favorably to prior work and is imperceptible in practice.

Before continuing, we wish to stress that any decision to use content moderation within end-to-end encrypted messengers requires weighing all of its potential benefits and risks, including the limitations of Hecate and prior works (see §7.2), and the risk of abuse by or coercion of the moderator. This is a complex policy question whose discussion should involve computer scientists, but not only computer scientists. We take no stance on the policy question in this work; instead, we observe that these policy discussions are already ongoing [2, 17, 60] and that a sub-optimal understanding of the technological possibilities may push a service provider or nation-state policymakers toward a worse policy decision. We undertake this research in order to demonstrate the feasibility of alternatives to blunt privacy-inhibiting legislation.

2 Overview

In this section, we describe our objectives for an asymmetric message franking (AMF) system. We begin by describing the setting and threat model for our work, and then we provide a high-level description of the security requirements and a brief description of how our Hecate protocol will achieve them.

2.1 Setting and Threat Model

In this work, we consider an EEMS that might contain network-level anonymity protections such as Signal's sealed sender [65] or Tor [27]. We focus on two party point-to-point communication; that said, our techniques translate directly to Signal's group messaging protocol as described in §7.1.

Within the context of any single message transmission, we refer to the participating clients using the following terminology: the *source* who initially produced the message within the messaging platform, the *receivers* who receive the message and can optionally decide to report it (in which case we call them a *reporter*) and the *forwarders* who are receivers that decide to send the message along to others. All clients only possess the computational power of a regular phone.

Due to forwarding, each message's communication graph has the structure of a tree rooted at the source. A client can have different roles in the communication trees of different messages.

In addition to the messenger clients, our model contains two (possibly separate, and more computationally powerful) entities that everyone can communicate with: (1) the *platform* that provides the messaging service, and (2) the content *moderator*. We consider the platform and moderator as possibly separate so that our model can capture settings where a platform outsources moderation tasks to other, more qualified organizations (e.g., Facebook's oversight board [56]).

Generally, the parties in the system view all other parties as potentially malicious and colluding together. Every party wants confidentiality and integrity to the strongest extent possible, even if some or all of their counterparty, the platform, and the moderator are colluding against them. In particular, we wish to retain all of the security goals that end-to-end encrypted messengers provide, as detailed in §2.2 and §5.

In this work, a malicious attacker has the power to compromise one or more parties, in which case it can observe these parties' local state (e.g., cryptographic keys) and run arbitrary software for the duration of their control of a victim's machine. A semi-honest party, by contrast, is assumed to perform all actions honestly, and the only objective against such a party is data minimization. We presume that the software implementing the encrypted messenger faithfully reproduces the intended specification so that the adversary cannot control the behavior of honest parties. Put another way, supply chain attacks and formal verification are out of scope of this work.

The parties' relationship toward the moderator is more subtle. The moderator and platform view each other as semi-honest; looking ahead to our Hecate construction, the moderator trusts the accuracy of any timestamp applied by the platform but it need not trust the platform for any other purpose. Clients have a choice: if they view the moderator as malicious then they must be assured of limits to the moderator's power, or if they view the moderator as semi-honest then they must be assured that the moderator can perform its role.

The objective of holding senders accountable for reported messages creates a tension with the security goals of end-to-end encrypted messengers. In particular, clients no longer receive confidentiality, deniability, anonymity, or other privacy guarantees for reported messages. Moreover, an AMF scheme imposes a limit on forward security, because messages sent in the past now can be revealed to the moderator in the present. Our objective is to ensure security up to these fundamental limits. We emphasize that even if the moderator is malicious and colluding with some clients, *all of the security guarantees for end-to-end encrypted messaging continue to hold for all unreported messages communicated between non-colluding clients.* Moreover, even for reported messages, security holds against all other parties who are not colluding with the moderator.

Another tension exists between content moderation and network anonymity. For example, *sealed sender* is a feature introduced by the Signal protocol to hide the identity of the sender from the platform. It offers sender confidentiality and minimizes the amount of metadata stored by the platform. But if the sender can deny ever sending a message, then can we hold anyone responsible for sending an abusive message? TGLMR [72] resolved this dilemma using zero-knowledge signatures; in this paper we contribute an alternative construction based solely on black-box use of standard crypto primitives.

2.2 Security goals

In an asymmetric message franking scheme, we aim to provide all of the security and privacy goals of encrypted messengers [22, 74]. Some EEMS goals (cf. §3.2) are already consistent with content moderation, in which case AMF constructions can use these properties and must ensure that they don't weaken them. To give a concrete example for our Hecate protocol: we use the EEMS as a black box, and we will take advantage of the receiver's ability to authenticate the sender's identity. On the other hand, some security goals are not fully compatible with content moderation, in which case we aim to make the smallest modification possible.

Below, we describe each security goal from Table 1 and highlight the extent to which it is impacted by content moderation. These security goals apply to all clients who construct properly formatted messages that adhere to the encrypted messaging protocol, whether or not their messages are subsequently reported. That is: even though malicious parties in a crypto protocol receive no security guarantees, the mereact of sending a reported message does not render a client malicious.

- *Confidentiality.* Anyone not involved in the creation, forwarding, or reporting of a message m must not learn anything about m except an upper bound on its size.
- *Anonymity.* The AMF scheme should not allow the platform, receiver, or moderator to learn anything about the source and forwarding path of a message beyond what they would learn from the underlying EEMS or a report.
- *Deniability.* If the moderator is honest, then every user should be able to deny a claim about the message contents made by any adversary (even other recipients of this message). If the moderator is malicious or loses control of their secret key material, any user can deny the contents of a non-reported message. Reported messages on the other hand are deniable to anyone other than the moderator.
- *Forward security.* Adversaries that compromise user's state in the present should not be able to deduce anything about messages exchanged in the past. This goal does not apply to messages that happen to remain on the phone in the present, which can still be read and reported.
- *Backward security.* Once a client recovers from a compromise event, then the compromised state becomes 'useless' after a short recovery period. In particular, the adversary cannot send a new message that (if reported) would cause the moderator to blame the victim client.
- *Unforgeability.* The adversary cannot send a message that appears to be sent by another party. An honest receiver will reject any malformed or tampered messages.
- *Accountability.* If a message passes a receiver's verification check and is subsequently reported, the moderator will trace it back to its original source. That is: nobody can falsely accuse someone who wasn't the source of a message, and the true source cannot evade detection and yet also have the message verified by the receiver.

2.3 Protocol Overview

In this section, we give a high level overview of our Hecate protocol in two stages (with and without message forwarding) and explain informally how it satisfies our security goals.

Hecate without forwarding. At a high level, our Hecate construction can be thought of as an interactive variant of designated-verifier signatures. Given a message m, the source constructs a 2-out-of-2 secret sharing, say $\mathsf{H}(\mathsf{m}) = \mathsf{x}_1 \oplus \mathsf{x}_2$. In Hecate, the moderator binds x_1 to the source's identity (which on its own reveals nothing about m), and then the source binds x_1 to x_2 without using any long-lived keys.

As shown in Fig. 2: since one of the two shares can be sampled even before m is known, during preprocessing the moderator selects x_1 as an encryption of the source's identity (which appears random to everyone else), samples an ephemeral digital signature keypair $(\mathsf{sk}_\mathsf{e}, \mathsf{pk}_\mathsf{e})$, and signs both x_1 and pk_e. The tuple of x_1, pk_e, and their signature constitute the preprocessing token token. During the online phase, the source uses the ephemeral key sk_e to sign x_2; we refer to the pair of x_2 and its signature as another token token.

The source provides both tokens to the receiver within the payload of an ordinary Signal packet, as shown in Fig. 2; ignore the other elements of the franked ciphertext $\mathsf{c}_{\mathsf{frank}}$ for now. Any receiver can check

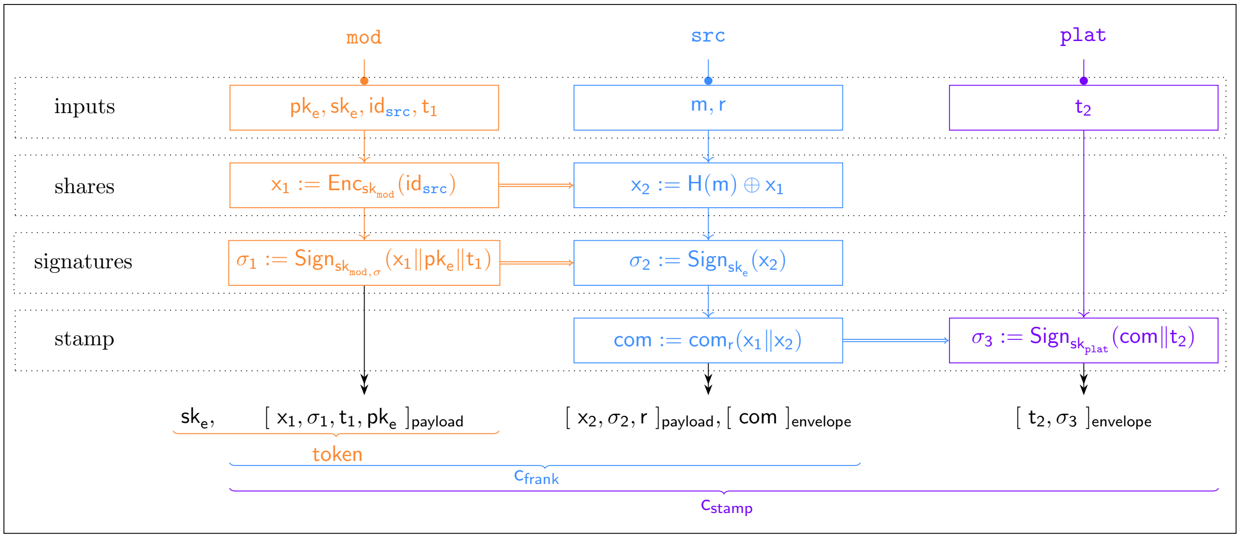

mod
src
plat
inputs
$\mathsf{pk_e, sk_e, id_{src}, t_1}$
$\mathsf{m, r}$
$\mathsf{t_2}$
shares
$\mathsf{x_1 := Enc_{sk_{mod}}(id_{src})}$
$\mathsf{x_2 := H(m) \oplus x_1}$
signatures
$\sigma_1 := \mathsf{Sign}_{\mathsf{sk}_{\mathsf{mod},\sigma}}(\mathsf{x_1 \| pk_e \| t_1})$
$\sigma_2 := \mathsf{Sign}_{\mathsf{sk_e}}(\mathsf{x_2})$
stamp
$\mathsf{com := com_r(x_1 \| x_2)}$
$\sigma_3 := \mathsf{Sign}_{\mathsf{sk_{plat}}}(\mathsf{com \| t_2})$
$\mathsf{sk_e}$, $[\ \mathsf{x_1}, \sigma_1, \mathsf{t_1}, \mathsf{pk_e}\ ]_{\mathsf{payload}}$
token
$[\ \mathsf{x_2}, \sigma_2, \mathsf{r}\ ]_{\mathsf{payload}}, [\ \mathsf{com}\ ]_{\mathsf{envelope}}$
$[\ \mathsf{t_2}, \sigma_3\ ]_{\mathsf{envelope}}$
$\mathsf{c_{frank}}$
$\mathsf{c_{stamp}}$

Figure 2: The construction of the different parts of a franked cipher. The outputs of the diagram correspond to each party's contribution to the eventual stamped cipher $\mathsf{c_{stamp}}$. The source constructs the franked cipher $\mathsf{c_{frank}}$ using a token provided by the moderator during preprocessing. We denote by $\mathsf{payload}$ and $\mathsf{envelope}$ two different parts of the ciphertext as defined in the Signal sealed sender protocol [65]; the platform and receiver can read the $\mathsf{envelope}$ whereas only the receiver can read the $\mathsf{payload}$.

on its own whether the signatures are valid and the underlying values $\mathsf{x_1}$ and $\mathsf{x_2}$ combine to form the real message m that the receiver also gets from the underlying Signal communication; if verification fails, then the message is malformed, so it is dropped without displaying on the receiver's device. If a verified message is later reported, the two tokens together will convince the moderator that the source was the originator of message m.

Achieving our security goals. Many of our security guarantees follow directly from the corresponding property of the underlying EEMS, so we focus on the most challenging goals here. Hecate provides accountability for the same reason as symmetric messaging franking schemes: the moderator created an authenticated encryption of the source's identity for its future self. Forward security holds because ephemeral signing keys $\mathsf{sk_e}$ from the past were deleted before a compromise event in the present. Deniability can be shown in two parts: if the moderator's keys are breached then anyone can produce signatures for any choices of $\mathsf{x_1}$ and $\mathsf{x_2}$, and otherwise the source's identity is hidden within the encrypted token so anyone could have 'forged' signatures of an $(\mathsf{x_1}, \mathsf{x_2})$ pair using her own tokens rather than those of the real source.

Backward (or post-compromise [24]) security is more challenging to address, and it is worth pausing for a moment to discuss what this guarantee means in the context of content moderation. If an adversary corrupts the source's phone, it *can* produce messages whose reports blame the source; this is inevitable. Our goal is to ensure that once the source recovers control of her phone, then (perhaps after a short delay δ) any new message produced by the adversary cannot implicate the honest source. To provide this guarantee within Hecate, the moderator includes a timestamp within its attestation to $\mathsf{x_1}$, and receivers drop any message where this timestamp is too old. This ensures that an adversary cannot continue to use pre-processing tokens after the compromise event.

Hecate with message forwarding. Next, we allow forwarding of messages and consider source tracing, in which the moderator should identify only the original source of a reported message. For the most part, our construction is already amenable to source tracing: a forwarder can simply include the original source's tokens within a forwarded message rather than generating new tokens that would implicate herself. However, our timestamp-based solution to backward security now fails because the age of x_1 is insufficient to determine whether the original source had control of her cryptographic keys at the moment that the *original message* was sent (as opposed to the time of the forwarding).

As shown in Figure 1, we solve this problem by appending a timestamp time as the data traverses through the platform, so that receivers can check whether the timestamps on the preprocessing and sending stages are close in time to each other. This is sufficient because the original source's message x_2 inherits backward security from the underlying encrypted messenger, so to verify backward security it suffices to verify whether the pre-processed token (which contains the identity of the source to blame) was produced close in time to the message transmission. As shown on the bottom of Figure 1, timestamps for forwarded messages are disregarded; future recipients only care about the timestamp from original source.

It only remains to bind the source timestamp to the message, so that it cannot be tampered later. Note that we cannot reveal x_2 to the platform, or else the platform and moderator together could recover the content of messages. Blind signatures are a possible solution to allow the platform to timestamp-and-sign obliviously, but constructions that only require one message received and sent by the platform require trusted setup [34], non-standard crypto assumptions [36,37], or a concretely slow runtime with non-black-box reductions [35,48]. Instead, we take advantage of the fact that the platform's actions need only be verified by recipients who already know x_1 and x_2, so it suffices for the platform to produce a signature σ of the current time time together with a commitment to the two shares. The corresponding decommitment randomness can be sent to the receiver within the encrypted messenger payload, so that the recipient can verify that it is well-formed.

3 Definitions

In this section, we present rigorous definitions for several cryptographic protocols, a nearly black-box model of the Signal protocol that we use in this work, and a new definition for an asymmetric message franking scheme that generalizes TGLMR [72].

3.1 Definitions of Cryptographic Building Blocks

This work uses four standard cryptographic building blocks that we use and adapt from Boneh-Shoup [14] and Katz-Lindell [49]. In what follows, we define the message space as $\mathcal{M} := \{0,1\}^*$, the key space as $\mathcal{K} := \{0,1\}^n$, the ciphertext space as $\mathcal{C} := \{0,1\}^*$, the randomness space $\mathcal{R} := \{0,1\}^n$ and the signature space as $\Sigma := \{0,1\}^n$, where n denotes the security parameter.

Definition 1 (Commitment scheme)**.** *A non-interactive commitment scheme is defined by two algorithms* Com *and* Vf*.*

- Com *is an algorithm that takes a random string* $\mathsf{r} \leftarrow\!\$\, \mathcal{R}$*, and a plaintext message* $m \in \mathcal{M}$ *and outputs a commitment* $\mathsf{com} := \mathsf{Com}(m, \mathsf{r})$*.*
- Vf *is an algorithm that takes a commitment* com*, a string* r *and a plaintext message* m *and checks if* $\mathsf{Vf}(m, \mathsf{com}, \mathsf{r}) := (\mathsf{Com}(m, \mathsf{r}) \stackrel{?}{=} \mathsf{com})$*.*

Definition 2 (Binding commitment)**.** *A commitment scheme* $\pi = \{\mathsf{Com}, \mathsf{Vf}\}$ *is* computationally binding *if for all probabilistic polynomial time (PPT) adversaries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}_{\pi}^{\mathrm{binding}_{\mathrm{com}}}(\mathcal{A}) = \Pr[\mathsf{Com}(m, \mathsf{r}, \mathsf{param}) = \mathsf{Com}(m', \mathsf{r}', \mathsf{param}) \mid m \neq m'] \leq \mathsf{negl}(n).$$

Definition 3 (Hiding commitment)**.** *Let* $\pi = \{\mathsf{Com}, \mathsf{Vf}\}$ *be a commitment scheme. Let* $\mathsf{Com}_{\mathsf{hiding}}^{\mathcal{A}}$ *be defined by the following experiment:*

- *The adversary* $\mathcal{A}$ *outputs a pair of messages* $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}$*.*

- *A uniform bit* $b \in \{0,1\}$ *and the randomness* $\mathsf{r} \leftarrow \{0,1\}^n$ *are chosen.*
- *The adversary* $\mathcal{A}$ *is given access to the commitment oracle* $\mathsf{O}^{\mathsf{com-hiding}}$ *which on messages* m_0 *and* m_1 *computes and returns the commitment* $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{m}_b, \mathsf{r})$*, where* $\mathsf{Vf}(\mathsf{Com}(\mathsf{m}_b, \mathsf{r}), \mathsf{m}_b, \mathsf{r}) = 1$.
- *The output of the experiment is* 1 *if* $b' = b$ *and* 0 *otherwise.*

A commitment scheme π *is computationally hiding if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}_{\pi}^{\text{hiding}_{\text{com}}}(\mathcal{A}) = \Pr[\mathsf{Com}_{\mathsf{hiding}}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Definition 4 (Encryption scheme)**.** *A private key encryption scheme is defined by three algorithms* $\mathsf{EncKGen}$*,* Enc *and* Dec *over a finite message space* $\mathcal{M}$*.*

- $\mathsf{EncKGen}$ *is a probabilistic key generation algorithm that output a key pair* $(\mathsf{pk}, \mathsf{sk})$ *sampled uniformly at random from* $\mathcal{K}$*, where* pk *is defined as the public key and* sk *is defined as the secret key.*
- Enc *is the encryption algorithm that takes as an input* sk *and plaintext message* $m \in \mathcal{M}$ *and outputs* $\mathsf{c} := \mathsf{Enc}_{\mathsf{sk}}(m)$ *where* $c \in \mathcal{C}$*.*
- Dec *is the decryption algorithm that takes as an input* pk *and a ciphertext* c *in the ciphertext space* $\mathcal{C}$ *and outputs a plaintext message* $m := \mathsf{Dec}_{\mathsf{pk}}(\mathsf{c})$ *such that* $\mathsf{c} := \mathsf{Enc}_{\mathsf{sk}}(m)$*.*

Definition 5 (CCA security)**.** *Let* $\pi = \{\mathsf{EncKGen}, \mathsf{Enc}, \mathsf{Dec}\}$ *be an encryption scheme. Let* $\mathsf{ENC}_{\mathsf{cca},\pi}^{\mathcal{A}}(n)$ *denote the following experiment:*

- $\mathsf{EncKGen}$ *is run to obtain* $(\mathsf{pk}, \mathsf{sk})$ *and a uniform bit* $b \in \{0,1\}$ *is chosen. The adversary* $\mathcal{A}$ *is given* pk*.*
- *The adversary* $\mathcal{A}$ *is given access to the encryption oracle* $\mathsf{O}_{\mathsf{cca}}^{\mathsf{enc}}$ *which, on messages* $\mathsf{m}_0, \mathsf{m}_1$*, outputs a ciphertext* $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{m}_b)$*.*
- *The adversary* $\mathcal{A}$ *is given access to the decryption oracle* $\mathsf{O}_{\mathsf{cca}}^{\mathsf{decrypt}}$ *which outputs the decrypted plaintext message* m *under* sk *when handed out a ciphertext* c'*.*
- $\mathcal{A}$ *continues to interact with the decryption and encryption oracles, but may not request a decryption of any ciphertext* c *returned by* $\mathsf{O}_{\mathsf{cca}}^{\mathsf{enc}}$*.*
- *Finally* $\mathcal{A}$ *output a bit* b'*. The output of the experiment is defined to be* 1 *if* $b = b'$*, and* 0 *otherwise.*

