How to Trace Viral Content in End-to-End Encrypted Messaging

Pedro Branco[1], Matthew Green[2], Aditya Hegde[2], Abhishek Jain[2,3], and Gabriel Kaptchuk[4]

[1]Bocconi University
[2]Johns Hopkins University
[3]NTT Research
[4]University of Maryland, College Park

Abstract

We study the problem of combating *viral* misinformation campaigns in end-to-end encrypted (E2EE) messaging systems such as WhatsApp. We propose a new notion of Hop Tracking Signatures (HTS) that allows for tracing originators of messages that have been propagated on long forwarding paths (i.e., gone viral), while preserving anonymity of everyone else. We define security for HTS against *malicious* servers.

We present both negative and positive results for HTS: on the one hand, we show that HTS does not admit *succinct* constructions if tracing and anonymity thresholds differ by exactly one "hop". On the other hand, by allowing for a larger gap between tracing and anonymity thresholds, we can build succinct HTS schemes where the signature size does not grow with the forwarding path. Our positive result relies on streaming algorithms and strong cryptographic assumptions.

Prior works on tracing within E2EE messaging systems either do not achieve security against malicious servers or focus only on tracing originators of pre-defined banned content.

Contents

1 Introduction

Social networks empower individuals to broadcast their ideas to a mass audience. In addition to enabling community building, these communication platforms can also be weaponized by malicious agents to launch viral misinformation campaigns that cause significant harm. This concern is not hypothetical: such misinformation campaigns have been aimed at destabilizing democracies [Cho18], encouraging medically unsafe behavior [CC20], and some campaigns have resulted in murder and genocidal attacks [Cho19]. As instances of these viral misinformation campaigns grow in frequency and impact, social networks have begun to take an active role in moderating the content spreading on their platform.

While content moderation is difficult in the best case, a novel challenge in this area stems from the presence of viral content on *end-to-end encrypted* (E2EE) messaging platforms, such as Meta's WhatsApp [Wha23]. Ostensibly these networks were designed for person-to-person or small group communications. However, the networks exhibit remarkable *emergent* behaviors through which "broadcast" functionality can be realized via the via *message forwarding* chains, in which a single piece of content is widely distributed to many users, who each forward it to additional users in their contacts. Moderating content in these systems can be technically complex, given that the platform (by design) cannot view the content of messages being exchanged between users [SM23].

The challenges of combating viral misinformation campaigns on E2EE messaging platforms has led to several governments urging (or mandating, in the case of the Indian government [oET21]) that platforms provide capabilities that allow law enforcement to identify the originator of viral misinformation—a capability discussed in the academic literature as *source traceback*. Practically, this is commonly understood as a requirement to provide law enforcement with limited access to plaintext, reducing the privacy guarantees provided for both "viral" and non-viral content. Providers and human rights advocates fear that such "backdoors" may be used to fundamentally undermine the confidentiality provided by end-to-end encryption—a concern that is supported by instances of alleged spying directed against opposition parties and activists [U.S22].

Source Tracing in End-to-End Encrypted Messaging Systems. A proposed alternative to combating viral misinformation campaigns within end-to-end encrypted messaging systems is to *restrict* the messages for which the platform can perform source traceback. This approach was first investigated by Tyagi et al. [TMR19], and relies on *abuse reporting* by clients; only once a message is (voluntarily) reported as misinformation by a user, the platform can use information embedded in the message to recursively walk back through a message's forwarding path until the message's creator is found. Since Tyagi et al.'s initial work, source tracing has emerged as a technical problem of interest [PEB21, IAV22, LRTY22, BGJP23] and several governments have even called for source tracing to be integrated into live deployments of end-to-end encrypted message platforms [RS20, Ban18]. However, providers have been reluctant to adopt these solutions because they do not provide sufficiently strong security properties [New21].

Most existing source tracing proposals suffer from a serious shortcoming: they assume that the platform does not collude with users. The only thing stopping the platform from executing source tracing in these proposals is access to plaintext; if the platform colludes with a user, they have the technical capability to identify the origin of arbitrary messages received by the colluding user. Thus, the platform's ability to trace the source is independent of the nature of the message itself and it is assumed that the platform will act honestly and only use its tracing power on messages with content that is identified as misinformation. Unfortunately, this honesty assumption is impractical, as demonstrated by previous instances of foreign governments colluding with platform employees [U.S22] and the use of E2EE platforms in conflict zones [Bla25].

The recent source tracing scheme of Bartusek et al. [BGJP23] attempts to mitigate the threat of collusion between clients and server. Namely, it proposes a system that provides a source tracing power to the platform that is limited—or "constrained," in the language proposed by Ananth et al. [AJJM22]—to content that falls within a very particular *predicate*: matching a pre-selected and hard-coded list of banned content. This means that the platform cannot carry out source traceback for any message that does not satisfy the predicate—even if the platform colludes with users—and similarly, licit content that wrongly satisfies the predicate (e.g., due to collisions in the perceptual hash function used [PFG+23, SHNK22]) becomes immediately traceable even if it has not gone "viral" at all. Bartusek et al. consider this problem in the context of known instances of illegal content, such as child sexual abuse material, as it is possible to predefine a predicate that captures a subset of non-compliant messages before they are transmitted. In the viral misinformation setting, however, creating a predicate for banned messages is a difficult problem—even with access to plaintext— as cultural contexts differ, intent can be hard to assign, and language shifts over time. Moreover, a message may only be considered problematic *after* it becomes *widespread*—long after it is initially sent.

Limiting Source Tracing to Viral Messages. In this work, we take a step back and re-examine this problem space from a theoretical—although practically motivated—perspective. Namely, we ask if it possible to create end-to-end encrypted messaging systems that provide a *constrained* source tracing power to malicious providers such that it can only be used on *viral* misinformation campaigns. We observe that in order for misinformation to be viral—and thus a reasonable target for source tracing—it must be widely disseminated among the platform's user base; a message containing misinformation that is passed among ten users cannot reasonably be construed as viral, even if half of those users report it as misinformation. This requires defining a predicate not over the message's content, but instead over *the manner in which a message has circulated on the platform* to determine whether it could be viral. This *virality predicate* can then be baked into the tracing system to cryptographically constrain the platform, making it impossible for even a malicious operator to collude with malicious users in order to trace non-viral messages—or even to weaken the security of users who inadvertently forward the content[1]. Most critically, achieving constrained source tracing in the presence of collusions rules out any solution that employs online trusted servers, or includes new and vulnerable master tracing keys.

Our approach can be seen as lifting recent work on *preconstrained cryptography* [GKL21, ABD+21, AJJM22, BGJP23, KLN23] into the distributed setting. In this paradigm, master keys of a cryptographic scheme are generated with limits on their power; by carefully choosing these limits, preconstrained cryptographic primitives can be used to construct cryptosystems that meet desirable policy goals. In the traceback setting, we limit the platform's tracing power such that it can only be used on messages that have traversed specific paths through the platform's userbase. This is more technically challenging than preconstrained encryption, since the predicate must reason over the (private) joint state of a *distributed system*, in which there are many agents, each holding relevant cryptographic key material.

[1]Importantly, our system prevents source tracing for non-viral messages, but does not provide any technical means to distinguish between misinformation and non-misinformation. As such, a malicious operator might be able to execute tracing on a viral message that is not misinformation, for an arbitrary notion of misinformation. Finding technical ways to address this gap is important before deployment can be considered.