We say that π *is* secure under a chosen-ciphertext attack *(CCA) if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}_{\pi}^{\text{enc}_{\text{cca}}} = \Pr[\mathsf{ENC}_{\mathsf{cca},\pi}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Definition 6 (CPA security)**.** *Let* $\pi = \{\mathsf{EncKGen}, \mathsf{Enc}, \mathsf{Dec}\}$ *be an encryption scheme. Let* $\mathsf{ENC}_{\mathsf{cpa},\pi}^{\mathcal{A}}(n)$ *denote a similar experiment to* $\mathsf{ENC}_{\mathsf{cca},\pi}^{\mathcal{A}}(n)$ *where the adversary* $\mathcal{A}$ *only has access to the encryption oracle that rename as* $\mathsf{O}_{\mathsf{cpa}}^{\mathsf{enc}}$*. We say that* π *is* secure under a chosen-plaintext attack *(CPA) if for all PPT adversaries* $\mathcal{A}$*, there exists a negligble function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}_{\pi}^{\text{enc}_{\text{cpa}}} = \Pr[\mathsf{ENC}_{\mathsf{cpa},\pi}^{\mathcal{A}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Definition 7 (Digital signature scheme)**.** *A signature scheme is defined as the triple of algorithms* $\mathsf{SigKGen}$*,* Sign*,* Vf *over the message space* $\mathcal{M}$ *and the signature space* Σ*.*

- $\mathsf{SigKGen}$ *is a probabilistic key generation algorithm that output a key pair* $(\mathsf{pk}, \mathsf{sk})$ *sampled from* $\mathcal{K}$*, where* pk *is the public verification key and* sk *is the secret signing key.*
- Sign *is the probabilistic signing algorithm that takes the signing key* sk *and a plaintext message* m *and outputs a signature* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})$*, where* $\sigma \in \Sigma$*.*
- Vf *is the deterministic verification algorithm which checks the signature* σ *against the plaintext message* m *and public key* pk *and outputs* $\perp$ *or* 1 *such that:*

$$\Pr[\mathsf{Vf}(\mathsf{pk}, \mathsf{m}, \mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})) = 1] = 1.$$

Definition 8 (EU-CMA security)**.** *Let* $\pi = \{\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Vf}\}$ *denote a digital signature scheme. Let* $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$ *be the experiment defined as:*

- $\mathsf{SigKGen}$ *is run to obtain* $(\mathsf{pk}, \mathsf{sk})$.
- *The adversary* $\mathcal{A}$ *is given* pk *and access to the signing oracle* $\mathsf{O}^{\mathsf{sign}}_{\mathsf{eu-cma}}$ *which on message* m *computes and outputs the signature* σ *of that message under the secret signing key* sk*. Let* $\mathcal{Q}$ *denote the set of all queries that* $\mathcal{A}$ *makes to* $\mathsf{O}^{\mathsf{sign}}_{\mathsf{eu-cma}}$.
- *The adversary* $\mathcal{A}$ *then outputs* (m', σ').
- *The experiment outputs* 1 *if and only if* $\mathsf{Vf}_{\mathsf{pk}}(\mathsf{m}', \sigma') = 1$ *and* $m' \notin \mathcal{Q}$*, and* 0 *otherwise.*

We say that π *is existentially unforgeable under an adaptive chosen-message attack (EU-CMA) if for all PPT adveries* $\mathcal{A}$*, there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}^{\mathrm{sig}_{\text{eu-cma}}}_{\pi}(\mathcal{A}) = \Pr[\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu\text{-}cma}}(n) = 1] \leq \mathsf{negl}(n).$$

Definition 9 (Hash function)**.** *A hash function with output length* l *is defined by two algorithms* Gen *and* H.

- Gen *is a probabilistic algorithm which outputs a key* $k \in \mathcal{K}$.
- H *is an algorithm which takes as input as key* k *and a string* $m \in \mathcal{M}$ *and outputs a string* $\mathsf{H}_k(m) \in \{0,1\}^{l(n)}$.

Definition 10 (Collision resistance)**.** *Let* $\pi = \{\mathsf{Gen}, \mathsf{H}\}$ *denote a hash function. Let* $\mathsf{Hash}^{\mathcal{A}}_{\mathsf{coll}}$ *be the experiment defined as:*

- Gen *is run to obtain* k.
- *The adversary* $\mathcal{A}$ *is given access to the hashing oracle* $\mathsf{O}^{\mathsf{hash}}$ *which on input* m *returns* $\mathsf{H}_k(\mathsf{m})$
- *The adversary then outputs* m_0 *and* m_1.
- *The experiment outputs* 1 *if and only if* $\mathsf{m}_0 \neq \mathsf{m}_1$ *and* $\mathsf{H}_k(\mathsf{m}_0) = \mathsf{H}_k(\mathsf{m}_1)$.

We say that π *is collision resistant if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}(n)$ *such that:*

$$\mathsf{Adv}^{\mathrm{hash}_{\mathrm{coll}}}_{\pi}(\mathcal{A}) = \Pr[\mathsf{Hash}^{\mathcal{A}}_{\mathsf{coll}}(n) = 1] \leq \mathsf{negl}(n).$$

3.2 Modeling an EEMS

End-to-end encrypted messaging systems (EEMS) using the Signal protocol [67] are a complex, delicate combination of standard cryptographic primitives. Starting with Cohn-Gordon et al. [22], there is a long line of research that analyzes the security of EEMS constructions such as the two-party Signal protocol itself (e.g., [4, 9, 46]), modified versions that provide provide stronger guarantees (e.g., [43, 45, 59]), and extensions to support group messaging (e.g., [18, 21, 61]). For the purposes of this work, we wish to treat an EEMS as a black box and consider only an API-level description of its operation and underlying security guarantees.

For this reason, we opt to use the ideal functionality modeling of an EEMS within the recent work of Bienstock et al. [13] rather than the game-based definitions in the literature [4, 9]). Their ideal functionality $\mathcal{F}_{\text{Signal}}$ models the creation, evolution, and destruction of communication 'sessions' between different pairs of parties, and keeps track of the long-term and ephemeral state that parties must hold for each operation. We follow this abstract model, with three changes. First, we allow a sender to attach public information outside of a sealed sender envelope that is visible to the platform, as shown in Fig. 2. Second, in order to support an anonymous network, we allow for inputs involving the parties' identities to be optional. Third, we add an explicit forgery method to highlight the fact that an EEMS achieves deniable authentication [29] (which is implicitly true in universally composable security models [20]): that is, the sender and receiver can forge a transcript showing that a message originated with the other party.

Hence, our abstract model of Signal involves three methods. All of these methods implicitly use the state of the party (or parties) that participate in each method.

- $\mathsf{send}_{\mathsf{eems}}(\mathsf{m}^*; \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}}) \rightarrow \mathsf{c}$: Run by the source client $\mathsf{id}_{\texttt{src}}$ with message m^*, this method sends a ciphertext c to the platform. This message m^* might contain payload and envelope components, similarly to how Signal's sealed sender operates. We sometimes omit the latter two inputs when they are clear from context.
- $\mathsf{deliver}_{\mathsf{eems}}(\mathsf{c}; \mathsf{id}_{\texttt{rec}}) \rightarrow \mathsf{m}^*$: An interactive protocol in which the platform delivers a ciphertext c to the receiver $\mathsf{id}_{\texttt{rec}}$. If this receiver was the intended target of a previous $\mathsf{send}_{\mathsf{eems}}$ that produced c, then they can decrypt using their local state to recover m^*. As above, we presume that $\mathsf{deliver}_{\mathsf{eems}}$ handles the payload and envelope of the message and splits c and m^* accordingly. Here, it is unclear whether to include $\mathsf{id}_{\texttt{rec}}$ as an input: for an anonymous communication channel it is important that the platform not know $\mathsf{id}_{\texttt{rec}}$, but for non-anonymous networks it may be required. We leave $\mathsf{id}_{\texttt{rec}}$ as an optional parameter, and throughout this work we focus on the stronger setting in which the network is anonymous so this input is omitted.
- $\mathsf{forge}_{\mathsf{eems}}(\mathsf{m}^*; \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}}) \rightarrow \mathsf{c}$: A forgery algorithm executed by a party $\mathsf{id}_{\texttt{rec}}$ and requiring its state $\mathsf{state}_{\texttt{rec}}$. It forges a transcript that looks as though the message m^* were sent by its counterparty $\mathsf{id}_{\texttt{src}}$ in an EEMS communication, with a destination of $\mathsf{id}_{\texttt{rec}}$. The parameters $\mathsf{id}_{\texttt{src}}$ and $\mathsf{id}_{\texttt{rec}}$ are optional for the same reasons as $\mathsf{send}_{\mathsf{eems}}$, and they will be omitted from this work.

3.3 Defining AMF with Preprocessing

Next, we present a rigorous definition for an asymmetric message franking system with preprocessing. This definition extends the one from TGLMR [72] in two ways. First, it includes an (optional) out-of-band communication between the moderator and sender, which results in a one-time *token* that is consumed when sending a message. Second, it is designed in a modular fashion so that it can be built on top of any EEMS that adheres to the model in §3.2.

Definition 11. *An* asymmetric message franking scheme with preprocessing $\mathsf{AMF} = (\mathsf{KGen}, \mathsf{TGen}, \mathsf{Frank}, \mathsf{Forward}, \mathsf{Stamp}, \mathsf{Inspect}, \mathsf{Verify}, \mathsf{Forge}_{\texttt{mod}}, \mathsf{Forge}_{\texttt{rec}})$ *is a tuple of algorithms called by different parties in the messaging ecosystem. We assume that each party has a unique identifier* id *provided by the underlying EEMS, and we define a state variable* state *for each party containing all keys and tokens generated by the AMF scheme and the underlying EEMS that have not yet been deleted. (Note that the user's* state *does not contain a transcript of prior messages exchanged.) The algorithms operate as follows.*

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}()$*: The key generation algorithm accessed by any party in the EEMS and used for creating (potentially multiple) cryptographic keys. The algorithm is at least run once at the beginning of time to setup the long-term key material for each party.*
- $\mathsf{TGen}(\mathsf{id}_{\texttt{src}}, \mathsf{time}_{\texttt{mod}}, \mathsf{sk}_{\texttt{mod}}) \rightarrow \mathsf{token}$*: An algorithm run by the moderator periodically that provides a one-time token for use when sending a message. An honest moderator should only provide tokens to a participant that correspond to their actual identity* $\mathsf{id}_{\texttt{src}}$*. It is assumed that the moderator can rely on the EEMS to authenticate a user's identity* $\mathsf{id}_{\texttt{src}}$ *before running* TGen*.*
- $\mathsf{Frank}(\mathsf{state}_{\texttt{src}}, \mathsf{m}, \mathsf{id}_{\texttt{rec}}, \mathsf{token}) \rightarrow \mathsf{m}_{\mathsf{frank}}$*: The message franking algorithm that allows a user with state* $\mathsf{state}_{\texttt{src}}$ *to frank a plaintext message* m *that they wish to send to a receiver* $\mathsf{id}_{\texttt{rec}}$*, using the* token *received during preprocessing. The* $\mathsf{state}_{\texttt{src}}$ *contains all key material produced by* KGen *and the underlying EEMS, although* Frank *need not use this state. The resulting franked message* $\mathsf{m}_{\mathsf{frank}}$ *can be sent to the platform using* $\mathsf{send}_{\mathsf{eems}}$*.*
- $\mathsf{Stamp}(\mathsf{c}_{\mathsf{frank}}, \mathsf{sk}_{\texttt{plat}}, \mathsf{time}) \rightarrow \mathsf{c}_{\mathsf{stamp}}$*: The stamping procedure run by the platform to authenticate and timestamp a franked cipher* $\mathsf{c}_{\mathsf{frank}}$*. The resulting stamped cipher* $\mathsf{c}_{\mathsf{stamp}}$ *can then be delivered to its intended recipient using the* $\mathsf{deliver}_{\mathsf{eems}}$ *method.* Stamp *does not have the sender or receiver's identity, even if* $\mathsf{deliver}_{\mathsf{eems}}$ *does.*
- $\mathsf{Forward}(\mathsf{m}_{\mathsf{frank}}, \mathsf{state}_{\texttt{fwd}}, \mathsf{id}_{\texttt{rec}}) \rightarrow \mathsf{m}_{\mathsf{frank}}{}'$*: Forwarding algorithm that allows a user with franked message* $\mathsf{m}_{\mathsf{frank}}$ *to produce a new franked message* $\mathsf{m}_{\mathsf{frank}}{}'$ *intended for a new recipient* $\mathsf{id}_{\texttt{rec}}$*. The format of* $\mathsf{m}_{\mathsf{frank}}$ *and* $\mathsf{m}_{\mathsf{frank}}{}'$ *are identical, so the ciphertexts of new and forwarded messages look indistinguishable to the platform.*

- $\mathsf{Verify}(\mathsf{m}_{\mathsf{frank}}, \mathsf{state}_{\mathtt{rec}}) \to (\mathsf{m}, \mathsf{report})$ *or* $\perp$*: The report construction algorithm that allows a receiver to validate a franked message* $\mathsf{m}_{\mathsf{frank}}$ *with respect to its state* $\mathsf{state}_{\mathtt{rec}}$*. If valid,* Verify *returns the corresponding plaintext message* m *along with a string* report *that the receiver can send to the moderator if they choose to report an abusive message.*
- $\mathsf{Inspect}(\mathsf{report}, \mathsf{sk}_{\mathtt{mod}}) \to (\mathsf{id}_{\mathtt{src}}, \mathsf{m}, \mathsf{time})$ *or* $\perp$*: The inspection algorithm that allows a moderator to handle reported message* report *using their secret key* $\mathsf{sk}_{\mathtt{mod}}$ *by validating and possibly source tracing them. If the verification step succeeds, the moderator produces the id of the source* $\mathsf{id}_{\mathtt{src}}$*, the message contents* m*, and a timestamp of the message* time.
- $\mathsf{Forge}_{\mathtt{mod}}(\mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{m}, \mathsf{sk}_{\mathtt{mod}}) \to \mathsf{m}_{\mathsf{frank}}$*: For deniability, this forgery protocol allows a moderator with secret key* $\mathsf{sk}_{\mathtt{mod}}$ *to forge a franked message with plaintext* m *on behalf of a user with id* $\mathsf{id}_{\mathtt{src}}$ *and with an intended recipient with id* $\mathsf{id}_{\mathtt{rec}}$*.*
- $\mathsf{Forge}_{\mathtt{rec}}(\mathsf{id}_{\mathtt{rec}}, \mathsf{m}, \mathsf{state}_{\mathtt{rec}}; \mathsf{id}_{\mathtt{src}}) \to \mathsf{c}_{\mathsf{frank}}$*: For deniability, this forgery algorithm allows a receiver with id* $\mathsf{id}_{\mathtt{rec}}$ *and state* $\mathsf{state}_{\mathtt{rec}}$ *to forge a franked ciphertext as though the message* m *was transmitted through the EEMS by the sender* $\mathsf{id}_{\mathtt{src}}$ *to the receiver* $\mathsf{id}_{\mathtt{rec}}$*. Note that* $\mathsf{id}_{\mathtt{src}}$ *is an optional parameter and may not be needed by systems that support anonymous messaging. In this work, we omit it from the presentation of this work since we are aiming for the highest level of anonymity.*

We say that an AMF scheme with preprocessing is secure if all computationally bounded attackers have negligible advantage in winning the deniability, anonymity, confidentiality, accountability, and backward secrecy games. These games are nuanced to describe, so rather than doing so here, we defer our discussion to the security analysis in §5.

4 Constructing Hecate

In this section, we describe the Hecate construction in detail. As per Def. 11, Hecate has eight algorithms. We describe them within this section, and we provide the full protocol specification of Hecate in Figs. 3-4. Because they are the most expensive of our standard crypto primitives, we also count the number of public key operations in each step here and in Table 2. For context, the prior AMF scheme from TGLMR [72] required at least 11 modular exponentiations per operation.

Key generation. KGen initializes a few long-term keys: the moderator samples an authenticated encryption key and both the moderator and platform sample a digital signature key pair. One strength of Hecate is that individual parties do not need any key material besides their existing EEMS keys, which simplifies our analysis of forward and backward security.

Token generation during preprocessing. In TGen, the moderator creates a batch of tokens for users at specific time intervals. Each token provides users with:

- Ephemeral session keys $(\mathsf{pk}_{\mathsf{e}}, \mathsf{sk}_{\mathsf{e}})$ that they can use to sign their message. None of the keys tie to users' long-term key material, thus giving the sender plausible deniability and confidentiality with respect to other users.
- A dual purpose randomized encryption $\mathsf{x}_1 := \mathsf{Enc}_{\mathsf{sk}_{\mathtt{mod}}}(\mathsf{id}_{\mathtt{src}})$ of the user's identity $\mathsf{id}_{\mathtt{src}}$ under the moderator's secret key $\mathsf{sk}_{\mathtt{mod}}$ that enforces accountability with respect to the moderator, confidentiality with respect to other user, and provides token integrity. The later property is ensured by having the sender create a share x_2 that along with x_1 reconstructs to a hash of the sent message.
- A timestamp t_1 that provides backward security.
- A signature σ_1 of the entire token that guarantees integrity and unforgeability of the token. σ_1 is signed with the moderator secret signing keys $(\mathsf{pk}_{\mathtt{mod},\sigma}, \mathsf{sk}_{\mathtt{mod},\sigma})$.

As shown in Table 2, the moderator requires two public key operations for token generation: 1 keygen operation to produce the ephemeral key pair and 1 signature to sign the public ephemeral key with the identity of the sender.

Message franking. The Frank method is executed every time the source wishes to send a message. The $\mathsf{construct}_{\mathsf{frank}}$ procedure requires an input plaintext message m from the source and consumes a single token

Command	Actor	KeyGen	Sign	Verify
TGen	mod	1	1	0
Frank	src	0	1	0
Stamp	plat	0	1	0
Forward	fwd	0	0	0
Verify	rec	0	0	3
Inspect	mod	0	0	3

Table 2: The number of public-key digital signature operations required for each of the interactive algorithms within Hecate (except for the one-time KGen at setup). We only count the additional cryptographic operations required for Hecate beyond those already required by the EEMS.

Source's Payload								Forwarder's Payload	Envelope		
x_1	x_2	nonce	pk_e	r	t_1	σ_1	σ_2	envelope of source	com	σ_3	t_2
32B	32B	12B	32B	32B	8B	64B	64B	104B	32B	64B	8B

Table 3: The format of a franked message delivered to the receiver, along with sizes in bytes for the implementation in §6. A franked message sent by the source is similar, except the envelope does not yet contain σ_3 or t_2.

at a time, and it produces a franked message m_{frank}. This can be combined with $send_{eems}$ to relay a franked ciphertext c_{frank} to the platform.

To produce the franked message, the sender begins by unpacking x_1 from the token and computes x_2 such that these variables constitute a 2-out-of-2 sharing of $H(m)$. Next, x_2 is signed via the ephemeral keys in the original token to produce σ_2. Collectively, x_2, σ_2, and elements of the pre-processing token (excluding the secret ephemeral key) will constitute the payload of the franked message. Then, the sender creates a commitment com of $x_1 \| x_2$ using the randomness r. The user then pushes com onto the envelope of the franked message and appends r to its payload. In total, a sender only requires 1 public key operation to sign the second share x_2. The constructed franked message m_{frank} has several properties: x_2 and com bind the online and preprocessing stages together, the signature allows the receiver to check the well-formedness of the message, and the use of an ephemeral signing key provides deniability with respect to anyone other than the moderator.

Stamping. In Stamp, the platform timestamps and digitally signs the envelope of a franked cipher c_{frank}; ergo, this procedure requires 1 public key operation. Then, the platform relays the resulting stamped cipher c_{stamp} to its intended recipient. Stamping prevents preprocessing token from being used indefinitely after a compromise to blame a victim client for unsent messages.