With this new conceptual approach to source tracing in hand, we study the feasibility of realizing it for a natural notion of virality: *long forwarding paths*, i.e., tracing can only be executed on messages for which the path between final receiver and creator contains a sufficiently large number of users. While not the only notion of virality that might be desirable, this predicate provides an important starting point for the study of preconstraining viral content tracing and is a meaningful notion of virality in practice, as we discuss in Section 4.1. Moreover, in Section 7, we show that the techniques developed in realizing source tracing for this predicate can be extended to realize source traceback for other virality predicates.

1.1 Our Contributions

In this work, we study source tracing for end-to-end encrypted messaging systems in which tracing can only be executed on *viral* messages. We find that establishing the feasibility of constructing asymptotically efficient instantiations of this paradigm—a first step toward concrete efficiency—is highly non-trivial, requiring careful attention to definitions and new techniques. More specifically, our contributions are as follows:

Hop Tracking Signatures. We propose the notion of Hop Tracking Signatures (HTSs), a new cryptographic primitive that allows source tracing for messages with *long forwarding paths*. Intuitively, the sender of a message should stay anonymous unless a malicious server gets access to a copy of that message that has been forwarded by t users (some of whom may be controlled by the adversary). We carefully define achievable notions of security for this primitive and investigate their limitations and applicability to real-world systems. The syntax and security of HTS is strongly motivated by ease of composition with E2EE messaging platforms. Specifically, HTSs are simply appended to the message being sent and do not require modifications to the messaging system.

Impossibility of Succinct, Non-Gapped Hop Tracking Signatures. We prove that it is impossible to construct Hop Tracking Signatures that are both *succinct* in the number of times the message has been forwarded and uses a single traceability threshold t, before which the source is completely anonymous but after which the source is always traceable. We prove this using lower bounds in communication complexity.

Succinct Hop Tracking Signatures with a Gap. We show that it is possible to construct succinct Hop Tracking Signatures when one admits a *gap* between the *anonymity threshold* t_{anon} (at which point anonymity ceases to hold for the source) and *traceability threshold* t_{trace} (at which point the source can be traced). Specifically, we present a construction of succinct Hop Tracking Signatures such that $t_{\mathsf{anon}} = (1-\epsilon)t$ and $t_{\mathsf{trace}} = (1+\epsilon)t$ for any $\epsilon < 1/2$ and positive integer t, such that $1/\epsilon$ and t are polynomial in the security parameter.

Our positive result requires some notion of *identity* for users, which we achieve using a registration model (it is easy to see that without any notion of user identity, the entire notion of virality becomes vacuous). We achieve this in our construction via a registration server that issues credentials to new users. This server is not involved in any further operation of the system. The security of our first construction holds against adversaries that can adaptively register new corrupted users and have access to the key material of either the main E2EE server or the registration server, but not both. As we discuss in Section 2.1, this corruption model is *necessary* for our result. Indeed, a similar security model is considered in [BGJP23], where the adversary can corrupt either the main server or the input-authentication server but not both.

In Section 7, we show that this registration server can be replaced with a non-secret functionality such as a bulletin board or a *key transparency* server [LL23]: in this model, registration of new users does not involve a secret key that the adversary could compromise in order to trace users. This approach places some limits on the ideal functionality we can achieve, however: since the only honest party in the system may be the message originator, the originator itself must commit to its view of the set of registered users in the system at the time it sends its message. Thus only those users who registered prior to the message being sent will count as valid forwarders, and users who register afterwards will not be able to increase the forwarding count. This provides a type of *forward secrecy* against adversaries that corrupt *both* the E2EE server and the registration server. Specifically, an adversary can compromise source anonymity only for messages sent after all servers are corrupted but cannot deanonymize messages sent before the time of compromise. In contrast to prior work [TMR19, PEB21, IAV22, LRTY22]—where a server that colludes with users can deanonymize the source of all messages, both before and after compromise—our approach limits such deanonymization to messages sent after the system is fully compromised.

Techniques and Assumptions. Our positive result crucially relies on streaming algorithms for counting distinct elements. A key contribution of our work is to use such algorithms in an *adversarial* model where the adversary can adaptively choose inputs and view the internal state and random tape of the streaming algorithm. Although prior work has studied streaming algorithms in adversarial models, known positive results require assumptions on the adversary that are not true in our setting. To overcome these challenges, we identify robustness properties of some streaming algorithms in the literature [BYJK+02] that allows us to carefully compose them with cryptographic tools to achieve security against such adversaries in our setting. See Section 2.6 for a discussion.

We also rely on standard cryptographic assumptions as well as two strong cryptographic primitives: extractable witness encryption (eWE) [GKP+13, GGHW14] and incrementally verifiable computation (IVC) for NP with knowledge soundness [Val08]. The latter of these primitives is known either based on heuristics [Val08] or extractability assumptions [BCCT13].[2] This is a topic of extensive research, with several practical constructions known and used in the blockchain ecosystem. eWE is currently not known from standard assumptions; moreover, prior work has shown barriers to constructing eWE from standard assumptions for specific auxiliary input distributions [GGHW14]. We note, however, that one could use existing candidates for "plain" witness encryption (which is known from well-studied assumptions) [JLS21, JLS22] as heuristic candidates for eWE.

Obtaining our results from weaker assumptions or moving towards concrete efficiency are both interesting directions for future research.

1.2 Related Work

We briefly summarize relevant related work below.

Pre-constrained Cryptography. A recent line of work has investigated the design of cryptographic primitives in which the master keys are constrained from the onset in their capabilities [GKL21, ABD+21, AJJM22, BGJP23, KLN23]. In the context of end-to-end encrypted messaging, the work of [BGJP23] explores the problem of tracing the source of pre-defined illegal content. However,

[2] IVC for NP without knowledge soundness was recently constructed based on standard assumptions [DJJ+25].

these solutions do not extend to our setting, where the tracing predicate depends on the private state of a distributed system, rather than the state of a single party.

Non-Interactive Secure Multiparty Computation over Graphs. A related line of work has investigated the design of secure multiparty computation (MPC) protocols in which the interaction patterns among parties are defined via arbitrary graphs [HIJ+16, BJPY18]. Our setting is somewhat similar to these works, in the sense that the computation of the virality predicate can be viewed as a form of secure computation over the forwarding graph. However, a key distinction is that our setting additionally requires preserving the anonymity of participants—a property not addressed in these works.

Tracing misinformation. The problem of tracing originators of misinformation in E2EE messaging systems was first studied by Tyagi et al. [TMR19]. They propose a system for *path traceback* that allows the E2EE server to trace the source and *all intermediate forwarders* of a message upon receiving a report by any user. Informally, the system maintains an *encrypted linked list* where each node in the list is a ciphertext held by the E2EE server while pointers to previous nodes are encryption keys held by users. To report a message, the user sends the decryption key for the last node in the linked list to the server, which allows it to decrypt ciphertexts in reverse order and thus trace all users in the forwarding path. The authors also propose a system for *tree traceback* which allows tracing all recipients of the message, including those on forwarding paths different from that of the reporting user. In addition to the server's storage requirements growing linearly with the number of forwards, a primary drawback of the proposed system is that privacy is only guaranteed when the E2EE server does not collude with any user. A single report on a message suffices to completely deanonymize all users in the forwarding path.

The follow-up work of Peale et al. [PEB21] allows tracing only the message source and avoids deanonymizing all users on the forwarding path. At a high level, their approach requires users sending a new message to interactively compute a publicly verifiable and non-malleable encryption of the source's identity under the server's secret key. This ciphertext is forwarded alongside the message. A recipient can report the message by sending the ciphertext to the server, which decrypts it to reveal the source. They also extend their solution to achieve a stronger unlinkability property, ensuring that users who receive the same message cannot determine whether it originated from the same source or from different sources. Subsequent work called Hecate [IAV22] extends the work of Peale et al. [PEB21] to also ensure forward and backward secrecy. While these works remove the need for large server storage, they do not provide any privacy guarantees when the E2EE server colludes with even a single user.

Finally, Liu et al. [LRTY22] build on the solution of Peale et al. [PEB21] by introducing an oblivious Bloom filter, which allows the server to efficiently and obliviously maintain a count of the number of reports for every message. This count can be queried by any client that receives the message. If the count exceeds a threshold t, the reporting user also submits the ciphertext—which, as in [PEB21], encrypts the source identity—to the server, which can then decrypt it to trace the source. However, as before, a server that colludes with even a single client can deanonymize the source since it learns the ciphertext; the threshold is enforced only in the non-collusion setting. Moreover, the proposed system counts the total number of reports, rather than the number of *distinct* users reporting a message. To account for this, the system requires enforcing a per-user limit on the number of reports that can be submitted.

In summary, while prior works on source tracing are concretely efficient, they provide a weaker notion of security where the anonymity of the source is not guaranteed if the E2EE server colludes with even a *single* client.

2 Technical Overview

We now give an overview of the key concepts in this work. We begin by giving an overview of the modeling choices that we make, discuss how the notion of a viral message can be captured formally and propose a simple but practically meaningful definition of virality. We then explore how to instantiate a system with the desired properties.

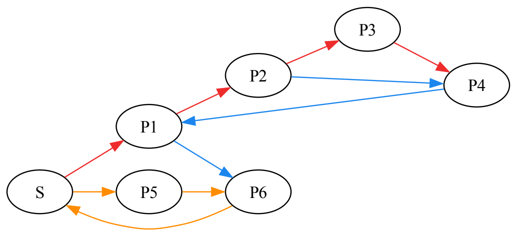
2.1 Modeling E2EE Messaging Setting and Virality

In this work, we make black-box use of an underlying end-to-end encrypted messaging platform, consisting of a set of *users* and a *server*. The platform realizes a very simple secure communication functionality: registered users can send messages to one another such that the server can only control if and when that message is delivered and can not read the contents of the messages. When a user wants to *forward* a received message to a new recipient, they simply add an annotation indicating that the message is being forwarded. This approach means that all forwarding logic can be contained within the message payload being transmitted, and thus the cryptographic protocol implementing end-to-end encryption remains unchanged.

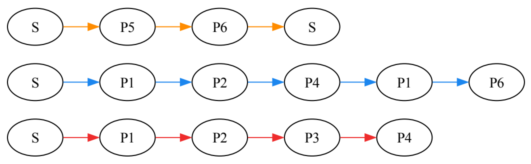
When a user sends a new (i.e., not forwarded) message, we call that user the message's *source*. Adding traceability to an end-to-end encrypted messaging platform allows the server to identify the source of a message upon receiving a report from a recipient, containing the message plaintext. Tracing should be possible even when the server obtains a report from a user that received a forwarded message—at which point it is not inherent that the reporting user knows the source's identity. Note that on properly end-to-end encrypted messaging platforms, the server cannot trace the origin of a message even given access to ciphertexts and user communication patterns because the server can not distinguish between encrypted messages containing forwarded content and one with fresh content.

Tracing viral messages to their source. When prior work [PEB21, IAV22, LRTY22] has considered the problem of source tracing within end-to-end encrypted messaging systems, these works provide the server with the power to trace the origin of *any* message reported by a user. This means that the server can trace the source of any message of its choice, as long as it colludes with a recipient. Moreover, previous works sometimes reveal the identity of all *intermediate* users who forwarded the message—an approach that can de-anonymize innocent users who were unaware that a message contained disinformation [TMR19]. The only exception is [BGJP23], where the server can carry out source traceback for a reported message only if the message content lies within a predefined set of banned messages. Within this work, we are concerned with exact origin tracing in which the server's capability to trace messages is strictly limited to messages that have gone "viral."

In order to formalize the notion of virality, we shift the focus of the protocol from the content of the message to the private metadata about the message: how that message has spread throughout the network. Specifically, the flow of the message through the platform via forwards, implicitly defines a *forwarding graph*: a directed multi-graph in which nodes represent users and edges represent each forwarding event, as shown in Figure 1a. Note that users can appear multiple times across different paths and even within the same path, since a user may forward the message to multiple recipients, receive a forward of the same message from different users as well as forward



(a) The forwarding graph is a directed multi-graph.



(b) Re-interpreting the forwarding graph as a set of forwarding paths.

Figure 1: The flow of a message via forwards in an E2EE messaging platform. Nodes represents users, edges represent forwarding events and S denotes the source user that created the message.

a message that it had previously forwarded. We observe that a viral message can be formally specified by defining a *virality predicate* over the message's forwarding graph. This virality predicate can then be embedded within the system to constrain the server's ability to trace the origin of messages.

While constraining the server's ability for source tracing using a virality predicate over the message's forwarding graph provides a useful framework, the practical utility of the system strongly depends on the choice of predicate. For example, a virality predicate that just tests for the number of times a message has been forwarded (number of edges in the forwarding graph) doesn't offer any meaningful security in the real-world—two users can repeatedly forward the message between themselves to render the predicate true, but such a message can hardly be considered viral. In this work, we will mainly focus on the *unique-forwarders-on-a-path* predicate. Specifically, a reported message is considered viral under this predicate only if the forwarding path—the sequence of users that forwarded the message, from the source to the reporting user, as shown in Figure 1b—consists of at least t unique users. Note that under this predicate a viral message has been received by at least t distinct users and thus our notion of virality is robust to the repeated forwarding attack discussed previously. While we limit our discussion in this section to the unique-forwarders-on-a-path predicate, we consider a more general predicate that tests for number of unique users across multiple forwarding paths in Section 7.

Equipped with a framework to formally define what makes a message viral, we now proceed to discuss the security properties that are required of a system that supports source traceback of only viral messages. Going forward, we use 'viral messages' to mean messages that satisfy the unique-forwarders-on-a-path predicate, unless stated otherwise.

Security Goals. Our aim is to design a system with the following intuitive security properties.

- *Source anonymity:* When the source of a message is honest, then it remains anonymous as long as the message is not viral. The malicious parties may choose to manipulate or forward the message in an attempt to identify the source. That is, the adversary could try to attack the cryptographic construction in order to execute the tracing functionality before the virality predicate is satisfied.[3]
- *Forwarder anonymity:* An honest forwarder—any intermediate user in the forwarding path—will always remain anonymous, even if the source of the message is eventually traced.

[3]As discussed above, the adversary is also allowed to interact with the system in an attempt to "artificially" make the message viral.

– *Traceability:* We must ensure that a malicious source cannot originate their message in a way that prevents them from being identified later or changes the output of the tracing functionality to be an innocent, honest user as the source.

We wish to achieve these goals within a model that reasonably approximates the powers of a strong, real-world adversary. Specifically, we are concerned about an end-to-end encrypted messaging system with a malicious server that may control a potentially large number of *malicious users*. Security against such an adversary prevents malicious insiders within the platform from performing source tracing inappropriately, even when they might control some number of accounts (either for personal use or for testing purposes).

While it may be tempting to study this problem in the *static* setting (i.e., the set of users and the subset of those users that are corrupt is determined before messages begin circulation) this does not match the power of real-world adversaries. As such, we let the adversary *adaptively* register new users in the system throughout the experiment, possibly sampling new users' cryptographic key material based on the messages it has already observed.

2.1.1 Boundaries on Achievable Security

There are, however, two inherent boundaries on the extent to which we can achieve these security goals within the envisioned system; one which affects the anonymity guarantee and the other which affects traceability.