Verification and reporting. On reception, the receiver executes Verify to validate the signatures, timestamps expiration date, packet integrity, and check the envelope commitments against the inner tokens. If a message fails the integrity check, the receiver drops the packet and the application never displays the plaintext message. Otherwise, Verify generates a plaintext message m that can be displayed on the receiver's phone, and a report that can be sent out to the moderator. In Hecate, the report solely consists of the franked message m_{frank}. When a moderator receives report, they locally run the Inspect method which performs the same verification procedure as the recipient, and if successful, decrypts the source's identity from the ciphertext x_1 within the token. Both the receiver of a message and a moderator who receives a report require 3 signature verifications to check that the two shares are not tampered with and have the right timestamps.

Message forwarding. Verified messages can alternatively be forwarded using the optional Forward method. There are two differences between Forward and Frank: the forwarder creates a nonsensical commitment outside the Signal envelope, and it moves the true commitment and signed timestamp into the payload of the franked message. Because it reuses the prior signature, the forwarder doesn't perform any public key operations of its own. Additionally, Frank and Forward payloads are indistinguishable to the platform but distinguishable

```
KGen [mod and plat, separately]
$(\mathsf{pk}_{\texttt{mod}}, \mathsf{sk}_{\texttt{mod}}) \leftarrow\$ \mathsf{EncKGen}(1^n)$
$(\mathsf{pk}_{\texttt{mod},\sigma}, \mathsf{sk}_{\texttt{mod},\sigma}) \leftarrow\$ \mathsf{SigKGen}(1^n)$
$(\mathsf{pk}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}}) \leftarrow\$ \mathsf{SigKGen}(1^n)$
```

```
TGen [mod → src]
$\mathsf{token} := \mathsf{construct}_{\mathsf{token}}(\mathsf{sk}_{\texttt{mod}}, \mathsf{id}_{\texttt{src}})$
return $\mathsf{token}$
```

```
Frank [src → plat]
$\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{token}, \mathsf{m})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
Forward [fwd → rec]
$\mathsf{m}_{\mathsf{frank}}' := \mathsf{construct}_{\mathsf{fwd}}(\mathsf{m}_{\mathsf{frank}})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}')$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
Stamp [plat → rec]
$\mathsf{c}_{\mathsf{stamp}} := \mathsf{stamp}_{\mathsf{time}}(\mathsf{c}_{\mathsf{frank}})$
$\mathsf{c}_{\mathsf{stamp}}' := \mathsf{send}_{\mathsf{eems}}(\mathsf{c}_{\mathsf{stamp}})$
return $\mathsf{c}_{\mathsf{stamp}}'$
```

```
Verify [rec → mod/rec]
$\mathsf{m}_{\mathsf{frank}} := \mathsf{deliver}_{\mathsf{eems}}(\mathsf{c}_{\mathsf{stamp}})$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{move}_{\mathsf{stamp}}(\mathsf{m}_{\mathsf{frank}})$
$(\mathsf{m}, \mathsf{report}) := \mathsf{vfRec}(\mathsf{m}_{\mathsf{frank}})$
if $(\mathsf{m}, \mathsf{report}) \overset{?}{=} \perp$ :
    return $\perp$
return $(\mathsf{m}, \mathsf{report})$
```

```
Inspect [mod]
if $\mathsf{vfMsg}(\mathsf{report})$ :
    return $(\mathsf{Dec}(\mathsf{report}.\mathsf{x}_1), \mathsf{report}.\mathsf{t}_2)$
return $\perp$
```

Figure 3: Hecate's construction along with the transmissions using the encrypted messenger. The notation $[a \to b]$ means that party a executes the method and sends the returned value to b. Note that the plat relays messages between the src and the rec. The receiver of a message may elect to forward or report it; we assume that Forward is preceded by a successful invocation of Verify. See Fig. 4 for more details on the methods used here.

by the receiver; see Table 3 for the format of $\mathsf{m}_{\mathsf{frank}}$. The receiver of a forwarded message executes Verify using the commitment, signature, and timestamp inside the payload.

We defer discussion of Hecate's forgery algorithms to §5, since these are proof artifacts of the deniability property rather than actual elements of the construction.

5 Security Analysis

In this section, we formally define the security properties of asymmetric message franking (AMF) schemes with preprocessing, and we prove that Hecate guarantees them. All of our definitions are written as indistinguishability games $\mathsf{GAME}^{\mathcal{A}}_{\mathsf{b}}$, and we want to show that the adversary's advantage

$$\mathsf{Adv}^{\mathsf{game}}_{\mathsf{Hecate}}(\mathcal{A}) = \left|\Pr\left[\mathsf{GAME}^{\mathcal{A}}_1 = 1\right] - \Pr\left[\mathsf{GAME}^{\mathcal{A}}_0 = 1\right]\right|$$

is negligible for each game, if the adversary $\mathcal{A}$ is computationally bounded to probabilistically polynomial time (PPT).

5.1 Deniability

Deniability states that *a sender should always be able to deny that they sent a particular message to anyone, except to the moderator when a message is reported.* Deniability could hold with respect to a colluding moderator and receivers, as shown in the $\mathsf{DENM}^{\mathcal{A}}_{\mathsf{b}}$ game in Fig. 6, or against malicious receivers who are not colluding with an honest moderator, which corresponds to the $\mathsf{DENR}^{\mathcal{A}}_{\mathsf{b}}$ game in Fig. 6.

Each deniability game provides the adversary $\mathcal{A}$ with polynomially-many queries to an oracle $\mathsf{O}^{\mathsf{DENM}}_{\mathsf{b}}$ or $\mathsf{O}^{\mathsf{DENR}}_{\mathsf{b}}$, respectively. For each query, the adversary chooses a plaintext message m and the sender $\mathsf{id}_{\texttt{src}}$ and corrupted receiver $\mathsf{id}_{\texttt{rec}\mathcal{A}}$ of that message. Both oracles behave similarly: depending on the parameterized choice bit b, the oracle will either forge a message as a corrupted moderator/receiver as if originating from $\mathsf{id}_{\texttt{src}}$, or ask the honest source $\mathsf{id}_{\texttt{src}}$ to produce it themselves. Deniability requires that no adversary can distinguish between forged and real messages, even with access to the secret keys of malicious parties. In

```
$\mathsf{construct_{token}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{id}_{\mathtt{src}})$
$(\mathsf{pk_e}, \mathsf{sk_e}) \leftarrow\!\!\$\ \mathsf{SigKGen}(1^n)$
$\mathsf{t}_1 := \mathsf{time}()$
$\mathsf{x}_1 := \mathsf{Enc}_{\mathsf{sk}_{\mathtt{mod}}}(\mathsf{id}_{\mathtt{src}})$
$\sigma_1 := \mathsf{Sign}_{\mathsf{sk}_{\mathtt{mod},\sigma}}(\mathsf{x}_1 \| \mathsf{pk_e} \| \mathsf{t}_1)$
$\mathsf{token} := (\mathsf{x}_1, \mathsf{t}_1, \sigma_1, (\mathsf{pk_e}, \mathsf{sk_e}))$
$\mathbf{return}\ \mathsf{token}$
```

```
$\mathsf{construct_{frank}}(\mathsf{m}, \mathsf{token})$
$\mathsf{r} \leftarrow\!\!\$\ \{0,1\}^n$
$(\mathsf{x}_1, \mathsf{t}_1, \sigma_1, (\mathsf{pk_e}, \mathsf{sk_e})) = \mathsf{token}$
$\mathsf{x}_2 := \mathit{split}(\mathsf{x}_1, H(\mathsf{m}))$
$\sigma_2 := \mathsf{Sign}_{\mathsf{sk_e}}(\mathsf{x}_2)$
$\mathsf{com} := \mathsf{com_r}(\mathsf{x}_1 \| \mathsf{x}_2)$
$\mathsf{envelope} := \mathsf{com}$
$\mathsf{payload} := (\mathsf{x}_1, \mathsf{x}_2, \mathsf{r}, \mathsf{t}_1, \sigma_1, \sigma_2, \mathsf{pk_e})$
$\mathsf{m_{frank}} := (\mathsf{payload}, \mathsf{envelope})$
$\mathbf{return}\ \mathsf{m_{frank}}$
```

```
$\mathsf{construct_{fwd}}(\mathsf{m_{frank}})$
$\mathsf{m_{frank}} := \mathsf{move_{stamp}}(\mathsf{m_{frank}})$
$\mathsf{m_{frank}}.\mathsf{envelope} \leftarrow\!\!\$\ \{0,1\}^n$
$\mathbf{return}\ \mathsf{m_{frank}}$
```

```
$\mathsf{vfRec}(\mathsf{m_{frank}})$
$\mathbf{if}\ \mathsf{vfMsg}(\mathsf{m_{frank}}):$
    $\mathbf{return}\ \bot$
$\mathsf{report} := \mathsf{m_{frank}}$
$\mathbf{return}\ (\mathsf{m_{frank}}.\mathsf{m}, \mathsf{report})$
```

```
$\mathsf{vfMsg}(\mathsf{report})$
$b_1 := \mathsf{vfToken}(\mathsf{report})$
$b_2 := \mathsf{vfCom}(\mathsf{report})$
$b_3 := \mathsf{vfExp}(\mathsf{report})$
$\mathbf{return}\ b_1 \wedge b_2 \wedge b_3$
```

```
$\mathsf{vfToken}(\mathsf{report})$
$\mathit{reveal} := \mathsf{open}(\mathsf{x}_1, \mathsf{x}_2)$
$b_1 := (\mathit{reveal} \overset{?}{=} \mathsf{H}(\mathsf{m}))$
$b_2 := \mathsf{Vf}_{\mathsf{pk}_{\mathtt{mod}}}(\mathsf{x}_1 \| \mathsf{pk_e} \| \mathsf{t}_1, \sigma_1)$
$b_3 := \mathsf{Vf}_{\mathsf{pk_e}}(\mathsf{x}_2, \sigma_2)$
$\mathbf{return}\ b_1 \wedge b_2 \wedge b_3$
```

```
$\mathsf{vfExp}(\mathsf{m_{frank}})$
$b := |\mathsf{t}_1 - \mathsf{t}_2| \overset{?}{<} \mathsf{expiry}$
$\mathbf{return}\ b$
```

```
$\mathsf{vfCom}(\mathsf{m_{frank}})$
$b_1 := \mathsf{Vf}(\mathsf{x}_1 \| \mathsf{x}_2, \mathsf{com}, \mathsf{r})$
$b_2 := \mathsf{Vf}_{\mathsf{pk}_{\mathtt{plat}}}(\mathsf{com} \| \mathsf{t}_2, \sigma_3)$
$\mathbf{return}\ b_1 \wedge b_2$
```

```
$\mathsf{move_{stamp}}(\mathsf{m_{frank}})$
$\mathbf{if}\ \mathsf{stamp} \overset{?}{\notin} \mathsf{payload}:$
    $\mathsf{payload} := \mathsf{payload} \| \mathsf{envelope}$
$\mathbf{return}\ \mathsf{m_{frank}}$
```

```
$\mathsf{stamp_{time}}(\mathsf{c_{frank}}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{t}_2 = \mathsf{time}()$
$\sigma_3 := \mathsf{Sign}_{\mathsf{sk}_{\mathtt{plat}}}(\mathsf{com} \| \mathsf{t}_2)$
$\mathsf{c_{stamp}}.\mathsf{envelope} := (\mathsf{com} \| \mathsf{t}_2 \| \sigma_3)$
$\mathsf{c_{stamp}}.\mathsf{payload} := \mathsf{c_{frank}}.\mathsf{payload}$
$\mathbf{return}\ \mathsf{c_{stamp}}$
```

Figure 4: Hecate's subroutines. See Appendix 3.1 for specifications and security guarantees of the crypto primitives used. We omit writing out attribute access notation when it is obvious from the context (i.e. com for instance is a shorthand for $\mathsf{c_{frank}}.\mathsf{com}$).

other words, we want to show that a user can always repudiate having sent a message even when the moderator/receiver provides access to their secret key material. The knowledge of the original source of a message is non-transferable in that sense. Additionally, $\mathsf{O}_{\mathsf{b}}^{\mathsf{DENR}}$ provides an interesting guarantee: since the receiver forgery $\mathsf{Forge}_{\mathtt{rec}}$ does not depend on the original sender of a message in any way, then it can be called by *any* user even if they were not participating in the forwarding path of that message. This is a strong claim since the number of possible senders of any particular message in Hecate is now as large as the number of users in the EEMS.

In Hecate, $\mathsf{Forge}_{\mathtt{rec}}$ allows users to forge messages by using their own pre-processing tokens and constructing their own franked messages that they send back to themselves. The receiver does not have any more capabilities than any other user, and in particular the sender, without the secret key of the moderator. In other words, the only thing that a receiver can do is construct the franked message themselves. In Hecate, tokens and franked messages are not bound to the sender's long term key material and the origin of franked messages as a result is indistinguishable without the secret key of the moderator.

In $\mathsf{Forge}_{\mathtt{mod}}$ on the other hand, the moderator can forge messages by using their secret key to produce tokens for any user identity of their choosing, constructing franked messages on their behalf and sending the resulting message to the receiver. This is again a result of how Hecate does not bind the user's long term key material to a franked message.

5.1.1 Formalizing moderator deniability

Theorem 5.1. Hecate *is deniable against a moderator. Any adversary $\mathcal{A}$ has advantage* $\mathsf{Adv}_{\mathsf{Hecate}}^{\mathsf{denm}}(\mathcal{A}) = 0$.

The essence of Thm. 5.1 is the claim that that Hecate's real send routine is indistinguishable from the

```
$\mathsf{send_{amf}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve_{state}}(\mathsf{id}_{\mathtt{src}})$
$\textit{fetch}$ token $\textit{from}$ $\mathsf{state}_{\mathtt{src}}$
$\mathsf{m_{frank}} :=$
    $\mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}, \mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
$\mathsf{c_{frank}} := \mathsf{send_{eems}}(\mathsf{m_{frank}})$
return $\mathsf{c_{frank}}$
```

```
$\mathsf{fwd_{amf}}(\mathsf{m_{frank}}, \mathsf{id}_{\mathtt{fwd}}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{state}_{\mathtt{fwd}} := \mathsf{retrieve_{state}}(\mathsf{id}_{\mathtt{fwd}})$
$\mathsf{m_{frank}}' :=$
    $\mathsf{Forward}(\mathsf{state}_{\mathtt{fwd}}, \mathsf{m_{frank}}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{c_{frank}} := \mathsf{send_{eems}}(\mathsf{m_{frank}}')$
return $\mathsf{c_{frank}}$
```

```
$\mathsf{O^{corrupt}}(\mathsf{id})$
// $\mathsf{global_t}$ is only relevant in $BAC$
$\mathsf{corrupted} = \mathsf{corrupted} \cup (\mathsf{id}, \mathsf{global_t})$
$\mathsf{state_{id}} := \mathsf{retrieve_{state}}(\mathsf{id})$
return $\mathsf{state_{id}}$
```

```
$\mathsf{receive_{amf}}(\mathsf{c_{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{c_{stamp}} := \mathsf{Stamp}(\mathsf{c_{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{m_{frank}} := \mathsf{deliver_{eems}}(\mathsf{c_{stamp}})$
$\mathsf{state}_{\mathtt{rec}} := \mathsf{retrieve_{state}}(\mathsf{id}_{\mathtt{rec}})$
if $\mathsf{Verify}(\mathsf{m_{frank}}, \mathsf{state}_{\mathtt{rec}}) \overset{?}{=} 0$ :
    return $\perp$
return $\mathsf{m_{frank}}$
```

```
$\mathsf{O^{request}}(\mathsf{id})$
// $\mathsf{global_t}$ is only relevant in $BAC$
if $(\mathsf{id}, \mathsf{global_t}) \in \mathsf{corrupted}$ :
    create a batch of $d$ tokens
        using $\mathsf{TGen}()$ for id
    $\mathsf{T} := \mathsf{T} \cup \mathsf{token}$
    $\mathsf{global_t} := \mathsf{global_t} + 1$
    return $d$ tokens
return $\perp$
```

Figure 5: Game subroutines and oracles used in several games. Here, d is a fixed parameter known to the moderator.

moderator's forgery. Intuitively, Hecate achieves moderator deniability because Hecate implements algorithms TGen, Frank and Forward without ever using the user's long term key materials and instead relying on ephemeral keys generated by the moderator themselves. Additionally, the preprocessing token relies on an encryption and signature by the moderator in a way that is not directly bound to the message. This claim holds even against a distinguisher who also has the moderator's secret key – that is, if the moderator chooses to leak their own keys in an attempt to convince the rest of the world about the actions of a sender.

Proof of Thm 5.1. We show via a series of hybrids that $\mathsf{DENM}_0^{\mathcal{A}} \overset{c}{\approx} \mathsf{DENM}_1^{\mathcal{A}}$ as shown in Figure 6. This effectively boils down to showing that $\mathsf{O}_0^{\mathsf{DENM}} \overset{c}{\approx} \mathsf{O}_1^{\mathsf{DENM}}$, and hence that $\mathsf{send_{amf}}$ is computationally indistinguishable from $\mathsf{Forge}_{\mathtt{mod}}$ (in Fig. 6) and $\mathsf{send_{eems}}$. Figure 7 shows the sequence of hybrid steps here, starting with the existing $\mathsf{send_{amf}}$ subroutine as Game$_0$.

Game$_1$: In $\mathsf{send_{amf}}$, we replace "*fetch* token *from* $\mathsf{state}_{\mathtt{src}}$" with the moderator token construction method $\mathsf{construct_{token}}$ on the source's id $\mathsf{id}_{\mathtt{src}}$. We can do so because the moderator in Hecate does not require any information from a user $\mathsf{id}_{\mathtt{src}}$ in order to construct a token on their behalf. The adversary cannot observe the authentication that occurs between the sender and the moderator since the oracle is acting on behalf of the moderator in this game.

Game$_2$: In $\mathsf{send_{amf}}$, we can disregard the state passed to Frank and replace it with its instantiation $\mathsf{construct_{frank}}$. Similarly to $\mathsf{construct_{token}}$ (and hence TGen), $\mathsf{construct_{frank}}$ does not require the state of the sender to construct the franked message.