Sybil Attacks. It is easy to see that allowing the adversary to create an unbounded number of users in the system—essentially performing a Sybil attack—quickly makes virality a vacuous concept. Specifically, such an adversary can always ensure that the forwarding path of any message contains sufficient number of distinct users, thus rendering the message viral. It is thus impossible to ensure anonymity of an honest source against such an adversary.

In order to ensure a meaningful notion of virality and guarantee anonymity of sources in the face of these Sybil attacks, we consider a registration-based model where a *registration server* is responsible for curating which users are part of the system. Specifically, the registration server legitimizes each new user's key material, enabling them to send and forward messages on the platform, similar to the role of existing Key Transparency [MBB+15, LL23] or Certificate Transparency [Cer, LLK13] systems. We emphasize that this is the only interaction that the registration server has with users and the registration server is completely uninvolved with the execution of source tracing.

We assume that the registration-server does not collude with the main (E2EE messaging) server. That is, our security goals should hold as long as the adversary does not learn the key material of both servers.[4] This is necessary, since otherwise, the adversary could run the previously described Sybil attack by using the registration server's private key to register key materials for an unbounded number of users.

Refresh Attacks. The very nature of E2EE messaging makes the tracing system susceptible to a source "refresh" attack where a corrupt user can always choose to copy the *content* of a received message into a new message—effectively forwarding the message while making themselves the source. This could reset any progress made towards satisfying the virality predicate. Such an

[4]We note that the source tracing proposal of [BGJP23], which is the only other work that constrains the server's source tracing ability based on a predicate, also requires a non-collusion assumption to ensure that the predicate is meaningful.

attack on the traceability guarantee of the system is somewhat inherent to the E2EE messaging functionality, since all users should be able to send new messages as well as view the content of received messages.

Due to these unavoidable issues, we propose our security definitions to guarantee traceability of the source when the message has been consecutively forwarded by sufficient number of unique *honest* users. Looking ahead to Section 4.1, we verify this intuition by modeling the overall forwarding graph of a message as a random graph where corrupt users always perform refresh attacks and terminate forwarding paths. We find that in a network of 2 billion users (e.g., WhatsApp), even extraordinarily powerful adversaries corrupting hundreds of millions of users cannot prevent long forwarding chains (e.g., of a million users) from being formed. This indicates that ensuring traceability when sufficient honest users consecutively forward the message can provide meaningful guarantees in practice.

In Section 2.7, we discuss the real-world implications of requiring a registration server as well as the consequence of refresh attacks.

2.2 Hop Tracking Signatures

Our goal is to allow source traceback of only viral messages in E2EE messaging systems, where a message is considered viral (for the purpose of our discussion) if its forwarding graph satisfies the unique-forwarders-on-a-path predicate discussed previously. However, any meaningful solution to supporting source traceback must require only minimal changes to existing E2EE messaging systems—it should be functionally equivalent to an E2EE messaging platform, except that it additionally allows users to forward and report messages as well as enables the server to trace the origin of viral messages. We adopt a similar approach as prior work to ensure this compatibility and focus our efforts on designing a cryptographic primitive that can be composed in a black-box manner with E2EE messaging.

A natural starting point for a cryptographic primitive that meets our needs is group signatures. Group signatures are an anonymous signature with the additional property that a master secret holder can deanonymize the sender. The problem with group signatures is that they provide no protections against a malicious secret key holder—once they have access to the group signature, they can always trace the source. Recognizing this problem, Bartusek et al. [BGJP23] initiated the study of pre-constraining group signatures, such that tracing powers are a function of the message content. In this work, we define *Hop Tracking Signatures* (HTS), an approach to pre-constrained group signatures that facilitates source tracing for only viral messages. Unlike group signatures, HTSs will have an explicit $\mathsf{Forward}$ algorithm that is run by each forwarder. Recall that we want this primitive to ensure (1) *source anonymity* when the length of the forwarding path is less than t, (2) there should always be *forwarder anonymity*, no matter the forwarding path length, and (3) *traceability* when the length of the forwarding path exceeds t. Composing such a primitive with E2EE messaging systems is then straightforward: the source computes and appends an HTS on the message when sending it to a recipient, each user runs the $\mathsf{Forward}$ algorithm on the HTS received from their sender before forwarding it, and a user can send the message and the appended HTS to the server to report a message. However, the difficulty in constructing HTS is that the source cannot perform the necessary cryptographic operations alone, as it does not know if the message will go viral. Instead, we need to perform a distributed computation (as the message is forwarded through the network) that checks if the predicate has been satisfied.

2.3 Warmup: A Non-Succinct Construction

We begin by developing a template for constructing HTSs that provides the desired security and correctness properties, at the expense of being communication inefficient. Our first task is to design a distributed computation that *counts* the number of unique users along the forwarding path. Importantly, each of these users must be an authorized part of the system—and must demonstrate their authorized status in order to participate in the distributed computation. This leads us naturally to having each forwarder sign the message as it is forwarded, appending the signature to the message. Determining the current forwarding path length is as simple as counting the number of attached signatures. The upside of this approach is that it is *unforgeable*, in that an adversary controlling some ℓ users in the system cannot make it look as though $\ell + 1$ users have forwarded the message. The downside of this approach, of course, is that it provides no *privacy* for any of the forwarders—a verifiable proof that they forwarded the message is visible to anyone who sees a plaintext.

In order to provide privacy, we can hope to replace the signatures with some form of anonymous credential, like a ring signature [RST01]. Because any given ring signature could feasibly have been produced by any user in the system (assuming the ring is the set of all users in the system), the server or a corrupt user cannot use them to implicate any particular user as having been part of the forwarding path. Such an approach seems to solve our privacy problem, but actually compromises the integrity of the count. Imagine a malicious user that signs a particular message an arbitrarily large number of times. By the anonymity property of the ring signature scheme, this result will be indistinguishable from a set of signatures produced by a large number of honest users, and thus seem to satisfy the predicate.

In order to make this template workable, we need a method to *de-duplicate* the signatures such that each user can only contribute a single one. One straight-forward approach to this is to limit the users' power to produce multiple signatures without detection. To provide this property, we can replace these ring signatures with *unique* ring signatures [FZ13], a derandomized version of ring signatures that ensures that re-signing a message will always result in the same signature. Using this tool, it will be trivial to observe if a malicious user has signed the message multiple times: simply look to make sure that all of the signatures are distinct.

All that remains is identifying a way to use this stream of unique ring signatures to conditionally release the identity of the source. The natural tool to do this generically is witness encryption [GGSW13]. The source can simply encrypt their identity with witness encryption such that the witness is a set of t distinct unique ring signatures on the chosen message. Once a path has gotten long enough—and the message has accumulated enough unique ring signatures—the witness encryption can be decrypted. One subtlety is that we must ensure that a malicious source cannot frame an honest user by putting the wrong identity into the witness encryption. This can be accomplished using a non-interactive zero-knowledge proof (of knowledge) that the sender knows the secret key of the identity being put into the witness encryption.

In summary, we have the source encrypt their identity within witness encryption (and prove correctness of this encryption) and have each forwarder add a unique ring signature to the message, t of which will serve as the key to the witness encryption. Anonymity of the source follows directly from the use of the witness encryption scheme and signature unforgeability—if the adversary can produce a valid witness when less than t users have signed the message, then it can forge unique ring signatures for honest parties. Privacy of the intermediary forwarders follows from the anonymity property of the signature scheme. Finally, a malicious source cannot frame an honest user when tracing is executed due to the zero-knowledge proof of knowledge.

2.4 Barriers To Succinctness

While the baseline HTS construction discussed in the previous subsection works, it is not succinct—the size of the signature grows linearly with the length of the forwarding path. With viral messages in particular, the forwarding path can be quite long. As such, it is desirable to find a HTS construction with the same security and correctness properties but where the signature size grows sublinearly in the forwarding path length.

While such a HTS would be desirable, we find that they cannot be constructed without relaxing our requirements in some way. To prove that such a succinct construction cannot exist, we reduce to a well known lower bound in communication complexity. Namely, we show in Section 5that if there exists a succinct HTS scheme then it could be used to construct a two party set disjunction protocol with sublinear communication complexity in the size of the sets. However, Kalyanasundaram and Schnitger [KS92] showed an information theoretic lower bound that the communication complexity of the disjunction functionality is $\Omega(n)$, where n is the size of the sets. Interestingly, the impossibility result only relies on the correctness property of the HTS scheme and not its source anonymity property i.e., the impossibility does not rely on security against a malicious server. Intuitively, this suggests a barrier to succinctly counting number of unique forwarders, even when all users behave honestly.

2.5 Gapped Hop Tracking Signature Schemes

Having established that the construction of succinct HTS schemes is impossible without relaxing our requirements, we turn our attention to finding an appropriate relaxation that permits a succinct construction. We begin by asking if there is a way to realize the template described in Section 2.3 without appending a unique ring signatures for each forward—which was the primary cause of non-succinctness. Observe that the unique ring signatures are playing multiple roles simultaneously: (1) *counting* the number of forwarders in the path, (2) *de-duplicating* any repeated users in the forwarding path, (3) providing *unforgeability* such that the adversary can not simply generate a witness to decrypt the witness encryption independently, and (4) providing *privacy* for each forwarder. When moving to the succinct regime, we must find a way to replicate all of these properties—all without allowing the state to grow. Moreover, these requirements seem to have a lot of internal tension e.g., the need for anonymity makes creating any kind of record difficult, while de-duplication seems to demand some kind of record.

We take as our starting point the need to do *counting* in a succinct way. We observe that an HTS scheme counts the number of unique forwarders in a *streaming* fashion. Thus, we need a streaming algorithm for counting unique values—a well studied problem [FM85, BYJK$^+$02, FEFGM07, KNW10, CVM22]. Streaming algorithms, by definition, use memory that grows at most sublinearly in the length of the stream processed. However, they suffer from two drawbacks: (1) these algorithms only approximate the number of unique elements, and some error in their count is inherent; and (2) these algorithms are rarely used in cryptographic settings, and ensuring robustness in our highly adversarial setting will be non-trivial. We first discuss how we address the first drawback and then continue to discuss the use of streaming algorithms in an adversarial setting.

Gapped Thresholds. Recall that the security requirements outlined in Section 2.1 required the source to be traceable when the forwarding path consisted of more than t unique forwarders and required the source to remain anonymous otherwise i.e., traceability and source anonymity were defined using a single threshold t. Informed by the hope to use a streaming algorithm in order to realize a succinct HTS scheme, we consider a natural relaxation of the source anonymity and traceability requirements that allow circumventing the impossibility result. The core idea is to introduce a gap in the source anonymity and traceability thresholds such that the HTS scheme does not need to go from anonymous (i.e. hiding the source) to tracing abruptly exactly when the t^{th} user forwards the message. In more detail, we separate the threshold t into a pair of thresholds that will parameterize an HTS scheme, an *anonymity threshold* t_{anon} and a *traceability threshold* t_{trace}, subject to the constraint that $t_{\mathsf{anon}} = (1-\epsilon)t$ and $t_{\mathsf{trace}} = (1+\epsilon)t$ where $0 < \epsilon < 1/2$ is a fixed constant and t is fixed polynomial in the security parameter. In order to reduce notational complexity, we refer to the primitive that realizes the relaxed security requirements as Hop Tracking Signatures (as the non-relaxed version is impossible to construct). A $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-secure HTS scheme should satisfy the following two (informal) properties:

(1) t_{anon} *Source Anonymity:* When the number of forwarders of a message is less than t_{anon}, a malicious server with access to the message plaintext should be able to learn nothing about the message's source.

(2) t_{trace} *Traceability:* When the number of forwarders of a message is more than t_{trace}, a server with access to the message plaintext should be able to learn the message's source.

The forwarder anonymity requirement remains unchanged. Importantly, the relaxed requirements do not guarantee source anonymity or traceability when the number of forwarders is greater than t_{anon} but less than t_{trace}. Nevertheless, we will always require that an adversary cannot frame an honest user regardless of the number of forwards.

Observe that an HTS scheme that has a gap in the source anonymity and traceability thresholds need not compute the exact number of unique forwarders, which in turn implies that the impossibility result discussed previously no longer applies to it. This motivates our approach of realizing succinct HTS using streaming algorithms as discussed previously, where the inherent error in the output of streaming algorithm manifests as the gap in the HTS scheme's thresholds.

2.6 Succinct HTS Using Streaming Algorithms

At a high level, we want to use the same basic strategy as in the baseline, non-succinct construction, but rather than appending a unique ring signatures each time the message is forwarded, the forwarder instead pushes its unique ring signature into the streaming algorithm[5]. The witness encryption statement will change to reason over the state of the streaming algorithm. While this template is intuitive, there remain a number of non-trivial problems we must address in order to make it work.

Background: Streaming Algorithms. Streaming algorithms facilitate computing functions over a large sequence of data with only limited memory (i.e., much less memory than the length of the sequence). Because the algorithm could not hold the full sequence at once, the data is provided to the algorithm as a *stream*, in that the algorithm only gets access to the data one element at a time and must discard each element before proceeding to the next. In general the algorithm will

[5]Looking ahead we will replace the unique ring signature altogether, but this intuition remains informative.

also only be able to see each element of the stream once. This structure is amenable to use in our setting: we execute the streaming algorithm in a distributed manner by having each forwarder use its input to update the streaming algorithm's state and additionally send the updated state to the recipient.

However, employing this strategy to design succinct HTS schemes requires using the streaming algorithm in an adversarial setting, where the adversary learns the intermediate state of the streaming algorithm and influences the input steam by controlling the forwarding path.

Streaming Algorithms in Adversarial Settings. The standard model for analyzing streaming algorithms assumes that the input stream is *independent* of the algorithm's internal state—an assumption that clearly does not hold in our setting. Other stronger models have been studied, but appear to also fall short of meeting our needs. The first of these models, *adversarially robust streaming algorithms* [BEY20, BHM+21, ABED+0, BEJWY22] provides robustness against an adaptive adversary with oracle access to the algorithm's current estimate. Importantly, however, the adversary is not given access to the algorithm's internal state. In our setting, corrupt forwarders learn *both* the intermediate state of the algorithm and can adaptively choose the message's forwarding path to control the input stream, making them more powerful than the modeled adversary. A second challenge stems from the fact that streaming algorithms rely on randomness, either during initialization or while processing inputs. In our setting, this randomness must be supplied by potentially corrupt parties—including the source and intermediate forwarders—which makes it unclear how to ensure that the randomness is truly uniform. The limitations of these algorithms in our setting are underscored by prior work that show that the estimates provided by streaming algorithms can be *arbitrarily inaccurate* in the presence of adaptively chosen inputs [HW13, PR21, MFS23], i.e., biasing even a polynomial number of inputs can cause the estimation of the count to be exponentially biased. Another, stronger model considers *white-box adversarial streaming algorithms* [ABJ+22, FJW24], which provides robustness against adversaries that can choose inputs and read the algorithm's internal state. Constructions in this model leverage fresh randomness that is sampled each time an input is processed; an important limitation on this adversary is that it *cannot* control or predict this randomness. Unfortunately, it is unclear how to ensure that corrupt forwarders use good randomness for updating the streaming algorithm's state in our setting[6].