Notice that the resulting game from the prior series of hybrid has transformed $\mathsf{send_{amf}}$ to look exactly like $\mathsf{Forge}_{\mathtt{mod}}$. The resulting game is identical to $\mathsf{DENM}_0^{\mathcal{A}}$, where only the branch corresponding to $b = 0$ is executed. □

5.1.2 Formalizing receiver deniability

Theorem 5.2. Hecate *is deniable against a malicious receiver. Concretely, for any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{A}'$ and $\mathcal{A}''$ such that:*

$$\mathsf{Adv}_{\mathsf{Hecate}}^{\mathsf{denr}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathcal{E}}^{\mathrm{eems_{deniability}}}(\mathcal{A}') + \mathsf{Adv}_{\mathcal{E}}^{\mathrm{enc_{cpa}}}(\mathcal{A}'').$$

That is: Hecate*'s deniability reduces to the deniability and CPA security properties of the underlying EEMS scheme $\mathcal{E}$.*

```
$\mathsf{DENM}^{\mathcal{A}}_{\mathsf{b}}$
  $s_1, s_2 \leftarrow\!\$\ \mathcal{A}$
  $(\mathsf{pk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod}}) \leftarrow\!\$\ \mathsf{KGen}(s_1)$
  $(\mathsf{pk}_{\mathtt{mod},\sigma}, \mathsf{sk}_{\mathtt{mod},\sigma}) \leftarrow\!\$\ \mathsf{KGen}(s_2)$
  $(\mathsf{pk}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}}) \leftarrow\!\$\ \mathsf{KGen}(1^n)$
  $b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{O}^{\mathsf{DENM}}_{\mathsf{b}}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod},\sigma})$
  return $b'$
```

```
$\mathsf{DENR}^{\mathcal{A}}_{\mathsf{b}}$
  $(\mathsf{pk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod}}) \leftarrow\!\$\ \mathsf{KGen}(1^n)$
  $(\mathsf{pk}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}}) \leftarrow\!\$\ \mathsf{KGen}(1^n)$
  $b' \leftarrow\!\$\ \mathcal{A}^{\mathsf{O}^{\mathsf{DENR}}_{\mathsf{b}}}$
  return $b'$
```

```
$\mathsf{Forge}_{\mathtt{mod}}(\mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{m}, \mathsf{sk}_{\mathtt{mod}})$
  $\mathsf{token} := \mathsf{construct}_{\mathsf{token}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{id}_{\mathtt{src}})$
  $\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{m}, \mathsf{token})$
  return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{Forge}_{\mathtt{rec}}(\mathsf{m}, \mathsf{state}_{\mathtt{rec}})$
  fetch $\mathsf{token}$ from $\mathsf{state}_{\mathtt{rec}}$
  $\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{m}, \mathsf{token})$
  $\mathsf{c}_{\mathsf{frank}} := \mathsf{forge}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{state}_{\mathtt{rec}})$
  return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{DENR}}_{\mathsf{b}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
  if $b = 0$ :
    fetch $\mathsf{state}_{\mathtt{rec}}$
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{Forge}_{\mathtt{rec}}(\mathsf{m}, \mathsf{state}_{\mathtt{rec}})$
    $\mathsf{m}_{\mathsf{frank}}' :=$
      $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
  else :
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{amf}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
    $\mathsf{m}_{\mathsf{frank}}' :=$
      $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
  return $\mathsf{m}_{\mathsf{frank}}'$
```

```
$\mathsf{O}^{\mathsf{DENM}}_{\mathsf{b}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
  if $b = 0$ :
    $\mathsf{m}_{\mathsf{frank}} :=$
      $\mathsf{Forge}_{\mathtt{mod}}(\mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{m}, \mathsf{sk}_{\mathtt{mod}})$
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}})$
    $\mathsf{m}_{\mathsf{frank}}' :=$
      $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
  else :
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{amf}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
    $\mathsf{m}_{\mathsf{frank}}' :=$
      $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
  return $\mathsf{m}_{\mathsf{frank}}'$
```

Figure 6: The security games for Deniability with respect to the Receiver ($\mathsf{DENR}^{\mathcal{A}}_{\mathsf{b}}$) and Moderator ($\mathsf{DENM}^{\mathcal{A}}_{\mathsf{b}}$).

```
$\mathsf{send}_0(n)$
  1 : $\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src}})$
  2 : fetch $\mathsf{token}$ from $\mathsf{state}_{\mathtt{src}}$
  3 : $\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}, \mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
  4 : $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
  5 : return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_1(n)$
  $\mathsf{construct}_{\mathsf{token}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{id}_{\mathtt{src}})$
  $\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}, \mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
  $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
  return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_2(n)$
  $\mathsf{construct}_{\mathsf{token}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{id}_{\mathtt{src}})$
  $\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{m}, \mathsf{token})$
  $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
  return $\mathsf{c}_{\mathsf{frank}}$
```

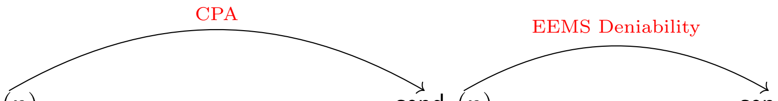
Figure 7: The hybrid steps modifying $\mathsf{send}_{\mathsf{amf}}$ in the Moderator Deniability game

The main difference with moderator deniability is that the adversary has to perform a forgery without the secret keys of the moderator. Intuitively, a forger can use her own tokens to create a franked message of her choosing and claim that it came from another source. Users with no access to the moderator's secret key should not be able to verify her claim without breaking the underlying encryption schemes.

Proof of Thm. 5.2. We show via a series of hybrids that $\mathsf{DENR}^{\mathcal{A}}_0 \overset{c}{\approx} \mathsf{DENR}^{\mathcal{A}}_1$, and more precisely show that $\mathsf{O}^{\mathsf{DENR}}_0 \overset{c}{\approx} \mathsf{O}^{\mathsf{DENR}}_1$, and hence that $\mathsf{send}_{\mathsf{amf}}$ is computationally indistinguishable from $\mathsf{Forge}_{\mathtt{rec}}$ (Figure 6).

Game$_0$: We start with $\mathsf{O}^{\mathsf{DENR}}_1$, we focus on the if-else branch corresponding to $b = 1$ since its the only one executed. We can disregard the other branch.

Game$_1$: In $\mathsf{O}^{\mathsf{DENR}}_1$, we replace $\mathsf{state}_{\mathtt{src}}$ in the $\mathsf{send}_{\mathsf{amf}}$ subroutine with $\mathsf{state}_{\mathtt{rec}}$ (lines 1, 2, 4). This is equivalent to replacing $\mathsf{send}_{\mathsf{amf}}$ with its instantiation $\mathsf{construct}_{\mathsf{frank}}$. Since the adversary does not have access to the moderator's secret key, then $\mathcal{A}$ has negligible advantage in distinguishing between different x_1 values in the constructed pre-processing token and the franked message. Specifically, they cannot distinguish between an encryption of $\mathsf{id}_{\mathtt{src}}$ and $\mathsf{id}_{\mathtt{rec}}$ without breaking the CPA security of the symmetric key encryption scheme used by the moderator during preprocessing. We formally show this by constructing an adversary $\mathcal{B}$ that can break the CPA security game $\mathsf{ENC}^{\mathcal{A}}_{\mathsf{cpa}}$. When adversary $\mathcal{A}$ queries oracle $\mathsf{O}^{\mathsf{DENR}}_1$, $\mathcal{B}$ queries $\mathsf{O}^{\mathsf{enc}}_{\mathsf{cpa}}$ with $\mathsf{id}_{\mathtt{src}}$ and $\mathsf{id}_{\mathtt{rec}}$, constructs the franked message with the resulting cipher-text. When $\mathcal{A}$ submits a choice bit b, $\mathcal{B}$ returns the same bit to $\mathsf{ENC}^{\mathcal{A}}_{\mathsf{cpa}}$ and succeeds if and only if $\mathcal{A}$ can distinguish between Games 0 and 1.



```
$\mathsf{send}_0(n)$
    $\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src}})$
    $\textit{fetch}$ $\mathsf{token}$ $\textit{from}$ $\mathsf{state}_{\mathtt{src}}$
    $\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}, \mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
    $\mathbf{return}$ $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_1(n)$
    $\mathsf{state}_{\mathtt{rec}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{rec}})$
    $\textit{fetch}$ $\mathsf{token}$ $\textit{from}$ $\mathsf{state}_{\mathtt{rec}}$
    $\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{m}, \mathsf{token})$
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
    $\mathbf{return}$ $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_2(n)$
    $\mathsf{state}_{\mathtt{rec}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{rec}})$
    $\textit{fetch}$ $\mathsf{token}$ $\textit{from}$ $\mathsf{state}_{\mathtt{rec}}$
    $\mathsf{m}_{\mathsf{frank}} := \mathsf{construct}_{\mathsf{frank}}(\mathsf{m}, \mathsf{token})$
    $\mathsf{c}_{\mathsf{frank}} := \mathsf{forge}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
    $\mathbf{return}$ $\mathsf{c}_{\mathsf{frank}}$
```

Figure 8: The hybrid steps modifying $\mathsf{send}_{\mathsf{amf}}$ in the Receiver Deniability game

Game$_2$: We replace $\mathsf{send}_{\mathsf{eems}}$ with $\mathsf{forge}_{\mathsf{eems}}$ since the underlying EEMS provides receiver deniability and the receiver hence can forge a channel with themselves.

The resulting game is identical to $\mathsf{DENR}_0^{\mathcal{A}}$, where only the branch corresponding to $b = 0$ is executed. □

5.2 Anonymity

Loosely speaking, the anonymity properties that we consider in this work restrict the moderator and any client from learning the metadata about the senders, receivers, and forwarders of messages that are transmitted between other people. We make no claims here about the level of anonymity provided by the underlying network environment (e.g., using sealed sender or Tor). Instead, our goal is to capture that the cryptography does not weaken any anonymity guarantees that happen to be provided by the underlying environment. Put another way: if we assume that the underlying network provides perfect anonymity, we examine the amount of metadata that each entity can learn through the abuse reporting system alone.

5.2.1 Anonymity with respect to the receiver

First, anonymity with respect to the receiver guarantees that *receivers should not be able to learn any other member of the forwarding path of a message beyond their direct neighbors.* For this security property, we assume that senders and forwarders of a message are honest and wish to hide themselves from non-neighboring recipients in the presence of an honest moderator.

5.2.2 Anonymity with respect to the receiver

We model this property in the $\mathsf{ANONR}_{\mathsf{b}}^{\mathcal{A}}$ game. The adversary can send and forward messages between parties using $\mathsf{O}^{\mathsf{send}}_{\mathsf{anon,b}}$, $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}$ and $\mathsf{O}^{\mathsf{deliver}}_{\mathsf{anon}}$ and is provided with the resulting franked message $\mathsf{m}_{\mathsf{frank}}$.

In this game, we do not attempt to hide chat participants from on another. To that end, both $\mathsf{O}^{\mathsf{send}}_{\mathsf{anon,b}}$ and $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}$ call $\mathsf{check}_{\mathsf{topology}}$ to ensure that the adversary provided the same pairs of senders and recipients when either of the provided receivers is corrupted. Without this check, $\mathcal{A}$ would trivially win the game by inspecting which of their provided correspondents sent the message or who received it. Additionally, $\mathsf{check}_{\mathsf{topology}}$ ensures that $\mathsf{O}^{\mathsf{send}}_{\mathsf{anon,b}}$ and $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}$ handle messages relayed from honest senders and forwarders. $\mathsf{O}^{\mathsf{deliver}}_{\mathsf{anon}}$ on the other hand allows $\mathcal{A}$ to send franked messages from corrupted nodes to an honest receiver. In $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}$, we also check that the queried $\mathsf{m}_{\mathsf{frank}}$ was initially received by either of the provided forwarders using the $\mathsf{check}_{\mathsf{received}}$ method in order to model the actual behavior of messages forwarders. Both $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}$ and $\mathsf{O}^{\mathsf{deliver}}_{\mathsf{anon}}$ check that the provided franked message are consistent with the recipient's state by calling Vf in the $\mathsf{receive}_{\mathsf{amf}}$ subroutine, thus eliminating any trivial wins that arise from malformed messages.

All three oracles, along with the corruption oracles $\mathsf{O}^{\mathsf{corrupt}}$ and $\mathsf{O}^{\mathsf{request}}$, allow the adversary to adaptively build any two message paths of their choice (modulo the topological restrictions) and receive the transcript of the chosen path, effectively encompassing the full power of a malicious recipient that may intercept messages along the path. The adversary is tasked with guessing which of the two message paths, with honest sources/roots, was chosen by the game. If they fail to distinguish between them, then they would

```
$\mathsf{ANONR}_b^{\mathcal{A}}$
$s \leftarrow\$ \mathcal{A}$
$(\mathsf{pk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod}}) \leftarrow\$ \mathsf{KGen}(1^n)$
$(\mathsf{pk}_{\mathtt{mod},\sigma}, \mathsf{sk}_{\mathtt{mod},\sigma}) \leftarrow\$ \mathsf{KGen}(1^n)$
$(\mathsf{pk}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}}) \leftarrow\$ \mathsf{KGen}(s)$
$b' := \mathcal{A}^{\mathsf{O}_b^{\mathsf{send}}, \mathsf{O}_b^{\mathsf{fwd}}, \mathsf{O}^{\mathsf{deliver}}}(\mathsf{sk}_{\mathtt{plat}})$
return $b'$
```

```
$\mathsf{ANONM}_b^{\mathcal{A}}$
$s_1, s_2, s_3 \leftarrow\$ \mathcal{A}$
$(\mathsf{pk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod}}) \leftarrow\$ \mathsf{KGen}(s_1)$
$(\mathsf{pk}_{\mathtt{mod},\sigma}, \mathsf{sk}_{\mathtt{mod},\sigma}) \leftarrow\$ \mathsf{KGen}(s_2)$
$(\mathsf{pk}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}}) \leftarrow\$ \mathsf{KGen}(s_3)$
$b' := \mathcal{A}^{\mathsf{O}_b^{\mathsf{fwd}}, \mathsf{O}^{\mathsf{deliver}}}(\mathsf{sk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod},\sigma}, \mathsf{sk}_{\mathtt{plat}})$
return $b'$
```

```
$\mathsf{O}^{\mathsf{deliver}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{receive}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{R} := \mathsf{R} \cup \langle \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{id}_{\mathtt{rec}} \rangle$
return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{check}_{\mathsf{received}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}0}, \mathsf{id}_{\mathtt{rec}1})$
if $\langle \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}0}, \mathsf{id}_{\mathtt{rec}1} \rangle \notin \mathsf{R}$ :
  return $\bot$
return $1$
```

```
$\mathsf{O}^{\mathsf{send}}_{\mathsf{anon,b}}(\mathsf{m}, \langle \mathsf{id}_{\mathtt{src}0}, \mathsf{id}_{\mathtt{rec}0} \rangle, \langle \mathsf{id}_{\mathtt{src}1}, \mathsf{id}_{\mathtt{rec}1} \rangle, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
if $\mathsf{check}_{\mathsf{topology}}(\langle \mathsf{id}_{\mathtt{src}0}, \mathsf{id}_{\mathtt{rec}0} \rangle, \langle \mathsf{id}_{\mathtt{src}1}, \mathsf{id}_{\mathtt{src}1} \rangle) = \bot$ :
  return $\bot$
$\mathsf{c}_{\mathsf{frank}} = \mathsf{send}_{\mathsf{amf}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{R} := \mathsf{R} \cup \langle \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}0}, \mathsf{id}_{\mathtt{rec}1} \rangle$
return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{fwd}}_{\mathsf{anon,b}}(\mathsf{m}_{\mathsf{frank}}, \langle \mathsf{id}_{\mathtt{fwd}0}, \mathsf{id}_{\mathtt{rec}0} \rangle, \langle \mathsf{id}_{\mathtt{fwd}1}, \mathsf{id}_{\mathtt{rec}1} \rangle, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
if $\mathsf{check}_{\mathsf{topology}}(\langle \mathsf{id}_{\mathtt{fwd}0}, \mathsf{id}_{\mathtt{rec}0} \rangle, \langle \mathsf{id}_{\mathtt{fwd}1}, \mathsf{id}_{\mathtt{rec}1} \rangle) = \bot$ :
  return $\bot$
if $\mathsf{check}_{\mathsf{received}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}0}, \mathsf{id}_{\mathtt{rec}1}) = \bot$ :
  return $\bot$
$\mathsf{c}_{\mathsf{frank}} = \mathsf{fwd}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{m}_{\mathsf{frank}}' := \mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
$\mathsf{R} := \mathsf{R} \cup \langle \mathsf{m}_{\mathsf{frank}}', \mathsf{id}_{\mathtt{rec}0}, \mathsf{id}_{\mathtt{rec}1} \rangle$
return $\mathsf{m}_{\mathsf{frank}}'$
```

```
$\mathsf{check}_{\mathsf{topology}}(\langle \mathsf{id}_{\mathtt{src}0}, \mathsf{id}_{\mathtt{rec}0} \rangle, \langle \mathsf{id}_{\mathtt{src}1}, \mathsf{id}_{\mathtt{rec}1} \rangle)$
if $\mathsf{id}_{\mathtt{src}0} \vee \mathsf{id}_{\mathtt{src}1} \in \mathsf{corrupted}$ :
  return $\bot$
if $\mathsf{id}_{\mathtt{rec}0} \vee \mathsf{id}_{\mathtt{rec}1} \in \mathsf{corrupted}$ :
  if $\mathsf{id}_{\mathtt{rec}0} \neq \mathsf{id}_{\mathtt{rec}1} \vee \mathsf{id}_{\mathtt{src}0} \neq \mathsf{id}_{\mathtt{src}1}$ :
    return $\bot$
return $true$
```

Figure 9: The security games for Anonymity with respect to the Receiver and Moderator.

have failed to determine the original sender of that message and the anonymity of that user and honest forwarder along the path is preserved.

Theorem 5.3. Hecate *is anonymous with respect to the receiver. For any PPT adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ that can win the chosen plaintext attack game with advantage* $\mathsf{Adv}^{\mathsf{anonr}}_{\mathsf{Hecate}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{enc_{cpa}}}_{\mathsf{Hecate}}(\mathcal{A}')$.

Informally, this theorem holds because the preprocessing tokens in Hecate only contain any information about the original sender's identity in encrypted form; without access to the moderator's secret key, a receiver can't distinguish between tokens that originate from different senders. Additionally, Hecate stores no information about forwarders of a message at all, thereby guaranteeing their anonymity as well. We provide a rigorous proof of this theorem below.

Proof of Thm. 5.3. At a high level, Hecate guarantees sender anonymity for the same reason it achieves receiver deniability: with no access to the moderator's secret key, message recipients cannot tell who the originator of a message is without breaking the underlying encryption scheme. Additionally, since the commitment scheme used for envelope commitments is hiding, then access to the platforms secret key does not reveal anything about the sender of a message. On the other hand, forwarding franked messages in Hecate does not utilize the forwarder's state or identity. In other words, no attribute in any franked message can be traced back to a forwarder guaranteeing forwarder anonymity.

We show via a series of hybrids that $\mathsf{ANONR}_0^{\mathcal{A}} \overset{c}{\approx} \mathsf{ANONR}_1^{\mathcal{A}}$.

Game$_0$: We start with $\mathsf{ANONR}_b^{\mathcal{A}}$ with $b = 0$.