Robustly Counting Unique Forwarders. In order to overcome this problem, we carefully compose cryptographic tools with streaming algorithms by adopting the following approach.

(1) We use a deterministic streaming algorithm, that does not use randomness to initialize its state or process inputs, but provides accurate estimates only for specific distributions of the input[7]—the stream must consist of repetitions and reorderings of uniformly random values. The advantage of using a deterministic streaming algorithm is that it is secure against an adversary that adaptively chooses the input stream based on the algorithm's internal state. In particular, an adversary that reorders or repeats uniformly random values gains no advantage from inspecting the state, as it can compute the same state on its own. Moreover, determinism

[6]One natural approach to adapting white-box adversarial streaming algorithms in our settings would be to leverage random oracles. Unfortunately, a random oracle does not seem to help since the adversary can query the oracle before choosing the forwarding path—the input stream is then chosen adaptively based on the random coins that *will be used* by the streaming algorithm. Moreover, it is unclear how to ensure that corrupt forwarders indeed use the random oracle's output for updating the streaming algorithm's state.

[7]A deterministic streaming algorithm cannot provide good estimates of the number of distinct elements for arbitrary streams [CK16].

sidesteps the need to verify that parties use uniform randomness when initializing and updating the streaming state, thereby addressing the second challenge in applying streaming algorithms to our setting.

(2) We use cryptographic techniques to ensure that the forwarders' inputs to the streaming algorithm follow the distribution required for obtaining accurate estimates of the number of unique forwarders. Specifically, since the deterministic streaming algorithm expects the input stream to consist of repetitions and reorderings of uniformly random values, we define a pseudorandom mapping from forwarder identities to *anonymized pseudonyms*, which are then input to the streaming algorithm. Key to ensuring pseudorandomness of this mapping is the fact that our use of streaming algorithms is in a *distributed setting*, where inputs to the streaming algorithm are provided by several parties, a subset of which are corrupt. The randomness provided by honest parties is key to ensuring that the inputs to the streaming algorithm are indeed (pseudo)random.

(3) An adversary that receives a signature can attempt to bias the input to the streaming algorithm by selectively forwarding through a subset of corrupt users or by registering new parties. That is, while the deterministic streaming algorithm is secure against adversaries that adaptively reorder and repeat uniformly random values, adversaries in our setting can also subsample or insert new inputs into the stream. To address this, we identify new robustness properties of the streaming algorithm—insertion robustness and deletion robustness—and show that these properties suffice to ensure source anonymity and traceability even under such adversarial behavior.

(4) Finally, we ensure that all users follow the prescribed protocol using zero-knowledge proofs.

In more detail, our starting point is the streaming algorithm for counting distinct elements proposed by Bar-Yossef et al. [BYJK+02]. The algorithm is initialized by sampling a pair-wise independent hash function h. Each input x_i is hashed and the algorithm maintains a running list of the γ smallest values of $h(x_i)$, where γ is a tunable parameter. The number of distinct elements hashed can be estimated by looking at the largest value on the list; as the number of distinct elements processed increases, the largest value on the list will, in expectation, decrease because h maps the x_i's to random (pairwise independent) values. The result is an estimate—for which we can get good, formal bounds—on the number of unique elements in the stream[8].

Observe that the only randomness used by the streaming algorithm is for sampling a pairwise independent hash function i.e., the randomness is used to map the input stream into a stream of random pairwise independent values. Given this stream of random values, the streaming algorithm is *deterministic* since it only needs to update the list of minimum hash values. Thus, we view the streaming algorithm of [BYJK+02] as a deterministic streaming algorithm for an input stream consisting of repetitions and reorderings of random values. We are left to ensure that the forwarders' inputs to the streaming algorithm have the required distribution to ensure accurate estimates of the number of unique forwarders.

Recall that in the template discussed previously, each forwarder used a unique ring signature as input to the streaming algorithm. Informally, unique ring signatures served as a pseudorandom

[8]In order to ensure that this estimate will be within a certain range with negligible error, multiple copies of this algorithm will been run in parallel.

injective map for each message, from the forwarder's identity to a publicly verifiable signature, where the fact that this mapping was chosen pseudorandomly ensured the forwarder's anonymity.

Now, to ensure that the input stream has the distribution required by the streaming algorithm, we use PRFs to replace unique ring signatures with a pseudorandom injective mapping from forwarder identities to anonymized pseudonyms. The primary challenge is ensuring that these pseudonyms are indeed pseudorandom in the presence of an adversary that can corrupt users and manipulate forwarding paths. We first consider the case of achieving source anonymity and then consider achieving traceability.

- **Source Anonymity:** To ensure source anonymity, we rely on the randomness sampled by the source to define the mapping from forwarder identity to pseudonyms. In more detail, the source samples a per-message key k_{msg} which is carried with the message, and each forwarder computes $x = \mathsf{PRF}(k_{\text{msg}}, \mathsf{ID}\|m)$ and uses x as input to the streaming algorithm, where ID is some per-user unique identifier. Observe that the choice of k_{msg} completely determines the contribution $x_i = \mathsf{PRF}(k_{\text{msg}}, \mathsf{ID}_i\|m)$ of a forwarder with identifier ID_i. In case of an honest source, k_{msg} is uniformly random which in turn implies that each x_i is uniformly random[9] — the adversary has no influence in the choice of a corrupt forwarder's pseudonym. Nevertheless, an adversary that obtains the signature learns k_{msg} and hence the pseudonyms of the t_{anon} corrupt users. Could it then forward the signature using a subset of corrupt users to bias the input stream? We show that the streaming algorithm is robust to such "deletions". Informally, this is because the streaming algorithm's output is inversely proportional to the γ-th minimum value in the input stream, and sub-sampling pseudonyms of corrupt forwarders can only increase the minimum value input to the streaming algorithm.
- **Traceability:** Traceability requires that the streaming algorithm provide good estimates when the source is corrupt. Our previous mapping from forwarder identity to pseudonyms is no longer random in case of a corrupt source since k_{msg} is chosen by the adversary. To circumvent this issue, each forwarder computes $x = \mathsf{PRP}(k_{\text{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$, where k_{msg} is the per-message key described above, fk is a user-specific key sampled during user registration and PRP is a pseudorandom permutation. Observe that in case of an honest source, the map continues to be pseudorandom because k_{msg} is uniformly random. In case of a corrupt source, the pseudonym of honest forwarders is pseudorandom since the user-specific key fk of honest users is unknown to the adversary and PRP is a bijection and hence does not affect the distribution of honest users' pseudonyms. This ensures that the streaming algorithm provides accurate estimates when the signature is forwarded by honest users. However, can the adversary prevent traceability by additionally forwarding the signature through corrupt users with non-pseudorandom pseudonyms? We show that the streaming algorithm is robust to such "insertion" attacks. This is because its output is inversely proportional to the γ-th minimum value in the input stream and forwarding through corrupt users can only decrease the minimum value input by honest forwarders.

To summarize, we exploit the randomness of an honest source for source anonymity and of honest forwarders for traceability. Furthermore, we exploit an inherent asymmetry to ensure robustness of the streaming algorithm to adversarially chosen forwarding paths: an adversary corrupting at most t_{anon} users cannot inflate the count, and it cannot deflate the count of a signature that has

[9]The pseudonyms are in fact pseudorandom, but nevertheless indistinguishable from uniformly random values. This suffices for our application.

been forwarded by at least t_{trace} honest forwarders. Forwarder anonymity is guaranteed because each user's pseudonyms are anonymized by the user-specific PRF key fk.