Game$_1$: We replace $\mathsf{state}_{\mathtt{src},0}$ with $\mathsf{state}_{\mathtt{src},1}$ in the $\mathsf{send}_{\mathsf{amf}}$ subroutine that is called by $\mathsf{O}^{\mathsf{send}}_{\mathsf{anon,b}}$. The moderator in Hecate binds a sender to a token by encrypting their identity on line 3 in $\mathsf{construct}_{\mathsf{token}}$ (Figure

ENC-CPA

```
$\mathsf{send}_0$
$\mathsf{state}_{\mathtt{src},0} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src},0})$
$\textit{fetch}$ token $\textit{from}$ $\mathsf{state}_{\mathtt{src},0}$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src},0}, \mathsf{m}, \mathsf{id}_{\mathtt{rec},0}, \mathsf{token})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_1$
$\mathsf{state}_{\mathtt{src},1} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src},1})$
$\textit{fetch}$ token $\textit{from}$ $\mathsf{state}_{\mathtt{src},1}$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src},0}, \mathsf{m}, \mathsf{id}_{\mathtt{rec},0}, \mathsf{token})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_2$
$\mathsf{state}_{\mathtt{src},1} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src},1})$
$\textit{fetch}$ token $\textit{from}$ $\mathsf{state}_{\mathtt{src},1}$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src},1}, \mathsf{m}, \mathsf{id}_{\mathtt{rec},0}, \mathsf{token})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{send}_3$
$\mathsf{state}_{\mathtt{src},1} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src},1})$
$\textit{fetch}$ token $\textit{from}$ $\mathsf{state}_{\mathtt{src},1}$
$\mathsf{m}_{\mathsf{frank}} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src},1}, \mathsf{m}, \mathsf{id}_{\mathtt{rec},1}, \mathsf{token})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

Figure 10: The hybrid steps of $\mathsf{send}_{\mathsf{amf}}$ in the $\mathsf{ANONR}_{\mathsf{b}}^{\mathcal{A}}$ game. We refer the reader to $\mathsf{ANONM}_{\mathsf{b}}^{\mathcal{A}}$ hybrids for $\mathsf{fwd}_{\mathsf{amf}}$'s hybrids (*We omit the final hybrid from this figure for ease of presentation*).

4) and generating the sub-token x_1. Without the moderator's key, the adversary cannot decrypt the identity of the sender within x_1 in the token constructed in $\mathsf{send}_{\mathsf{amf}}$ (Figure 5) on line 2 , without breaking the CPA security of the underlying encryption scheme. The formalism here is similar to Game 1 of $\mathsf{DEN}_{\mathtt{rec}}^{\mathcal{A}}$.

Game$_2$: We replace $\mathsf{state}_{\mathtt{src},0}$ with $\mathsf{state}_{\mathtt{src},1}$ in Frank in the $\mathsf{send}_{\mathsf{amf}}$ subroutine that is called by $\mathsf{O}_{\mathsf{anon},\mathsf{b}}^{\mathsf{send}}$. The user's state and long term keys are never used during the construction of the franked message via $\mathsf{construct}_{\mathsf{frank}}$ in Hecate. $\mathsf{m}_{\mathsf{frank}}$ is only bound to a particular user by x_1 which we have discussed and handled in the previous hybrid.

Game$_3$: We replace $\mathsf{id}_{\mathtt{rec}0}$ with $\mathsf{id}_{\mathtt{rec}1}$ in both $\mathsf{O}_{\mathsf{anon},0}^{\mathsf{send}}$ and $\mathsf{O}_{\mathsf{anon},0}^{\mathsf{fwd}}$. In $\mathsf{O}_{\mathsf{anon}}^{\mathsf{send}}$, $\mathsf{check}_{\mathsf{topology}}$ enforces that $\mathsf{id}_{\mathtt{src}0} = \mathsf{id}_{\mathtt{src}1}$ and $\mathsf{id}_{\mathtt{rec}0} = \mathsf{id}_{\mathtt{rec}1}$ when either of the receivers is corrupted (and similarly for $\mathsf{O}_{\mathsf{anon}}^{\mathsf{fwd}}$). If both receivers are honest, then we can make this replacement because Hecate does not use any information related to the receiver of a message when constructing or forwarding a franked message (see $\mathsf{construct}_{\mathsf{frank}}$ and $\mathsf{construct}_{\mathsf{fwd}}$ respectively) and the adversary cannot hence distinguish between both Games 2 and 3.

We have shown that $\mathsf{O}_{\mathsf{anon},0}^{\mathsf{send}} \overset{c}{\approx} \mathsf{O}_{\mathsf{anon},1}^{\mathsf{send}}$ in Hecate since both oracles are now identical. The next series of hybrids are similar to the ones we have already seen.

Game$_4$: We replace $\mathsf{state}_{\mathtt{fwd},0}$ with $\mathsf{state}_{\mathtt{fwd},1}$ in $\mathsf{Forward}$ in the $\mathsf{fwd}_{\mathsf{amf}}$ subroutine that is called by $\mathsf{O}_{\mathsf{anon},\mathsf{b}}^{\mathsf{fwd}}$. The reason we can do so is two folds: (1) Hecate does not use the forwarder states in constructing the franked message, (2) when the receiver of a forwarded message is corrupted, $\mathsf{check}_{\mathsf{topology}}$ enforces that $\mathsf{id}_{\mathtt{fwd},0} = \mathsf{id}_{\mathtt{fwd},1}$ and $\mathsf{id}_{\mathtt{rec}0} = \mathsf{id}_{\mathtt{rec}1}$. In either case, the source or forwarder of a message have to be honest.

The resulting game is identical to $\mathsf{ANONR}_1^{\mathcal{A}}$. We conclude that Hecate is sender and forwarder anonymous and the advantage of the adversary is equal to that of the $\mathsf{ENC}_{\mathsf{cca}}^{\mathcal{A}}$. □

Second, anonymity with respect to the moderator ensures that *the moderator should not be able to learn members of the forwarding path of a reported message beyond the neighbors of colluding receivers and the reported source.* Here, honest forwarders want to be assured that, when their direct contacts are honest, only their neighboring recipients know that they forwarded a specific message. Since the moderator can directly trace the source of a franked message after receiving it, we can additionally assume that for all intents and purpose the sender of a message in this game is also colluding with them.

```
send_0
$\mathsf{state}_{\mathtt{fwd},0} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd},0})$
$\mathsf{m}_{\mathsf{frank}}' := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd},0}, \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec},0})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}')$
$\textbf{return } \mathsf{c}_{\mathsf{frank}}$
```

```
send_1
$\mathsf{state}_{\mathtt{fwd},1} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd},1})$
$\mathsf{m}_{\mathsf{frank}}' := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd},1}, \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec},0})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}')$
$\textbf{return } \mathsf{c}_{\mathsf{frank}}$
```

```
send_2
$\mathsf{state}_{\mathtt{fwd},1} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd},1})$
$\mathsf{m}_{\mathsf{frank}}' := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd},1}, \mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec},1})$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}}')$
$\textbf{return } \mathsf{c}_{\mathsf{frank}}$
```

Figure 11: The hybrid steps of $\mathsf{send}_{\mathsf{amf}}$ in the $\mathsf{ANONM}_{\mathsf{b}}^{\mathcal{A}}$ game. We refer the reader to the moderator anonymity game for $\mathsf{fwd}_{\mathsf{amf}}$'s hybrids.

5.2.3 Anonymity with respect to the moderator

We show this property via the $\mathsf{ANONM}_1^{\mathcal{A}}$ game, where the adversary now only has access to $\mathsf{O}_{\mathsf{anon,b}}^{\mathsf{fwd}}$, $\mathsf{O}_{\mathsf{anon,b}}^{\mathsf{deliver}}$, $\mathsf{O}^{\mathsf{corrupt}}$, $\mathsf{O}^{\mathsf{request}}$ oracles. Contrary to the prior property, the adversary now chooses the moderator's secret keys and controls the root of the message path. They must now, as a result, use $\mathsf{O}_{\mathsf{anon,b}}^{\mathsf{deliver}}$ to deliver messages between the compromised sender and honest recipients. If, by the end of the game, the adversary cannot guess the correct message path chosen by the game then they could not have distinguished between the different honest forwarders provided in each message path. The adversary in this game is strictly stronger than the one in $\mathsf{ANONR}_{\mathsf{b}}^{\mathcal{A}}$ because of the gained source tracing capability from the moderators secret keys, and hence implies the anonymity of forwarders in the presence of malicious receivers and an honest moderator. However, $\mathsf{ANONR}_{\mathsf{b}}^{\mathcal{A}}$ independently makes that guarantee because of the way $\mathsf{O}_{\mathsf{anon,b}}^{\mathsf{fwd}}$ handles interactions between honest users. By the end of $\mathsf{ANONR}_{\mathsf{b}}^{\mathcal{A}}$, if the adversary could not guess the message path chosen by the game, then they could not distinguish between honest forwarders with honest neighbors as well.

Theorem 5.4. Hecate *is anonymous with respect to the moderator. Any adversary $\mathcal{A}$ has advantage* $\mathsf{Adv}_{\mathsf{Hecate}}^{\mathsf{anonm}}(\mathcal{A}) = 0$.

Proof of Thm. 5.4. In this proof we need to show that a corrupted moderator cannot learn the forwarding path of a message beyond senders/forwarders/receivers that she has already corrupted and their neighbors. The reader is referred to the similar receiver anonymity proof for more details.

We can show via a series of hybrids that $\mathsf{ANONM}_0^{\mathcal{A}} \overset{c}{\approx} \mathsf{ANONM}_1^{\mathcal{A}}$.

Game$_0$: We start with $\mathsf{ANONM}_0^{\mathcal{A}}$ with $b = 0$.

Game$_1$: We replace $\mathsf{state}_{\mathtt{fwd},0}$ with $\mathsf{state}_{\mathtt{fwd},1}$ in $\mathsf{Forward}$ in the $\mathsf{fwd}_{\mathsf{amf}}$ subroutine that is called by $\mathsf{O}_{\mathsf{fwd,b}}^{\mathsf{fwd}}$. The reason we can do so is two folds: (1) Hecate does not use the forwarder states in constructing the franked message, (2) when the receiver of a forwarded message is corrupted, $\mathsf{check}_{\mathsf{topology}}$ enforces that $\mathsf{id}_{\mathtt{fwd},0} = \mathsf{id}_{\mathtt{fwd},1}$ and (3) that both forwarders passed to $\mathsf{O}_{\mathsf{anon,0}}^{\mathsf{fwd}}$ are honest.

Game$_3$: We can replace $\mathsf{id}_{\mathtt{rec}0}$ with $\mathsf{id}_{\mathtt{rec}1}$ in the $\mathsf{fwd}_{\mathsf{amf}}$ and $\mathsf{receive}_{\mathsf{amf}}$ subroutines that are called by $\mathsf{O}_{\mathsf{anon,b}}^{\mathsf{fwd}}$. $\mathsf{check}_{\mathsf{topology}}$ enforces that $\mathsf{id}_{\mathtt{rec}0} = \mathsf{id}_{\mathtt{rec}1}$ when either of the receivers is corrupted (and similarly for $\mathsf{O}_{\mathsf{anon,0}}^{\mathsf{fwd}}$). If both receivers are honest, then we can make this replacement because Hecate does not use any information related to the receiver of a message when constructing or forwarding a franked message (see $\mathsf{construct}_{\mathsf{fwd}}$) and the adversary cannot hence distinguish between both Games 2 and 3.

We can conclude that $\mathsf{ANONM}_0^{\mathcal{A}}$ becomes indistinguishable from $\mathsf{ANONM}_1^{\mathcal{A}}$. □

Note that our construction and security games do not consider forwarding graphs (i.e. trees with cycles). In those cases, users can identify that the same message was forwarded to them multiple times, a property called *tree linkability* in prior work [57]. Additionally, we allow users (but not the platform or the moderator) to distinguish between a sent and a forwarded message as is the case in several messaging system. We believe that an exciting opportunity for future work is to combine the ideas in this paper with the tree unlinkability scheme by Peale et al. [57].

5.3 Message Confidentiality and Forward Secrecy

Message confidentiality dictates that *any party not involved in the creation, reception or reporting of a message should not be able to learn anything about the message.* Moreover, forward secrecy guarantees that

```
$\mathsf{CONF\text{-}FS}_{\mathsf{b}}^{\mathcal{A}}$
$s_1, s_2 \leftarrow\$ \mathcal{A}$
$(\mathsf{pk}_{\texttt{mod}}, \mathsf{sk}_{\texttt{mod}}) \leftarrow\$ \mathsf{KGen}(s_1)$
$(\mathsf{pk}_{\texttt{mod},\sigma}, \mathsf{sk}_{\texttt{mod},\sigma}) \leftarrow\$ \mathsf{KGen}(s_2)$
$\mathsf{F} := \emptyset$
$b' := \mathcal{A}^{\mathsf{O}^*}(\mathsf{sk}_{\texttt{mod}}, \mathsf{sk}_{\texttt{mod},\sigma})$
return $b'$
```

```
$\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}(\mathsf{m}_0, \mathsf{m}_1, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}})$
if $\forall \mathsf{id} \in \{\mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}}\}, \mathsf{id} \in \mathsf{corrupted}$ :
    return $\perp$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{amf}}(\mathsf{m}_b, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}})$
$\mathsf{C} := \mathsf{C} \cup \mathsf{c}_{\mathsf{frank}}$
return $\mathsf{c}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{decrypt}}_{\mathsf{conf\text{-}fs}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}})$
if $\mathsf{c}_{\mathsf{frank}} \in \mathsf{C}$ :
    return $\perp$
$\mathsf{m}_{\mathsf{frank}} :=$
    $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}, \mathsf{sk}_{\texttt{plat}})$
return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}(\mathsf{m}_{\mathsf{frank}0}, \mathsf{m}_{\mathsf{frank}1}, \mathsf{id}_{\texttt{fwd}}, \mathsf{id}_{\texttt{rec}})$
if $\forall \mathsf{id} \in \{\mathsf{id}_{\texttt{fwd}}, \mathsf{id}_{\texttt{rec}}\}, \mathsf{id} \in \mathsf{corrupted}$ :
    return $\perp$
$\mathsf{c}_{\mathsf{frank}} := \mathsf{fwd}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}b}, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}})$
return $\mathsf{c}_{\mathsf{frank}}$
```

Figure 12: The security games for Message Confidentiality. Here, we use O^* to denote $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$, $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$, $\mathsf{O}^{\mathsf{decrypt}}$, $\mathsf{O}^{\mathsf{corrupt}}$ and $\mathsf{O}^{\mathsf{request}}$.

corrupted users should be guaranteed confidentiality of all their messages and interactions prior to the time of compromise. In this work, we consider the state of users to consist entirely of their key material and their tokens; ergo, Hecate can only guarantee confidentiality for messages that have been securely deleted from the local device prior to the compromise event.

We provide a combined definition of message confidentiality and forward security in Figure 12. It guarantees message confidentiality because the $\mathsf{CONF\text{-}FS}_{\mathsf{b}}^{\mathcal{A}}$ game requires that content moderation does not break CCA security. Additionally, it guarantees forward security because the adversary in the $\mathsf{CONF\text{-}FS}_{\mathsf{b}}^{\mathcal{A}}$ game is allowed to corrupt any user of their choice, and in particular they can corrupt users who had previously honestly interacted using $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$ and $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$. The game requires that the adversary cannot learn anything about their previously exchanged honest messages, their prior keys, or states.

In this game, we note an important type difference between the franked messages $\mathsf{m}_{\mathsf{frank}}$ returned by $\mathsf{O}^{\mathsf{decrypt}}_{\mathsf{conf\text{-}fs}}$ on one hand, and the franked cipher $\mathsf{c}_{\mathsf{frank}}$ returned by $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$ and $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$ on the other. $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$ and $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$ produce a franked cipher that is handed out to the platform for stamping before it gets relayed back to the receiver. In other words, the platform cannot read any part of $\mathsf{c}_{\mathsf{frank}}$ that is not intended for it. $\mathsf{O}^{\mathsf{decrypt}}_{\mathsf{conf\text{-}fs}}$ returns the franked message after it has been delivered and hence decrypted by the receiver. If a content moderation scheme does not handle the distinction between the franked message and franked cipher properly, by say appending the id of the sender to the envelope, then the adversary should be able to easily win the game.

Theorem 5.5. *Our scheme* Hecate *is message confidential and forward secure. Concretely, for any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{A}'$ and $\mathcal{A}''$ such that:*

$$\mathsf{Adv}^{\mathsf{conf\text{-}fs}}_{\mathsf{Hecate}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{hiding}_{\mathrm{com}}}_{\mathsf{Hecate}}(\mathcal{A}') + \mathsf{Adv}^{\mathrm{enc}_{\mathrm{cca}}}_{\mathsf{Hecate}}(\mathcal{A}'').$$

Informally, the theorem holds because Hecate constructs franked messages by appending tokens to the payload of the message, and by adding a commitment and timestamp to its envelope (see $\mathsf{construct}_{\mathsf{frank}}$ and $\mathsf{stamp}_{\mathsf{time}}$ in Figure 4). The tokens are encrypted alongside the plaintext message. The identifying content of the commitment is encrypted, and we rely on the hiding properties of the commitment scheme and the security of the connection that exists between parties and the platform. No part of a Hecate franked message can therefore break this security property. We prove this theorem in detail in what follows.

Proof of Thm. 5.5. In this proof, we show that Hecate does not break the CCA security of the underlying cryptographic scheme.

Game$_0$: We start with $\mathsf{CONF\text{-}FS}_{0}^{\mathcal{A}}$.

Game$_1$: In $\mathsf{send}_{\mathsf{amf}}$ (called by $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$), we replace m_0 with m_1 in Frank on line 4. Since the adversary does not have access to r, then they cannot only decommit com with negligible property equal to breaking

Com. Hiding

ENC-CCA

```
send_0
1 : $\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src}})$
2 : fetch token from $\mathsf{state}_{\mathtt{src}}$
3 : $\mathsf{m}_{\mathsf{frank}\,0} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}_0,$
4 :         $\mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
5 : $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}\,0})$
6 : return $\mathsf{c}_{\mathsf{frank}}$
```

```
send_1
1 : $\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src}})$
2 : fetch token from $\mathsf{state}_{\mathtt{src}}$
3 : $\mathsf{m}_{\mathsf{frank}\,0} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}_1,$
4 :         $\mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
5 : $\mathsf{c}_{\mathsf{frank}\,0} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}\,0})$
6 : return $\mathsf{c}_{\mathsf{frank}\,0}$
```

```
send_2
$\mathsf{state}_{\mathtt{src}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{src}})$
fetch token from $\mathsf{state}_{\mathtt{src}}$
$\mathsf{m}_{\mathsf{frank}\,1} := \mathsf{Frank}(\mathsf{state}_{\mathtt{src}}, \mathsf{m}_1,$
        $\mathsf{id}_{\mathtt{rec}}, \mathsf{token})$
$\mathsf{c}_{\mathsf{frank}\,1} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}\,1})$
return $\mathsf{c}_{\mathsf{frank}\,1}$
```

Com. Hiding

ENC-CCA

```
fwd_3
$\mathsf{state}_{\mathtt{fwd}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd}})$
$\mathsf{m}_{\mathsf{frank}'_0} := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd}},$
        $\mathsf{m}_{\mathsf{frank}\,0}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{c}_{\mathsf{frank}\,0} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}'_0})$
return $\mathsf{c}_{\mathsf{frank}\,0}$
```

```
fwd_4
$\mathsf{state}_{\mathtt{fwd}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd}})$
$\mathsf{m}_{\mathsf{frank}'_0} := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd}},$
        $\mathsf{m}_{\mathsf{frank}\,1}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{c}_{\mathsf{frank}\,0} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}'_0})$
return $\mathsf{c}_{\mathsf{frank}\,0}$
```

```
fwd_5
$\mathsf{state}_{\mathtt{fwd}} := \mathsf{retrieve}_{\mathsf{state}}(\mathsf{id}_{\mathtt{fwd}})$
$\mathsf{m}_{\mathsf{frank}'_1} := \mathsf{Forward}(\mathsf{state}_{\mathtt{fwd}},$
        $\mathsf{m}_{\mathsf{frank}\,1}, \mathsf{id}_{\mathtt{rec}})$
$\mathsf{c}_{\mathsf{frank}\,1} := \mathsf{send}_{\mathsf{eems}}(\mathsf{m}_{\mathsf{frank}'_1})$
return $\mathsf{c}_{\mathsf{frank}\,1}$
```

Figure 13: The hybrid steps of the $\mathsf{CONF\text{-}FS}^{\mathcal{A}}_{\mathsf{b}}$ game

the secrecy of the commitment scheme. We construct $\mathcal{B}$ that can uses an $\mathcal{A}$ that can distinguish between both games, to break the secrecy of the commitment scheme. When $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{send}}_{\mathsf{b}}$ on messages m_0 and m_1, $\mathcal{B}$ queries $\mathsf{O}^{\mathsf{com\text{-}hiding}}$ with both messages and uses the returned com to construct the franked cipher on behalf of the moderator and the users $\mathsf{c}_{\mathsf{frank}}$. If $\mathcal{A}$ succeeds in picking a choice bit, then $\mathcal{B}$ will pick the same choice bit in $\mathsf{Com}^{\mathcal{A}}_{\mathsf{hiding}}$ and will win with at least the same advantage.

Game$_2$: In $\mathsf{send}_{\mathsf{amf}}$ (called by $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$), we replace $\mathsf{m}_{\mathsf{frank}\,0}$ with $\mathsf{m}_{\mathsf{frank}\,1}$ in Frank. Since the adversary does not have access to the source $\mathsf{id}_{\mathtt{src}}$'s encryption secret keys, then they can only notice this change with advantage equal to breaking the CCA security $\mathsf{ENC}^{\mathcal{A}}_{\mathsf{cca}}$ of $\mathsf{send}_{\mathsf{eems}}$ and decrypting the contents of $\mathsf{c}_{\mathsf{frank}}$. When $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{send}}_{\mathsf{conf\text{-}fs,b}}$ on messages $\mathsf{m}_{\mathsf{frank}\,0}$ and $\mathsf{m}_{\mathsf{frank}\,1}$, $\mathcal{B}$ queries $\mathsf{O}^{\mathsf{enc}}_{\mathsf{cca}}$ with both messages and uses the returned cipher text $\mathsf{c}_{\mathsf{frank}}$ to construct the franked message on behalf of the moderator and the users. When $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{decrypt}}$ on ciphertext c, $\mathcal{B}$ behaves similarly and requests the decryption of the cipher-text from $\mathsf{O}^{\mathsf{decrypt}}_{\mathsf{cca}}$. If $\mathcal{A}$ succeeds in picking a choice bit, then $\mathcal{B}$ will pick the same choice bit in $\mathsf{ENC}^{\mathcal{A}}_{\mathsf{cca}}$ and will win with at least the same advantage.

Game$_3$: In $\mathsf{fwd}_{\mathsf{amf}}$ (called by $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$), we replace $\mathsf{m}_{\mathsf{frank}\,0}$ with $\mathsf{m}_{\mathsf{frank}\,1}$ in $\mathsf{Forward}$ on line 4. When a message is forwarded in Hecate, the commitment com in the original franked message is replaced with a random commitment and the adversary cannot distinguish this change.

Game$_4$: In $\mathsf{fwd}_{\mathsf{amf}}$ (called by $\mathsf{O}^{\mathsf{fwd}}_{\mathsf{conf\text{-}fs,b}}$), we replace $\mathsf{m}_{\mathsf{frank}'_0}$ with $\mathsf{m}_{\mathsf{frank}'_1}$ in $\mathsf{Forward}$. Since the adversary does not have access to the source $\mathsf{id}_{\mathtt{src}}$'s encryption secret keys, then they can only notice this change with advantage equal to breaking the CCA security of $\mathsf{send}_{\mathsf{eems}}$ and decrypting the contents of $\mathsf{c}_{\mathsf{frank}}$. The proof is similar to the on in Game$_1$.

The resulting $\mathsf{CONF\text{-}FS}^{\mathcal{A}}_0$ is identical to $\mathsf{CONF\text{-}FS}^{\mathcal{A}}_1$ and the adversary can only win the $\mathsf{CONF\text{-}FS}^{\mathcal{A}}_{\mathsf{b}}$ game with advantage equal to twice the advantage of breaking the CCA security of the underlying encryption scheme and the hiding property of the commitment scheme. □

5.4 Unforgeability and Accountability

These properties describe a scheme's ability to *bind senders to well-formed messages while guaranteeing that no user can be accused of sending a message that they did not send.* They go hand-in-hand because well-