We ensure that each user follows the prescribed algorithm using zero-knowledge proofs. Specifically, we use Incrementally Verifiable Computation (IVC) [Val08], which allows for incrementally computing a proof for a computation, enabling the verification of any intermediate state and the efficient updating of this proof to prove the correctness of subsequent steps. Importantly, the size of an IVC is independent of the number of steps in the computation which in turn ensures that the HTS is succinct. Zero-knowledge of the IVC ensures forwarder anonymity since the witness for computing each incremental proof contains the forwarder's key fk.

Summary of the Construction. Equipped with a streaming algorithm to succinctly and robustly count the number of unique forwarders, IVC to ensure honest behavior and witness encryption to constrain the server, our HTS scheme works as follows.

(1) The source $\mathsf{U}_{\mathrm{src}}$, begins by computing a group signature σ_{gs} on the message using its signing key. It then encrypts the group signature using a witness encryption scheme[10] (ct) and proves the correctness of this encryption (π).

(2) The source initializes the streaming algorithm state $\mathsf{st}_{\mathsf{init}}$ and samples a key k_{msg}. Finally, $\mathsf{U}_{\mathrm{src}}$ initializes the IVC as π_0^{IVC}.

(3) The source then sends the message $(m, \mathsf{ct}, \pi, \mathsf{st}_{\mathsf{init}}, k_{\mathrm{msg}}, \pi_0^{\mathrm{IVC}})$ via the end-to-end encrypted messaging platform.

(4) When a user wants to forward the message $(m, \mathsf{ct}, \pi, \mathsf{st}_i, k_{\mathrm{msg}}, \pi_i^{\mathrm{IVC}})$, it does the following:

   – Compute streaming algorithm input as $x = \mathsf{PRP}(k_{\mathrm{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$,
   – Update the streaming state to be st_i by processing the input x,
   – Update the IVC to π_{i+1}^{IVC} by proving that the streaming algorithm's state was correctly updated.

(5) When the server gets access to a plaintext message, it can attempt to decrypt the witness encryption using the streaming algorithm state and the IVC as a witness. If the count is above t_{trace}, then the decryption will succeed and it will obtain the group signature σ_{gs}. It can then decrypt σ_{gs} using the opening key to learn the identity of the source.

Need for Extractability and Puncturability. In our scheme, the IVC proof serves as a witness for decrypting the witness encryption. However, since an IVC proof only offers computational soundness, we need *extractable* witness encryption[11] to ensure source anonymity, i.e., security requires extracting an IVC proof from an adversary that can successfully win the source anonymity experiment.

Additionally, to prove anonymity, we must extract a witness from the IVC proof.[12] Thus, we rely on the knowledge soundness of the IVC. However, source anonymity must also rely on the

[10] We leave the statement and language for the witness encryption unspecified in this overview, as formally describing it distracts from the overall intuition we wish to cultivate.

[11] This means that valid IVC proofs exist for false statements, they are simply infeasible to find. Hence, the witness encryption statement might not be true, and we are unable to rely on the security of standard witness encryption.

[12] A witness for the IVC will be a signature scheme, and thus we will be able to reduce the hardness of breaking anonymity to the unforgeability of the underlying signature

privacy of the honest user's secret keys which in turn requires simulating the IVC. This raises the following conundrum: how can we simulate IVC proofs while also using the IVC extractor? In particular, simulation-extractable IVC are not known because IVC inherently requires non-black-box extraction.

To address this issue, we follow the approach of prior works [BJPY18, AWZ23] and define a primitive called *Puncturable NIZKs*. Here, a trapdoor can be punctured on a predicate to simulate proofs for statements where the predicate is true, while proofs for statements where the predicate is false are guaranteed to be sound. Each step of the IVC then verifies a Puncturable NIZK proof. In the security proof, we can puncture the target statement. This allows the IVC extractor to simulate all proofs except the target one and, thus, the reduction will be able to extract a valid witness.

2.7 Real World Implications

Finally, we conclude the overview with a discussion on real world implications of our modeling choices discussed in Section 2.1. The primary focus of this work is to explore the feasibility of using cryptography to combat misinformation while preserving the security of end-to-end encryption. Thus, we limit our discussion to how our security model and definitions translate to real-world scenarios. However, further research is crucial, especially on the social implications of deploying tracing systems for viral messages, before such systems are deployed in practice.

Requiring a Registration Server. Recall that ensuring a meaningful notion of virality and preserving anonymity of honest sources requires preventing an adversary from running Sybill attacks. We thus assume a registration-server that does not collude with the main (E2EE messaging) server i.e., an adversary does not learn the key material of both servers. Given this requirement, we design a system that places minimal requirements on the registration authority. Ideally, the system should only require a "universal" registration functionality that simply issues certificates to legitimate users and is not tied to the E2EE messaging system itself. Our construction is motivated by this goal: the registration-server is only used during user registration and is not required for the regulation operations of the E2EE system. In Section 7, we discuss approaches to limit the consequences of a compromised registration server.

In general, the problem of identifying "bots" and preventing Sybill attacks is of broad interest due to its relevance to social networks and other systems. Finding good practical solutions is of significance beyond our work. Indeed, centralized solutions (e.g., ID.me and other national digital identity providers) are starting to be deployed, and research to move these credential systems out of government control and add privacy is already underway.

Refresh Attacks. A consequence of the E2EE messaging functionality allowing all users to send new messages is that a corrupt user can reset the source of a forwarded message by sending an identical message anew. While our analysis in Section 4.1 indicates that the resulting traceability guarantee is meaningful in practice, it nevertheless serves only as an approximation of how messages are forwarded in the real world. The following discussion in [TMR19] however provides stronger evidence as to advantages of such tracing guarantees in the real world: "WhatsApp ran an experiment on the effectiveness of limiting the number of forwards a user could make of a given message. They found that this was effective despite the fact that the user could circumvent the limit after they reached it by copying and pasting the message. As a result, this forward limitation is now deployed globally."

3 Preliminaries

Basic Notation. We use $[x, y]$ to denote the set $\{x, x+1, \ldots, y\}$ where $x, y \in \mathbb{Z}$ and $x \leq y$. We use $[x]$ as a shorthand to denote $[1, x]$. We use $\mathbf{x} \in \mathcal{X}^\ell$ to denote a vector of ℓ elements in $\mathcal{X}$. For a vector $\mathbf{x} = (x_1, \ldots, x_\ell)$, we use $\mathbf{x}_i = x_i$ to denote the i-th element in $\mathbf{x}$ where $i \in [\ell]$ and $|x| = \ell$ to denote its length. For any algorithm $\mathcal{A}$, we use $\mathcal{A}^{\mathsf{F}}(x; r)$ to denote that the algorithm runs on input x, with oracle access to $\mathsf{F}(\cdot)$ and random tape r. In most cases, the random tape and oracles are omitted if it is clear from context or not required. We use λ as the computational security parameter, $\mathsf{negl}(\lambda)$ to denote negligible functions, and $\mathsf{poly}(\lambda)$ to denote the set of polynomials in λ. We will slightly abuse notation and often use p to denote both the polynomial as well as its evaluation $p(\lambda)$ at λ. For two probability ensembles $\{A_i\}_i$ and $\{B_i\}_i$, we use $\{A_i\}_i \overset{\mathsf{p}}{\equiv} \{B_i\}_i$ to denote that the ensembles are identical, $\{A_i\}_i \overset{\mathsf{s}}{\approx} \{B_i\}_i$ to denote that the ensembles are statistically close and $\{A_i\}_i \overset{\mathsf{c}}{\approx} \{B_i\}_i$ to denote that the ensembles are computationally indistinguishable. We use $\Pr[E : A]$ to denote the probability of an event E in an experiment defined by executing A.