```
$\mathsf{ACC}^{\mathcal{A}}$
1 :  $s \leftarrow\$ \mathcal{A}$
2 :  $(\mathsf{pk}_{\texttt{mod}}, \mathsf{sk}_{\texttt{mod}}) \leftarrow\$ \mathsf{KGen}(1^n)$
3 :  $(\mathsf{pk}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}}) \leftarrow\$ \mathsf{KGen}(s)$
4 :  $\mathsf{report} \leftarrow\$ \mathcal{A}^{\mathsf{O}^*}(\mathsf{sk}_{\texttt{plat}})$
5 :  $(\mathsf{id}, \mathsf{time}, \mathsf{m}) := \mathsf{Inspect}(\mathsf{m}_{\mathsf{frank}}, \mathsf{sk}_{\texttt{mod}})$
6 :  if $\mathsf{id} \notin \{\bot, \mathsf{corrupted}\} \land \mathsf{m} \notin \mathsf{M}$ :
7 :      return 1
8 :  return 0
```

```
$\mathsf{O}^{\mathsf{deliver}}_{\mathsf{acc}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}})$
1 :  $\mathsf{m}_{\mathsf{frank}} :=$
2 :    $\mathsf{receive}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}})$
3 :  return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{send}}_{\mathsf{acc}}(\mathsf{m}, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}})$
1 :  $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{amf}}(\mathsf{m}, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}})$
2 :  $\mathsf{m}_{\mathsf{frank}} :=$
3 :    $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}, \mathsf{sk}_{\texttt{plat}})$
4 :  $\mathsf{F} := \mathsf{F} \cup \mathsf{m}_{\mathsf{frank}}$
5 :  return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}^{\mathsf{fwd}}_{\mathsf{acc}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\texttt{src}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}_{\texttt{plat}}, \mathsf{sk}_{\texttt{plat}})$
1 :  $\mathsf{c}_{\mathsf{frank}} := \mathsf{fwd}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\texttt{fwd}}, \mathsf{id}_{\texttt{rec}})$
2 :  $\mathsf{m}_{\mathsf{frank}}' :=$
3 :    $\mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\texttt{rec}}, \mathsf{time}, \mathsf{sk}_{\texttt{plat}})$
4 :  return $\mathsf{m}_{\mathsf{frank}}'$
```

Figure 14: The security games for Accountability. Here, we use O^* to denote $\mathsf{O}^{\mathsf{send}}$,$\mathsf{O}^{\mathsf{fwd}}$, $\mathsf{O}^{\mathsf{deliver}}$,$\mathsf{O}^{\mathsf{request}}$ and $\mathsf{O}^{\mathsf{corrupt}}$.

formed messages are necessarily bound to their original sender and cannot be attributed to anyone else. Note that these properties should hold with respect to an honest moderator who handles source tracing reported messages. Ergo, in the unforgeability and accountability game, the adversary attempts to create a message and fool the moderator into believing that it came from a different user.

We model accountability via the $\mathsf{ACC}^{\mathcal{A}}$ security game in Figure 14. The adversary starts by making polynomially many queries to $\mathsf{O}^{\mathsf{send}}$, $\mathsf{O}^{\mathsf{fwd}}$, $\mathsf{O}^{\mathsf{deliver}}$, $\mathsf{O}^{\mathsf{corrupt}}$, and $\mathsf{O}^{\mathsf{request}}$ that collectively allow $\mathcal{A}$ to build any message path of their choice and receive the resulting franked messages at each node within that path. Afterward, the adversary is tasked with producing a franked message that: (1) can be reported back to an existing user (line 5 in $\mathsf{ACC}^{\mathcal{A}}$) (2) traces back to an uncorrupted party (line 6), and (3) was not previously created during the challenge phase (line 6). In producing a message that can pass these predicates, the adversary can effectively produce new messages that can be traced back to other users. Note that who the message traces back to, beyond being an existing honest user, is inconsequential: if the adversary cannot find anyone else to blame them for a message, regardless of who they are, then they cannot avoid accountability.

Theorem 5.6. Hecate *holds users accountable. For any PPT adversary $\mathcal{A}$ that makes at most q queries to its $\mathsf{O}^{\mathsf{send}}$ oracle, there exist PPT adversaries $\mathcal{A}'$ and $\mathcal{A}''$ such that:*

$$\mathsf{Adv}^{\mathsf{acc}}_{\mathsf{Hecate}}(\mathcal{A}) \leq (q+1) \cdot \mathsf{Adv}^{\mathrm{sig_{eu\text{-}cma}}}_{\mathcal{S}}(\mathcal{A}') + \mathsf{Adv}^{\mathrm{hash_{coll}}}_{\mathcal{H}}(\mathcal{A}'').$$

Proof of Thm. 5.6. We show how the $\mathsf{ACC}^{\mathcal{A}}$ game can only be won in Hecate with negligible probability. We note that the game's win condition requires a well-formed $\mathsf{m}_{\mathsf{frank}}$ that traces back to an existing id (line 6) and that contains a plain-text message that was not previously submitted and stored in $\mathsf{state}_{\mathsf{chal}}$ (line 6). $\mathsf{ACC}^{\mathcal{A}}$ verifies the message integrity via Verify (on line 5), which returns an id in the case of a well formed message and $\bot$ otherwise. Verify is composed of vfMsg (as seen in Figure 3 of Hecate's construction) which itself makes three separate calls to vfExp, vfCom and vfToken (line 1-2) based on which it determines the validity of the message. For a franked message $\mathsf{m}_{\mathsf{frank}}$ to be viable, all three functions must evaluate to true. vfExp and vfCom are envelope commitment verification steps and are hence inconsequential for the accountability property. It's sufficient for the sake of this proof to show that vfToken can never return true in Hecate. We handle the other two verification checks in backward security.

In order for vfToken to return true, all three clauses lines (2-4) must also evaluate to true.

Game$_0$: This game is equivalent to vfToken, since this is the only method in vfMsg (Inspect's instantiation in Hecate) that is relevant for Hecate.

Game$_1$: We replace line 3 of vfToken with a new boolean predicate $b := \mathsf{token} \overset{?}{\in} \mathsf{T}$ that checks whether

$$\begin{tikzcd}\mathsf{vfToken}_0(n) \arrow[r, bend left, "\text{EU-CMA}"] & \mathsf{vfToken}_1(n) \arrow[r, bend left, "\text{q.EU-CMA}"] & \mathsf{vfToken}_2(n) \arrow[r, bend left, "\text{col. resist.}"] & \mathsf{vfToken}_3(n)\end{tikzcd}$$

```
vfToken_0(n)
1 :  $b_1 := (reveal \overset{?}{=} \mathsf{H}(\mathsf{m}))$
2 :  $b_2 := \mathsf{Vf}_{\mathsf{pk}_{\mathsf{mod}}}(\mathsf{x}_1 \| \mathsf{pk}_\mathsf{e} \| \mathsf{t}_1, \sigma_1)$
3 :  $b_3 := \mathsf{Vf}_{\mathsf{pk}_\mathsf{e}}(\mathsf{x}_2, \sigma_2)$
4 :  return $b_1 \wedge b_2 \wedge b_3$
```

```
vfToken_1(n)
$b_1 := (reveal \overset{?}{=} \mathsf{H}(\mathsf{m}))$
$b := \mathsf{token} \overset{?}{\in} \mathsf{T}$
$b_3 := \mathsf{Vf}_{\mathsf{pk}_\mathsf{e}}(\mathsf{x}_2, \sigma_2)$
return $b_1 \wedge b \wedge b_3$
```

```
vfToken_2(n)
$b_1 := (reveal \overset{?}{=} \mathsf{H}(\mathsf{m}))$
$b := \langle t, \mathsf{token}, * \rangle \overset{?}{\in} \mathsf{F}$
return $b_1 \wedge b$
```

```
vfToken_3(n)
$b := \mathsf{m}_{\mathsf{frank}} \overset{?}{\in} \mathsf{F}$
return $b$
```

Figure 15: The hybrid steps modifying $\mathsf{vfToken}$ in the Accountability game

a token was generated by $\mathsf{O}^{\mathsf{request}}$, and hence by a legitimate call to TGen. Note that the only available way for the adversary to retrieve pre-processing tokens is through a call to $\mathsf{O}^{\mathsf{request}}$ with the id of a corrupted party and that every such token is stored in T on line 5. Since the adversary does not have access to the moderator's secret key, then $\mathcal{A}$ has a negligible advantage in distinguishing this change by breaking the EU-CMA of the underlying signature scheme, forging the signature of the moderator and creating their own token without using $\mathsf{O}^{\mathsf{request}}$. We formally show this by assuming that there exists an adversary $\mathcal{A}$ that can distinguish between Games 0 and 1, and showing how to construct an adversary $\mathcal{B}$ that breaks EU-CMA. We primarily show how $\mathcal{B}$ constructs $\mathsf{O}^{\mathsf{request}}$, all other oracles outputs can be trivially generated by $\mathcal{B}$ in Hecate since they are ephemeral and only depend on the pre-processing token and additionally do not depend on the moderator's secret key. When $\mathcal{A}$ requests a pre-processing token, $\mathcal{B}$ locally construct x_1, t_1 and $(\mathsf{pk}_\mathsf{e}, \mathsf{sk}_\mathsf{e})$, then signs their concatenation by making a query to $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$. $\mathcal{B}$ then sends the resulting token to $\mathcal{A}$ and waits until $\mathcal{A}$ submits an $\mathsf{m}_{\mathsf{frank}}$. $\mathcal{B}$ additionally checks franked messages delivered using $\mathsf{O}^{\mathsf{deliver}}_{\mathsf{acc}}$ that pass the verification step and the predicates on line 6 in the accountability game. If $\mathcal{A}$ does not fail, $\mathcal{B}$ can strip $\mathsf{m}_{\mathsf{frank}}$ of everything except the moderator's pre-processring signature σ_1 and submit that to the EU-CMA game. Since this $\mathcal{A}$ is required to submitted a new franked message originating from uncorrupted parties, then the moderator signature in $\mathsf{m}_{\mathsf{frank}}$ will not correspond to any of outputs of $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$, and $\mathcal{B}$ will win the EU-CMA games when $\mathcal{A}$ does.

Game$_2$: Let $t := (\mathsf{x}_2, \sigma_2)$, i.e. the subset of the franked message constructed during the online stage. We replace and combine lines 3 and 4 with the boolean predicate $b := (\langle t, \mathsf{token}, * \rangle \overset{?}{\in} \mathsf{F})$ that checks that the immutable tuple $\langle t, \mathsf{token} \rangle$ is a subset of some franked message in F. Recall that $\mathsf{O}^{\mathsf{send}}_{\mathsf{acc}}$ is the adversary's only way to instruct honest parties to construct and send messages. All such messages are saved in F on line 4. Since the adversary does not have access to honest parties' ephemeral keys, then $\mathcal{A}$ has negligible advantage in distinguishing between the original signature verification and the predicate we replaced it with by breaking the EU-CMA of the underlying signature scheme, forging the signature of the an honest party and creating their own franked message originating from that user without using $\mathsf{O}^{\mathsf{send}}_{\mathsf{acc}}$. We formally show this by assuming that there exists an adversary $\mathcal{A}$ that can distinguish between Games 1 and 2, and showing how to construct an adversary $\mathcal{B}$ that breaks EU-CMA. Let q be an upper bound on the number of queries $\mathcal{A}$ can make to $\mathsf{O}^{\mathsf{send}}_{\mathsf{acc}}$. We construct an $\mathcal{B}$ that has access to q different EU-CMA games (with their own $\mathsf{O}^{\mathsf{sign}}_{\mathsf{eu-cma}}$) for q ephemeral key pairs (where $q \in \mathsf{poly}(\lambda)$) and that will try to win at least one of these games. $\mathcal{B}$ starts by sampling and fixing the moderator's secret key and uses it to sign pre-processing tokens. When $\mathcal{A}$ queries $\mathsf{O}^{\mathsf{send}}_{\mathsf{acc}}$, $\mathcal{B}$ picks one of the unused q ephemeral public keys, constructs the pre-processing token for that public key by randomly sampling its secret ephemeral key, and generates the rest of the franked message by asking the $\mathsf{O}^{\mathsf{sign}}_{\mathsf{eu-cma}}$ associated with that public key to sign x_2. Note that $\mathcal{B}$ can trivially generate all other fields in the franked message since they have access to the moderator's secret key. Additionally, $\mathcal{B}$ can entirely act on behalf of the moderator when $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{request}}$. $\mathcal{B}$ then waits until $\mathcal{A}$ submits an franked messages or delivers a well formed $\mathsf{m}_{\mathsf{frank}}$ using $\mathsf{O}^{\mathsf{deliver}}_{\mathsf{acc}}$ that pass predicates on line 6. $\mathcal{B}$ can win the EU-CMA game $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$ by stripping that $\mathsf{m}_{\mathsf{frank}}$ of everything except its online signature σ_2 and its ephemeral public key pk_e that it submits to the corresponding $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$ game. If $\mathcal{A}$ succeeds at distinguishing Games 1 and 2 using $\mathsf{m}_{\mathsf{frank}}$, then $\mathcal{B}$ will win $\mathsf{Sig}^{\mathcal{A}}_{\mathsf{eu-cma}}$ since $\mathsf{m}_{\mathsf{frank}}$ will need to be a new franked message originating from an

```
$\mathsf{BAC}_{\delta}^{\mathcal{A}}$
1 :  $(\mathsf{pk}_{\mathtt{mod}}, \mathsf{sk}_{\mathtt{mod}}) \leftarrow\$ \mathsf{KGen}(1^n)$
2 :  $(\mathsf{pk}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}}) \leftarrow\$ \mathsf{KGen}(1^n)$
3 :  $\mathcal{A}^{\mathsf{O}^*}$
4 :  $\mathsf{corrupted} := \emptyset, \mathsf{F} := \emptyset$
5 :  $\mathsf{m}_{\mathsf{chal}} \leftarrow\$ \mathcal{M}$
6 :  $\mathsf{recovery}_{\mathsf{start,t}} := \mathsf{global}_{\mathsf{t}}$
7 :  $\mathsf{global}_{\mathsf{t}} := \mathsf{global}_{\mathsf{t}} + \delta$
8 :  $\mathsf{recovery}_{\mathsf{delay,t}} := \mathsf{global}_{\mathsf{t}}$
9 :  $\mathsf{m}_{\mathsf{frank}} \leftarrow \mathcal{A}^{\mathsf{O}^*}(\mathsf{m}_{\mathsf{chal}})$
10 : if $\mathsf{check}_{\mathsf{report}}(\mathsf{m}_{\mathsf{frank}})$ :
11 :   return 1
12 : return 0
```

```
$\mathsf{check}_{\mathsf{report}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{m}_{\mathsf{chal}})$
1 :  $(\mathsf{m}, \mathsf{report}) := \mathsf{Verify}(\mathsf{m}_{\mathsf{frank}})$
2 :  $(\mathsf{id}_{\mathtt{src}}, \mathsf{time}, \mathsf{m}) := \mathsf{Inspect}(\mathsf{report})$
3 :  if $\mathsf{id}_{\mathtt{src}} \overset{?}{\neq} \bot \land (\mathsf{id}_{\mathtt{src}}, *) \overset{?}{\notin} \mathsf{corrupted}$
4 :    $\land\ \mathsf{m}_{\mathsf{frank}} \notin \mathsf{F}$ :
5 :      if $\mathsf{m} \overset{?}{=} \mathsf{m}_{\mathsf{chal}}$
6 :        $\lor\ \mathsf{time} > \mathsf{recovery}_{\mathsf{delay,t}}$ :
7 :            return 1
8 :  return 0
```

```
$\mathsf{O}_{\mathsf{bs}}^{\mathsf{deliver}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}})$
1 :  $\mathsf{m}_{\mathsf{frank}} :=$
2 :    $\mathsf{receive}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}_{\mathtt{plat}}, \mathsf{sk}_{\mathtt{plat}})$
3 :  return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}_{\mathsf{bs}}^{\mathsf{send}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
1 :  $\mathsf{c}_{\mathsf{frank}} := \mathsf{send}_{\mathsf{amf}}(\mathsf{m}, \mathsf{id}_{\mathtt{src}}, \mathsf{id}_{\mathtt{rec}})$
2 :  $\mathsf{m}_{\mathsf{frank}} := \mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
3 :  $\mathsf{F} := \mathsf{F} \cup \mathsf{m}_{\mathsf{frank}}$
4 :  return $\mathsf{m}_{\mathsf{frank}}$
```

```
$\mathsf{O}_{\mathsf{bs}}^{\mathsf{fwd}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{fwd}}, \mathsf{id}_{\mathtt{rec}})$
1 :  $\mathsf{c}_{\mathsf{frank}} := \mathsf{fwd}_{\mathsf{amf}}(\mathsf{m}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{fwd}}, \mathsf{id}_{\mathtt{rec}})$
2 :  $\mathsf{m}_{\mathsf{frank}}' := \mathsf{receive}_{\mathsf{amf}}(\mathsf{c}_{\mathsf{frank}}, \mathsf{id}_{\mathtt{rec}}, \mathsf{time}, \mathsf{sk}_{\mathtt{plat}})$
3 :  return $\mathsf{m}_{\mathsf{frank}}'$
```

Figure 16: The security games for Backward Security, where O^* denotes $\mathsf{O}^{\mathsf{send}}$, $\mathsf{O}^{\mathsf{fwd}}$, $\mathsf{O}^{\mathsf{deliver}}$, $\mathsf{O}^{\mathsf{corrupt}}$ and $\mathsf{O}^{\mathsf{request}}$.

uncorrupted party and could not have have been generated by $\mathsf{O}_{\mathsf{eu-cma}}^{\mathsf{sign}}$. Since each $\mathsf{Sig}_{\mathsf{eu-cma}}^{\mathcal{A}}$ game is defined by a separate pair of ephemeral keys generated i.i.d., then these games are independent of one another and by the union bound the advantage of the adversary is at most equal to $q \times \mathsf{Adv}_{\mathcal{A}}^{\mathsf{sig}_{\mathsf{eu-cma}}}(n)$.

Game$_3$: Finally, we can replace and combine all three lines in $\mathsf{vfToken}$ with the boolean predicate $b := \langle \mathsf{token}, t, m \rangle \overset{?}{\in} \mathsf{M}$, which is in essence equivalent to $b := \mathsf{m}_{\mathsf{frank}} \overset{?}{\in} \mathsf{M}$. The adversary can distinguish the change made with negligible advantage equal to the likelihood of finding a collision $m' \notin \mathsf{M}$ of the collision resistant hash function H, such that $\exists \mathsf{m}_{\mathsf{frank}}.m \in \mathsf{M}, \mathsf{H}(m) = \mathsf{H}(m')$. Let's assume that there exists an adversary $\mathcal{A}$ that can distinguish between Games 2 and 3. We construct an adversary $\mathcal{B}$ that will try to win the collision resistance hash game. When $\mathcal{A}$ queries $\mathsf{O}_{\mathsf{acc}}^{\mathsf{send}}$, $\mathcal{B}$ requests the hash of m from $\mathsf{O}^{\mathsf{hash}}$ of the collision resistance hash game, then constructs the rest of the franked message honestly. Note that all other oracles can be run by $\mathcal{B}$ since they act on behalf of the moderator and honest users. $\mathcal{B}$ then waits until $\mathcal{A}$ submits an franked messages or delivers a well formed $\mathsf{m}_{\mathsf{frank}}$ using $\mathsf{O}_{\mathsf{acc}}^{\mathsf{deliver}}$ that pass predicates on line 6. If $\mathcal{A}$ succeeds at distinguishing Games 2 and 3 using $\mathsf{m}_{\mathsf{frank}}$, then $\mathcal{B}$ will win the collision resistance hash game since the only way $\mathcal{A}$ can distinguish between both games is if finds a collision m' of $\mathsf{H}(\mathsf{m})$ that was not stored in M.

Now notice that there can be no franked message $\mathsf{m}_{\mathsf{frank}}$ that can satisfy the hybrid predicate $\mathsf{m}_{\mathsf{frank}} \overset{?}{\in} \mathsf{M}$ and the winning condition of the game $\mathsf{id} \notin \{\bot, \mathsf{corrupted}\} \land \mathsf{m}_{\mathsf{frank}}.\mathsf{m} \notin \mathsf{M}$, since we have shown that the adversary has to necessarily submit a stored franked message in M. The adversary cannot therefore construct a franked message $\mathsf{m}_{\mathsf{frank}}$ that can win this game. We can reduce the attacker's advantage in winning the accountability game to the sum of its advantage in breaking the collision-resistance or $(q+1)$ the EU-CMA games. □

5.5 Backward Security

Backward Security requires that *an adversary who controlled the state and keys of a device pre-compromise should not be able to benefit from them after device recovery.* In particular, the adversary should be unable to craft new messages from a recovered user or claim that during-compromise messages were sent out (not forwarded) after the compromise period.

5.5.1 Formalizing the game's objectives

To codify backward security, we design the game $\mathsf{BAC}_{\delta}^{\mathcal{A}}$ in Figure 16. We provide the adversary with oracle access to $\mathsf{O}^{\mathsf{corrupt}}$, $\mathsf{O}^{\mathsf{request}}$, $\mathsf{O}^{\mathsf{send}}$, $\mathsf{O}^{\mathsf{fwd}}$, $\mathsf{O}^{\mathsf{deliver}}$ that allow them to perform any possible interaction between users.

Similarly to prior games, the adversary can corrupt users using $\mathsf{O}^{\mathsf{corrupt}}$ and request their pre-processing tokens via $\mathsf{O}^{\mathsf{request}}$. However contrary to prior games, handling time in $\mathsf{BAC}_{\delta}^{\mathcal{A}}$ is necessary since backward security is a property of the corruption recovery period. In $\mathsf{BAC}_{\delta}^{\mathcal{A}}$, corruption is not indefinite and we define it with respect to the global time variable $\mathsf{global_t}$. When $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{corrupt}}$, the oracle stores the id of the corrupted user along with the time of corruption at $\mathsf{global_t}$ in the set $\mathsf{corrupted}$. When $\mathcal{A}$ calls $\mathsf{O}^{\mathsf{request}}$ to request pre-processing tokens for corrupted users, the oracle first checks that the requested user is in $\mathsf{corrupted}$ at the current global time and increments the global time if that check succeeds. In other words, the global time is incremented each time a pre-processing token is returned to $\mathcal{A}$, effectively requiring them to re-corrupt users at the new global time. By defining time in this manner, we capture how an adversary can no longer impersonate recovered users and request tokens on their behalf.

The adversary can additionally send and forward messages between any two users regardless of their corruption status via $\mathsf{O}^{\mathsf{send}}$, $\mathsf{O}^{\mathsf{fwd}}$ and $\mathsf{O}^{\mathsf{deliver}}$. $\mathsf{O}^{\mathsf{send}}$ in particular grants the adversary the power to request that an honest user creates a new franked message that the oracle stores in the set F. Otherwise, with access to $\mathsf{O}^{\mathsf{request}}$, the adversary can create franked messages on their own before sending/forwarding them along. $\mathsf{O}^{\mathsf{fwd}}$ allows them to forward a franked message between honest users, and $\mathsf{O}^{\mathsf{deliver}}$ allows them to send/forward messages from a corrupted user to an honest one. With access to all these oracles, we model an all powerful adversary that can fully control and build a forwarding tree of their choice.

The $\mathsf{BAC}_{\delta}^{\mathcal{A}}$ games proceeds in three phases.

The first phase marks the pre-recovery phase where the adversary is given access to all aforementioned oracles and can interact with users as they see fit (line 3).

When the adversary is done, the second phase begins and the game master sets up the post-recovery period by: (1) sampling a challenge $\mathsf{m_{chal}}$ uniformly at random from the message space excluding plain-text messages sent in the first phase (line 5), (2) saving the time at which the first phase ended in $\mathsf{recovery_{start,t}}$ (line 6), (3) and advancing the current time by a grace period delta to mark the delayed beginning of the recovery period $\mathsf{recovery_{delay,t}}$ (line 8).

The final phase marks the post-recovery period when the adversary is given $\mathsf{m_{chal}}$ and asked to provide a franked message $\mathsf{m_{frank}}$ that is well formed, traces back to an honest user and either: (1) contains the plain-text challenge or (2) is timestamped after the the delayed recovery period, after they are done interacting with the provided oracles. The former two possibilities provided to the adversary encompass this works notion of backward security: the adversary should neither (1) be able to produce a message that they did not think of during compromise, (2) nor should they circulate a message that was not timestamped during the time of compromise. The later point essentially captures how abuse reporting systems may deem messages that were sent during known data breach periods as unaccountable and possibly invalid. This is especially important in an era where the number of cyber-attack campaigns is on a rise, and where the adversary may attempt to create many messages during the compromise period and delay reporting them or further circulating them until an opportune time after the recovery period begins. This is especially critical in the case where a public official's device is hacked and when the time at which their leaked messages were sent can influence public opinion.

Note that the game master resets both the $\mathsf{corrupted}$ and F sets on line 4 since they are not useful in the first phase when the recovery period has not yet begun. Both sets will allow the game master to evaluate the response of $\mathcal{A}$ with respect to the winning condition. Since backward security is a property of corruption recovery, the winning condition is itself a function of that period. In this essence, we define recovery after some fixed δ has passed on line 8, to model how recovery in practice is not instantaneous and may require a grace period.

Theorem 5.7. Hecate *is backward secure. For any PPT adversary $\mathcal{A}$, there exist PPT $\mathcal{A}'$, $\mathcal{A}''$, and $\mathcal{A}'''$*

EU-CMA

```
$\mathsf{vfMsg}_0(n)$
    $b_1 := \mathsf{vfToken}(\mathsf{token}, \mathsf{m})$
    $b_2 := \mathsf{vfCom}(\mathsf{token})$
    $b_3 := \mathsf{vfExp}(\mathsf{token})$
    return $b_1 \wedge b_2 \wedge b_3$
```

```
$\mathsf{vfMsg}_1(n)$
    $b := (\mathsf{token}, *) \overset{?}{\in} \mathsf{F} \vee \mathsf{token} \overset{?}{\in} \mathsf{T}$
    $b_2 := \mathsf{vfCom}(\mathsf{token})$
    $b_3 := \mathsf{vfExp}(\mathsf{token})$
    return $b \wedge b_2 \wedge b_3$
```

Accountability

Commit. Bind.

```
$\mathsf{vfMsg}_2(n)$
    $b := (\mathsf{token}, *) \overset{?}{\in} \mathsf{F}$
    $b_2 := \mathsf{vfCom}(\mathsf{token})$
    return $b \wedge b_2$
```

```
$\mathsf{vfMsg}_3(n)$
    $b := (\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, *) \overset{?}{\in} \mathsf{F}$
    $b_2 := \mathsf{vfCom}(\mathsf{token})$
    return $b \wedge b_2$
```

```
$\mathsf{vfMsg}_4(n)\ \boxed{\mathsf{vfMsg}_5(n)}$
    $b := \mathsf{m}_{\mathsf{frank}} \overset{?}{\in} \mathsf{F}$
    return $b$
```

```
$\mathsf{vfCom}_3(n)$
    $b_1 := \mathsf{Vf}(\mathsf{x}_1 \| \mathsf{x}_2, \mathsf{com}, \mathsf{r})$
    $b_2 := \mathsf{Vf}_{\mathsf{pk}_{plat}}(\mathsf{com} \| \mathsf{t}_2, \sigma_3)$
    return $b_1 \wedge b_2$
```

```
$\mathsf{vfCom}_4(n)$
    $b := (\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, \mathsf{com}, *) \in \mathsf{F}$
    $b_2 := \mathsf{Vf}_{\mathsf{pk}_{plat}}(\mathsf{com} \| \mathsf{t}_2, \sigma_3)$
    return $b \wedge b_2$
```

```
$\mathsf{vfCom}_5(n)$
    $b := \mathsf{m}_{\mathsf{frank}} \in \mathsf{F}$
    return $b$
```

Figure 17: The hybrid steps modifying vfMsg and vfCom in the Backward Security game. We use the shorthand token to refer to $\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}.\mathsf{token}$ and use the regular expression wildcard operator ”$*$” throughout.

such that:

$$\mathsf{Adv}^{\mathsf{bac}}_{\mathsf{Hecate}}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{acc}}_{\mathsf{Hecate}}(\mathcal{A}') + \mathsf{Adv}^{\mathrm{sig}_{\text{eu-cma}}}(\mathcal{A}'') + \mathsf{Adv}^{\mathrm{binding}_{\mathrm{com}}}(\mathcal{A}''').$$

Proof of Thm. 5.7. The winning condition of the game requires the chosen $\mathsf{m}_{\mathsf{frank}}$ to trace back to an honest user. $\mathsf{m}_{\mathsf{frank}}$ may then either contain the challenge $\mathsf{m}_{\mathsf{chal}}$ chosen by the game master, or be timestamped after the recovery period delay $\mathsf{recovery}_{\mathsf{delay,t}}$ on line 8. The game requires $\mathsf{m}_{\mathsf{frank}}$ to be traceable by the moderator via Inspect in $\mathsf{check}_{\mathsf{report}}$. In Hecate, Inspect is a conjunction of three separate verification steps: vfExp that checks the expiry of the different time components of the franked message, vfCom that checks the envelope commitment, and vfToken which checks all other parts of the franked message and which we discuss in depth in the accountability game.

In this proof we distinguish between the payload of the franked message that we denote by $\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}$ and its envelope $\mathsf{m}_{\mathsf{frank}}.\mathsf{envelope}$. In what follows, $\mathsf{m}_{\mathsf{frank}}$ is equivalent to ($\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}$, $\mathsf{m}_{\mathsf{frank}}.\mathsf{envelope}$), $\mathsf{m}_{\mathsf{frank}}.\mathsf{envelope}$ is equivalent to (com, σ_2, t_2). We use the regular expression operator $*$ to denote that a field can take any value and is not specified by a specific game reduction. Each game hybrid will allow us to specify different pieces of the eventual franked message.

Game$_1$: We replace vfToken with checking that $\mathsf{m}_{\mathsf{frank}}$'s preprocessing token is either $(\mathsf{token}, *) \overset{?}{\in} \mathsf{F}$ or $\mathsf{token} \overset{?}{\in} \mathsf{T}$ (where token is a shorthand for $\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}.\mathsf{token}$). The adversary can distinguish this change with negligible probability equal to the probability of winning the EU-CMA game since the adversary does not have access to the moderator's secret key and has therefore a negligible chance in constructing well formed tokens for a user of their choice locally. Recall that T is the set of corrupted users tokens constructed by $\mathsf{O}^{\mathsf{request}}$ and that $(\mathsf{token}, *) \in \mathsf{F}$ refers to the set of honest user tokens constructed by $\mathsf{O}^{\mathsf{send}}_{\mathsf{bs}}$. We refer the reader to Game 1 in the accountability game for more details.

Game$_2$: We parametrize $\mathsf{expiry} := \delta$ and replace vfExp with $(\mathsf{token}, *) \overset{?}{\in} \mathsf{F}$, i.e. we drop the $\mathsf{token} \overset{?}{\in} \mathsf{T}$ from the disjunction in Game_1. vfExp requires that $|\mathsf{t}_2 - \mathsf{t}_1| < \mathsf{expiry}$, i.e. that $\mathsf{m}_{\mathsf{frank}}$'s moderator and platform time stamps are within a set expiry time from one another (line 1). Note that the winning condition of the game requires $\mathsf{id}_{\mathtt{src}} \notin \mathsf{corrupted}$. In other words the only possible way for token to have been created by a call $\mathsf{O}^{\mathsf{corrupt}}$ and $\mathsf{O}^{\mathsf{request}}$ must happen prior to the beginning of the recovery time, i.e. $\mathsf{token.t}_1 < \mathsf{recovery}_{\mathsf{start,t}}$. Recall that users can only be corrupt for one epoch at the current $\mathsf{global}_{\mathsf{t}}$ before being considered as honest and requiring the adversary to call $\mathsf{O}^{\mathsf{corrupt}}$ and $\mathsf{O}^{\mathsf{request}}$ if they wish to corrupt them again. This definition of corruption is enforced by $\mathsf{O}^{\mathsf{request}}$: if the id passed is not corrupted at the current $\mathsf{global}_{\mathsf{t}}$, the oracle returns $\perp$, otherwise it returns the appropriate token and increments $\mathsf{global}_{\mathsf{t}}$. Additionally, since the game master increments the global time by δ after $\mathsf{recovery}_{\mathsf{start,t}}$ on line 8 in $\mathsf{BAC}_{\delta}^{\mathcal{A}}$ then $\mathsf{envelope.t}_2 > \mathsf{recovery}_{\mathsf{start,t}} + \delta$ (where $\mathsf{envelope}$ is a shorthand for $\mathsf{m}_{\mathsf{frank}}.\mathsf{envelope}$). In other words, $\mathsf{token} \notin \mathsf{T}$ since otherwise $|\mathsf{envelope.t}_2 - \mathsf{t}_1| > \mathsf{expiry}$ and vfExp would return $\perp$.

Game_3: We replace $\mathsf{vfToken}$ with checking that $(\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, *) \overset{?}{\in} \mathsf{F}$, where $\mathsf{payload}$ refers to all elements of the franked message that are not on the envelope of the message. The adversary can distinguish this change with negligible probability equal to the advantage of the accountability game since they do not have access to the secret ephemeral keys of users for token constructed with $\mathsf{O}^{\mathsf{send}}$. We refer the reader to that proof for more details and not that both Games 1 and 3 are jointly upper bounded by the advantage of the accountability game.

Game_4: In vfCom, we replace line 1 with $b_1 := (\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, \mathsf{com}, *) \in \mathsf{F}$. We show that the adversary can distinguish between Games 1 and 2 with negligible advantage equal to the probability of breaking the binding property of the commitment scheme. Let's assume that there exists an adversary $\mathcal{A}$ that can distinguish both games, we construct an adversary $\mathcal{B}$ that can break the commitment game. Whenever $\mathcal{A}$ calls either $\mathsf{O}_{\mathsf{bs}}^{\mathsf{send}}$, $\mathsf{O}_{\mathsf{bs}}^{\mathsf{fwd}}$ or $\mathsf{O}_{\mathsf{bs}}^{\mathsf{deliver}}$, $\mathcal{B}$ constructs/modifies the franked message honestly on behalf of the moderator, the platform and the users and returns the result back to $\mathcal{A}$. Note that $\mathcal{B}$ stores every single such message. When $\mathcal{A}$ returns a challenge $\mathsf{m}_{\mathsf{frank}}$ back to $\mathcal{B}$ or successfully delivers an $\mathsf{m}_{\mathsf{frank}}$ to an honest user that was not previously logged in F using $\mathsf{O}_{\mathsf{bs}}^{\mathsf{deliver}}$, $\mathcal{B}$ strips the resulting $\mathsf{m}_{\mathsf{frank}}$ of everything except the commitment and its corresponding decommitment, and submits these values along with the original decommitment stored in F to the commitment game. If $\mathcal{A}$ succeeds then they will have necessarily submitted a decommitment that is different than the original one stored in F and $\mathcal{B}$ will win its corresponding game.

Game_5: In vfCom, we replace line 1 with $b_2 := (\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, \mathsf{com}, \mathsf{t}_2, \sigma_2) \in \mathsf{F}$, which effectively implies replacing vfCom with $(\mathsf{m}_{\mathsf{frank}}.\mathsf{payload}, \mathsf{m}_{\mathsf{frank}}.\mathsf{envelope}) \in \mathsf{F}$ (i.e. $\mathsf{m}_{\mathsf{frank}} \in \mathsf{F}$). We can show similarly to Game 1 in the accountability game that, without the platform's secret key, the adversary can distinguish between Games 2 and 3 with negligible advantage equal to the probability of breaking EU-CMA security property of the underlying signature scheme.

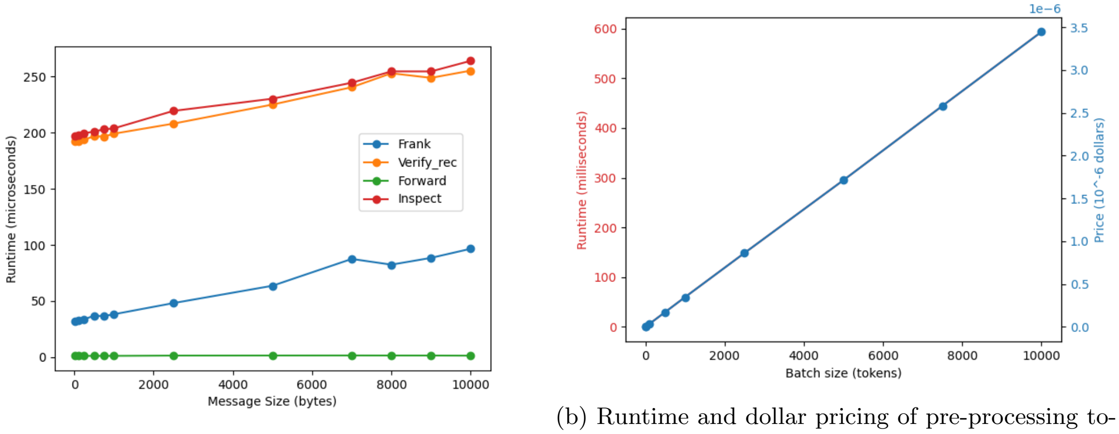
Game_6: We replace vfMsg with $\perp$ since we have shown that $\mathsf{m}_{\mathsf{frank}} \in \mathsf{F}$ and the game's winning condition requires otherwise.

We can therefore conclude that the adversary has negligible advantage in winning the $\mathsf{BAC}_{\delta}^{\mathcal{A}}$ game in Hecate equal to:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{Hecate}}^{\text{bac}}(\mathcal{A}) \leq\ & \mathsf{Adv}_{\mathsf{Hecate}}^{\text{accountability}}(\mathcal{A}) \\ & + \mathsf{Adv}^{\text{sig}_{\text{eu-cma}}}(\mathcal{A}) \\ & + \mathsf{Adv}^{\text{binding}_{\text{com}}}(\mathcal{A}).\end{aligned}$$ □

6 Implementation and Evaluation

We implemented Hecate as a Rust library that can be used as a back-end by other systems. Our implementation uses Signal's official platform agnostic API library libsignal-client [66] for our encryption, commitment and hashing building blocks. To that effect, we use libsignal-client's implementation of AES-256 GCM for symmetric encryption and HMAC with SHA-256 for commitments. We use the ed25519-dalek [26] crate for ed25519 signatures and their associated functions and SHA256 from the sha2 crate for our hash functions. Our implementation is open source and available at [40].



(a) Online runtime of Hecate's components as a function of message size in bytes

(b) Runtime and dollar pricing of pre-processing token generation (TGen) as function of the token batch size.

Figure 18: Runtime measurements.

In this section, we show experimental results when executing each component of Hecate in isolation. Then, we measure the overhead of Hecate when integrated into a Signal client.

6.1 Performance Cost and Comparison

In this section, we measure the runtimes and transmission sizes for each component within Hecate, using Criterion, a Rust benchmarking suite. We also evaluate prior open-source message franking systems on the same machine and compare them to Hecate.

Experimental setup. We ran all experiments on Amazon Web Services in the US East-Ohio Region, using a t3.small EC2 virtual machine running Ubuntu 20.04 LTS with 2GB of RAM on a 3.1 GHz Intel Xeon Platinum Processor. Each data point shown is the average of 300 trials, with outliers removed. We chose this machine and number of experiment trials to align with prior work [57, 72].

Hecate communication costs. We list the size of Hecate's franked ciphers in Table 3. The sizes in Table 3 stem from the fact that our libraries yield 32 byte commitments with 32 byte long commitment randomness, 32 byte ciphertexts, 64 byte signatures, 32 byte symmetric and public keys, 12 byte nonces for symmetric encryption, and 8 byte Unix timestamps. We remark that there exist more compact instantiations of these primitives; our choices were motivated by ease of implementation on top of libsignal-client.

Hecate's online runtime. Fig. 18a shows the performance of each component of Hecate for message sizes ranging from 10 bytes to 10 kB. Overall, the runtime costs remain low especially when compared to the cryptography already required within an end-to-end messaging system (cf. §6.2).

Most components require executing a SHA-256 hash function (to calculate x_2) whose runtime is linear in the message size, along with 0-3 digital signature operations whose cost is independent of message size (cf. Table 2). As a result, the signature(s) dominate the cost for small message sizes and the hash function dominates the cost for large messages. The two costs are balanced at a message size of 7.5 kB, where each of hashing and digital signing takes about $33\mu s$. Verify and Inspect are slower because they require about $192\mu s$ to verify 3 signatures. On the other hand, Forward, Stamp and TGen all have fast runtimes that are independent of message size. We remark that a forwarder is assumed already to have verified a message at reception time, so its only work during Forward is to move the envelope contents into the payload.

	Sent	Received	Report
Tyagi et al. [72]	489 B	489 B	489 B
Peale et al. [57]	256 B	320 B	160 B
Hecate	380 B	484 B	380 B

Table 4: Communication overhead of Hecate and other message franking schemes, in bytes.

	TGen	Frank	Verify	Inspect	Forward	Stamp
Tyagi et al. [72]	–	6339 μs	5461 μs	5939 μs	–	–
Peale et al. [57]	–	8.98 μs	69.56-138.19 μs	73.64 μs	8.46 μs	24.53 μs
Hecate	58.4 μs	38.24 μs	199.15 μs	203.87 μs	1.16 μs	29.17 μs

Table 5: Runtimes of message franking schemes, in microseconds, for a message size of 1 kB. The runtime of Verify within Peale et al. [57] differs based on whether the message is authored (left) or forwarded (right).

Hecate's preprocessing cost. We also measure TGen over various batch sizes from 1 to 10,000 tokens. Fig. 18b shows the runtime and computational cost based on a rate of 2.09¢ per hour for a t3.small AWS instance at the time of this writing. Our measurements show that the price of generating a batch of 10^4 tokens is 3.45×10^{-6} USD. Extrapolating to the scale of 10^{11} tokens (the approximate number of messages sent through WhatsApp daily [54]), we estimate the cost of token generation to be 35 cents per day. We also highlight that the moderator does not need to remember these ephemeral signing keys in between preprocessing and reporting; in fact the moderator doesn't require any storage cost at all.

Comparison with prior work. In this section, we compare our Rust implementation with the open-source software by TGLMR [72] and Peale et al. [57]. To ensure a level comparison: we re-ran the benchmarks from prior work [57, 72] on our t3.small AWS instance, we only considered the tree-linkable version of Peale et al., and we removed the double ratchet encryption within the benchmarks of Peale et al. in order to measure only the overhead of their message franking scheme.

We show a comparison of communication overhead in Table 4, and we compare computation overhead in Table 5 for a message size of 1 kB. The benchmarks of TGLMR [72] were orders of magnitude slower than the other works because their construction of designated verifier signature performs more group operations; their communication overhead was also the highest. The comparison between Hecate and Peale et al. [57] is more nuanced. We stress that Hecate achieves additional security properties like anonymity and backward security. As a consequence, senders perform more work in Hecate and transmit more data, whereas Peale et al. leverage a non-anonymous network so that the platform can tag the originator of a message. On the other hand, Peale et al. require a forwarder to generate a commitment, whereas Hecate only requires random generation of a 32 byte string (which could even be computed beforehand).

6.2 End-to-End Prototype Deployment

In this section, we integrate Hecate into an existing Signal client and show that Hecate adds minimal overhead.

Implementation. To test the end-to-end overhead of sending and receiving messages, we integrated our Rust Hecate library into the Java tools signal-cli [64] and libsignal-client [66]. The sender's Frank procedure adds the Hecate payload to a message before encrypting it using the EEMS, and then appends the Hecate envelope. The receiver decrypts the franked message and runs Verify. Our modified libraries are available as open source repositories [41, 42]. The sender gets tokens by running the Rust library prior to the start of the experiment.

We deployed the sender and receiver signal-cli instances on one machine with a 1.90GHz Intel i7-8650U CPU and 16GB of RAM running Ubuntu 20.04 LTS. They were connected over a wide-area network to an instance of signal-server [68].

	Hecate	No Hecate	Diff
Send	2.55 ms	2.38 ms	0.16 ms
Receive	3.19 ms	2.52 ms	0.67 ms
Total	5.74 ms	4.91 ms	0.83 ms
E2E Latency	37.28 ms	36.3 ms	0.98 ms

Table 6: Computation and communication costs for signal-cli with and without Hecate, for a message size of 1 kB. Send and Receive (10^4 trials) correspond to local computation prior to sending or after receiving the message over the network. E2E Latency (600 trials) starts at the beginning of Send and stops at the end of Receive, with network latency included.

Evaluation. We measured the client side overhead of running signal-cli with and without Hecate on messages of size 1 kB. In both cases, we measured the average of 10,000 trials of running local signal-cli operations and 600 trials of end-to-end (E2E) latency, with outliers removed. Our timer for the end-to-end latency test starts as soon as the source's signal-cli begins franking a message, and it ends when the receiver's signal-cli completes processing the franked ciphertext and outputs the message. These sample sizes are larger than in §6.1 to overcome the noise added by the network latency, the polling rate of the receiver, and the warm-up time of the jvm instance of signal-cli for each of the parties.

Our results are shown in Table 6. They showcase how the findings from Table 5 translate to imperceptible overheads in an actual deployment of Hecate on a Signal client. The inclusion of Hecate adds less than a millisecond of runtime locally and over the network, on average. Moreover, this difference is dwarfed by the sample variance of signal-cli due to the sources of measurement uncertainty.

7 Conclusion and Discussion

In this work, we constructed the first abuse reporting protocol that combines asymmetric message franking and source tracing. We integrated this construction into a Signal client and showed that its performance impact was imperceptible. Along the way, we generalized the AMF model to accommodate preprocessing, and we formalized security properties that hadn't previously been considered by message franking schemes.

In this final section, we discuss some extensions of Hecate, known limitations, and opportunities for future work.

7.1 Extensions

We extend Hecate's communication from the two-device setting to more realistic flows supported by Signal.

Group Messaging. Hecate's definitions and constructions can be ported in a straightforward manner to Signal's group messaging protocol, in which broadcasts to a group of size N are implemented via N individual point-to-point messages, after a server-assisted consensus protocol to determine the group [21]. We note that there exist recent works and an IETF standardization effort on sub-linear ends-to-ends encrypted group chats [5, 6, 12, 23, 63]; it remains an open problem to design abuse reporting mechanisms for these protocols. We do not yet know if there are any efficiency gains and added benefits to federating content moderation among different members of a group.

Multiple Devices. Hecate can easily support multiple devices for the same user (e.g., a phone and laptop) by giving each device its own independent set of tokens. The moderator can use the same $\mathsf{id}_{\mathtt{src}}$ for both sets of tokens, so that reports only name an identity rather than a device.

7.2 Limitations

We discuss a few limits of our approach.

Reporting Benign Messages. Our construction allows receivers to report messages that may later be deemed to be non-abusive. While it might be possible to require the receiver to prove to an honest moderator that the message they are reporting is actually abusive, this question is incredibly delicate and is therefore out of scope for this and all prior works on end-to-end abuse reporting. In the special case that the receiver is colluding with the moderator, we remark that (a) there is little that can be done to prevent false reports, and (b) the overall leakage is no worse than what EEMSs would already reveal to this colluding set.

Distinguishing Forwarded vs. Original Messages. In our construction, receivers can distinguish between sent and forwarded messages. While this may be a desirable feature in a messaging app, it is still a leakage in our system.

Forwarding Cycle Linkability. If the forwarding path of a message contains a cycle, i.e. a receiver receives the same forwarded message multiple times, then they can tell that these messages originate from the same source. This is an inherent weakness of Hecate as a result of forwarding the same tokens per message that we do not attempt to protect against. It may be possible to combine Hecate with Peale et al.'s techniques to remove this leakage [57].

Forwarding Tree Up-Rooting. Receiver's of a message may "up-root" its forwarding sub-tree by acting as the original senders of that message instead of forwarding it themselves. In general, we do not believe that this poses a concern as users do not have the incentive to incriminate themselves with bad messages. Moreover, prior work [73] suggests that this problem warrants an application side solution, if any, and not a cryptographic one that would restrict users from copying received messages and acting as their creators.

7.3 Future Work

Looking ahead, we believe there exist at least five possible avenues of future research in the space of privacy-respecting content moderation.

Content Censorship. Content moderation systems can be misused for censorship purposes. Questions surrounding what constitutes misinformation or a "bad" message fall outside the scope of this work and into the realm of policy making and social media regulation. We believe however that it may be interesting to federate the role of the moderator in: (1) defining bad messages, (2) verifying reports, (3) taking actions with respect to flagged contents and users.

Super Spreaders. A recent line of work [69, 78] on misinformation spread in social mediadistinguishes between honest users who forward misinformation and malicious actors that act as super spreaders of misinformation. Honest users can mistakenly forward or send misinformation content without ever realizing it. Super spreaders on the other are adversarially creating or spreading bad content. Future work could examine aggregate behavior in order to distinguish malicious vs. mistaken users.

Partial opening. Known AMF constructions like Hecate only allow the receiver to report all or none of a message. It should be possible to achieve partial opening to the moderator by extending the message franking techniques of Leontiadis and Vaudenay [52] to the asymmetric setting.

Stronger Notions of Backward Security. Backward security makes no guarantees with respect to anything created during the time of compromise. In the context of content moderation, this implies that the adversary can blame users for old compromised messages. We encourage future research into ways to limit the damage of adversarial moderation reports or to allow honest parties to correct the record post-recovery.

Ensuring System Security. We emphasize that our study of abuse reporting has been primarily through a cryptographic lens, and as a result does not capture all aspects of security. For example, many of our crypto definitions assume that clients already have sufficient preprocessing tokens in hand. When implementing Hecate, careful attention is required to ensure that adversaries cannot obtain a side channel by, for example, influencing when preprocessing is run. We encourage cryptographers, systems security researchers, usability experts, and domain specialists to investigate whether and how to integrate Hecate (or any abuse reporting mechanism) into an end-to-end encrypted messaging system in a matter that promotes online trust, safety, and security.

Acknowledgements

The authors are grateful to Kinan Dak Albab for his extensive feedback on this work. We also graciously thank Hoda Malecki and Sarah Scheffler for their valuable insights on an earlier version of the problem we tackle in this work.

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497. Springer, Heidelberg, February 2010.

[2] Surabhi Agarwal. India proposes alpha-numeric hash to track WhatsApp chat. *India Times*, 2021.

[3] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. MCMix: Anonymous messaging via secure multiparty computation. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 1217–1234. USENIX Association, August 2017.

[4] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2019.

[5] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 248–277. Springer, Heidelberg, August 2020.

[6] Joël Alwen, Sandro Coretti, Daniel Jost, and Marta Mularczyk. Continuous group key agreement with active security. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 261–290. Springer, Heidelberg, November 2020.

[7] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[8] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers' computation in private information retrieval: PIR with preprocessing. *Journal of Cryptology*, 17(2):125–151, March 2004.

[9] Mihir Bellare, Asha Camper Singh, Joseph Jaeger, Maya Nyayapati, and Igors Stepanovs. Ratcheted encryption and key exchange: The security of messaging. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 619–650. Springer, Heidelberg, August 2017.

[10] Luca Belli. Whatsapp skewed brazilian election, proving social media's danger to democracy. https://theconversation.com/whatsapp-skewed-brazilian-election-proving-social-medias-danger-to-democracy-106476, 2018.

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

[12] Alexander Bienstock, Yevgeniy Dodis, and Paul Rösler. On the price of concurrency in group ratcheting protocols. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 198–228. Springer, Heidelberg, November 2020.

[13] Alexander Bienstock, Jaiden Fairoze, Sanjam Garg, Pratyay Mukherjee, and Srinivasan Raghuraman. What is the exact security of the Signal protocol? https://cs.nyu.edu/~afb383/publication/uc_signal/uc_signal.pdf, 2021.

[14] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. https://toc.cryptobook.us/book.pdf, 2020.

[15] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, pages 77–84. ACM, 2004.

[16] Colin Boyd, Anish Mathuria, and Douglas Stebila. *Protocols for Authentication and Key Establishment, Second Edition.* Information Security and Cryptography. Springer, 2020.

[17] Brazilian fake news draft bill no. 2.630, of 2020. https://docs.google.com/document/d/1MHMDHsVJBi45PI1R5lAyoLmZvZk8eULHisYFqGy9X2s, 2020.

[18] Sébastien Campion, Julien Devigne, Céline Duguey, and Pierre-Alain Fouque. Multi-device for signal. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part II*, volume 12147 of *LNCS*, pages 167–187. Springer, Heidelberg, October 2020.

[19] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, Heidelberg, May 2001.

[20] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.

[21] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20*, pages 1445–1459. ACM Press, November 2020.

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

[23] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1802–1819. ACM Press, October 2018.

[24] Katriel Cohn-Gordon, Cas J. F. Cremers, and Luke Garratt. On post-compromise security. In *CSF*, pages 164–178. IEEE Computer Society, 2016.

[25] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE Computer Society Press, May 2015.

[26] dalek-cryptography. ed25519-dalek. https://github.com/dalek-cryptography/ed25519-dalek, 2020.

[27] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security 2004*, pages 303–320. USENIX Association, August 2004.

[28] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.

[29] Yevgeniy Dodis, Jonathan Katz, Adam Smith, and Shabsi Walfish. Composability and on-line deniability of authentication. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 146–162. Springer, Heidelberg, March 2009.

[30] Elizabeth Dwoskin and Annie Gowen. On WhatsApp, fake news is fast – and can be fatal. https://www.washingtonpost.com/business/economy/on-whatsapp-fake-news-is-fast--and-can-be-fatal/2018/07/23/a2dd7112-8ebf-11e8-bcd5-9d911c784c38_story.html, July 2018. Accessed: 09-28-2020.

[31] Facebook. Messenger secret conversations: Technical whitepaper (version 2.0). https://about.fb.com/wp-content/uploads/2016/07/messenger-secret-conversations-technical-whitepaper.pdf, 2017.

[32] Pooya Farshim, Beno$\hat{5}$t Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Robust encryption, revisited. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 352–368. Springer, Heidelberg, February / March 2013.

[33] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.

[34] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Heidelberg, August 2006.

[35] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 197–215. Springer, Heidelberg, May / June 2010.

[36] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 391–408. Springer, Heidelberg, August / September 2016.

[37] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253. Springer, Heidelberg, August 2015.

[38] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.

[39] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 29–37. Springer, Heidelberg, April 1990.

[40] Hecate Rust implementation. https://github.com/Ra1issa/hecate, 2022.

[41] Hecate's modified libsignal-client implementation. https://github.com/Ra1issa/libsignal-client, 2022.

[42] Hecate's modified signal-cli implementation. https://github.com/Ra1issa/signal-cli, 2022.

[43] Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[44] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154. Springer, Heidelberg, May 1996.

[45] Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019.

[46] Daniel Jost, Ueli Maurer, and Marta Mularczyk. A unified and composable take on ratcheting. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 180–210. Springer, Heidelberg, December 2019.

[47] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. Outside looking in: Approaches to content moderation in end-to-end encrypted systems. https://cdt.org/wp-content/uploads/2021/08/CDT-Outside-Looking-In-Approaches-to-Content-Moderation-in-End-to-End-Encrypted-Systems.pdf, 2021.

[48] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Round-optimal blind signatures in the plain model from classical and quantum standard assumptions. In *EUROCRYPT*, Lecture Notes in Computer Science, 2021.

[49] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography.* CRC press, 2020.

[50] Anunay Kulshrestha and Jonathan Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In *USENIX Security Symposium*. USENIX Association, 2021.

[51] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 1–16. Springer, Heidelberg, November 2007.

[52] Iraklis Leontiadis and Serge Vaudenay. Private message franking with after opening privacy. Cryptology ePrint Archive, Report 2018/938, 2018. https://eprint.iacr.org/2018/938.

[53] Linsheng Liu, Daniel S. Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). *CoRR*, abs/2109.04559, 2021.

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

[55] Ian Martiny, Gabriel Kaptchuk, Adam Aviv, Dan Roche, and Eric Wustrow. Improving Signal's sealed sender. In *NDSS*. The Internet Society, 2021.

[56] Oversight Board. Ensuring respect for free expression, through independent judgment. https://oversightboard.com/, 2021.

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

[58] Riana Pfefferkorn. Content-oblivious trust and safety techniques: Results from a survey of online service providers. https://ssrn.com/abstract=3920031, 2021.

[59] Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018.

[60] Newley Purnell and Jeff Horowitz. WhatsApp says it filed suit in India to prevent tracing of encrypted messages. https://www.wsj.com/articles/whatsapp-says-it-filed-suit-in-india-to-prevent-tracing-of-encrypted-messages-11622000307, 2021.

[61] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. In *EuroS&P*, pages 415–429. IEEE, 2018.

[62] Shahrokh Saeednia, Steve Kremer, and Olivier Markowitch. An efficient strong designated verifier signature scheme. In Jong In Lim and Dong Hoon Lee, editors, *ICISC 03*, volume 2971 of *LNCS*, pages 40–54. Springer, Heidelberg, November 2004.

[63] Michael Schliep and Nicholas Hopper. End-to-end secure mobile group messaging with conversation integrity and deniability. In *WPES@CCS*, pages 55–73. ACM, 2019.

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

[65] Signal. Technology preview: Sealed sender for signal. https://signal.org/blog/sealed-sender/, 2018.

[66] Signal. libsignal-client. https://github.com/signalapp/libsignal-client, 2020.

[67] Signal. Technical information. https://signal.org/docs/, 2021.

[68] Signal. Signal-server. https://github.com/signalapp/Signal-Server, 2022.

[69] Kate Starbird. Online rumors, misinformation and disinformation: The perfect storm of covid-19 and election2020. In *Enigma 2021*. USENIX Association, February 2021.

[70] Li Q. Tay, Mark J. Hurlstone, Tim Kurz, and Ullrich K. H. Ecker. A comparison of prebunking and debunking interventions for implied versus explicit misinformation. PsyArXiv, 2021.

[71] Kurt Thomas, Devdatta Akhawe, Michael Bailey, Dan Boneh, Elie Bursztein, Sunny Consolvo, Nicola Dell, Zakir Durumeric, Patrick Gage Kelley, Deepak Kumar, Damon McCoy, Sarah Meiklejohn, Thomas Ristenpart, and Gianluca Stringhini. Sok: Hate, harassment, and the changing landscape of online abuse. https://research.google/pubs/pub49786.pdf, 2021.

[72] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. Asymmetric message franking: Content moderation for metadata-private end-to-end encryption. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 222–250. Springer, Heidelberg, August 2019.

[73] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. Traceback for end-to-end encrypted messaging. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 413–430. ACM Press, November 2019.

[74] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. SoK: Secure messaging. In *2015 IEEE Symposium on Security and Privacy*, pages 232–249. IEEE Computer Society Press, May 2015.

[75] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *SOSP*, pages 137–152. ACM, 2015.

[76] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[77] WhatsApp. Two billion users – connecting the world privately. https://blog.whatsapp.com/two-billion-users-connecting-the-world-privately/, 2020.

[78] Liang Wu, Fred Morstatter, Kathleen M Carley, and Huan Liu. Misinformation in social media: definition, manipulation, and detection. *ACM SIGKDD Explorations Newsletter*, 21(2):80–90, 2019.