3.1 Basic Cryptographic Primitives

We will use pseudorandom functions (PRFs), pseudorandom permutations (PRPs) and digital signature schemes that are existentially unforgeable under chosen message attack; all of which are implied by one-way functions [LR89, HILL99, Rom90].

Definition 1 (Pseudorandom Function)**.** Let $\ell \in \mathsf{poly}(\lambda)$ and $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ be an efficiently computable keyed function. PRF is a pseudorandom function if for all non-uniform polynomial time distinguishers D there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$\left|\Pr_{k \leftarrow \{0,1\}^\lambda}\left[\mathsf{D}^{\mathsf{PRF}(k,\cdot)}(1^\lambda) = 1\right] - \Pr_{f \leftarrow \mathsf{Func}_\ell}\left[\mathsf{D}^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where Func_ℓ is the set of all functions from $\ell(\lambda)$ length bitstrings to $\ell(\lambda)$ length bitstrings.

Definition 2 (Pseudorandom Permutation)**.** Let $\ell \in \mathsf{poly}(\lambda)$ and $\mathsf{PRP} : \{0,1\}^\lambda \times \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ be an efficiently computable keyed permutation. PRP is a pseudorandom permutation if for all non-uniform polynomial time distinguishers D there exists negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$\left|\Pr_{k \leftarrow \{0,1\}^\lambda}\left[\mathsf{D}^{\mathsf{PRP}(k,\cdot)}(1^\lambda) = 1\right] - \Pr_{f \leftarrow \mathsf{Perm}_\ell}\left[\mathsf{D}^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

where Perm_ℓ is the set of all permutations over $\ell(\lambda)$ length bitstrings.

Definition 3 (Digital Signature)**.** Let $\ell \in \mathsf{poly}(\lambda)$. A digital signature scheme $\mathsf{Sig} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a tuple of algorithms with the following syntax.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ is a PPT algorithm that outputs a public key pk and a signing key sk.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$ is a PPT algorithm that takes the signing key sk and a message $m \in \{0,1\}^{\ell(\lambda)}$ as input and outputs a signature σ on m.
- $\mathsf{Verify}(\mathsf{pk}, m, \sigma) =: b$ is a polynomial time algorithm that takes the public key pk, a message $m \in \{0,1\}^{\ell(\lambda)}$ and a signature σ as input and outputs a bit $b \in \{0,1\}$ denoting if σ is valid.

We require that the signature scheme satisfies the following properties.

- **Correctness:** For every $\lambda \in \mathbb{N}$ and every message $m \in \{0,1\}^{\ell(\lambda)}$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1 : \begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m) \end{array}\right] = 1.$$

- **Unforgeability:** For all $\lambda \in \mathbb{N}$ and all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\Pr\left[\begin{array}{c} m \notin \Sigma \quad \wedge \\ \mathsf{Verify}(\mathsf{pk}, m, \sigma) = 1 \end{array} : \begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(1^\lambda, \mathsf{pk}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where Σ is the set of messages $\mathcal{A}$ queries to its signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$.

3.2 Non-Interactive Zero Knowledge

A non-interactive zero-knowledge (NIZK) argument system enables a prover to compute a proof that convinces a verifier about the validity of a given statement without revealing any additional information about the witness. An argument of knowledge provides a stronger guarantee that the prover indeed possesses a valid witness, as opposed to ensuring just the validity of the statement. Currently, we know how to build NIZKs in the common reference string model from a wide variety of assumptions such as the subgroup decision assumption over pairings [GOS06], Learning With Errors (LWE) [CCH+19, PS19], a combination of Decisional Diffie-Hellman (DDH) and Learning Parity with Noise (LPN) [BKM20, BCD+25], or sub-exponential DDH [JJ21]. NIZK arguments of knowledge can be built generically from NIZKs and public-key encryption schemes with perfect correctness [DP92].

Definition 4 (Non-interactive Zero-knowledge Argument Of Knowledge)**.** Let $\mathcal{L}$ be an NP language with relation $\mathcal{R}$. A Non-Interactive Zero Knowledge (NIZK) argument of knowledge for $\mathcal{L}$ is a tuple of algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following semantics.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$ is a PPT setup algorithm that outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$ is a PPT algorithm that takes as input a common reference string crs, a statement x in $\mathcal{L}$ and a witness w and outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) =: b$ is a polynomial time algorithm that takes as input a common reference string crs, a statement x and a proof π and outputs a bit $b \in \{0,1\}$ indicating if the proof is accepted or rejected.

We require a NIZK argument of knowledge to satisfy the following properties.

- **Completeness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) \neq 1 : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Knowledge Soundness:** There exists an extractor $\mathcal{E} = (\mathsf{Setup}, \mathsf{Extract})$ which is a pair of PPT algorithms with the following properties. Firstly,

$$\left\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right\}_\lambda \overset{c}{\approx} \left\{\mathsf{crs} : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\lambda)\right\}_\lambda .$$

Moreover, for all non-uniform polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge \\ (x, w) \notin \mathcal{R} \end{array} : \begin{array}{r} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}.\mathsf{Setup}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathcal{E}.\mathsf{Extract}(\mathsf{td}, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Zero-Knowledge:** There exists a simulator $\mathcal{S} = (\mathsf{Setup}, \mathsf{Prove})$ which is a pair of PPT algorithms such that for all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\left|\begin{array}{r} \Pr\left[\mathcal{A}^{\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)\right] \\ - \Pr\left[\mathcal{A}^{\mathsf{SimProve}(\cdot,\cdot)}(\mathsf{crs}) = 1 : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)\right] \end{array}\right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{SimProve}(\cdot, \cdot)$ is an oracle that when called on input (x, w) returns $\perp$ if $(x, w) \notin \mathcal{R}$ and returns $\mathcal{S}.\mathsf{Prove}(\mathsf{crs}, \mathsf{td}, x)$ otherwise.

3.3 Group Signatures

Group signature schemes, first introduced by Chaum and van Heyst [Cv91], allow a group of users, each having its own private signing key, to compute signatures on behalf of the group such that the signatures can be verified under a common public verification key. The scheme guarantees that a signature does not provide any information about the identity of the user that signed the message except to an authority that posses an opening key. Our definition is adapted from [BSZ05], which defines group signatures with a registration server that issues private signing keys to new group members, and an opener that can trace the source of a signature. Group signatures can be constructed generically from perfectly correct public-key encryption schemes secure against chosen-ciphertext attacks, digital signature schemes that are existentially unforgeable under chosen message attacks, and simulation-sound adaptive NIZK proofs for NP [BSZ05].

Definition 5 (Group Signatures)**.** A group signature scheme $\mathsf{GSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Register}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ has the following syntax.

- $\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk})$ is a PPT setup algorithm that outputs the group public key gpk, the registration key rsk, and the opening key osk.

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$ is a PPT key generation algorithm that allows a user that wishes to join the group to compute its public key pk and private key sk.

- $\mathsf{Register}\langle \mathcal{S}_{\mathsf{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{pk}, \mathsf{sk})\rangle \rightarrow \langle \mathsf{pk}, \mathsf{usk}\rangle$ is an interactive protocol between the registration server $\mathcal{S}_{\mathsf{reg}}$ and a user U where the registration server has the registration key rsk as input and the user has its locally computed public and private key $(\mathsf{pk}, \mathsf{sk})$ as input. At the end of the protocol, the registration server obtains the user's public key pk as output and the user obtains a private signing key usk as output.

- $\mathsf{Sign}(\mathsf{usk}, m) \to \sigma$ is a PPT signing algorithm that allows a user to compute a signature σ on a message $m \in \{0,1\}^*$ using its private signing key usk.

- $\mathsf{Verify}(\mathsf{gpk}, m, \sigma) =: b$ is a polynomial time algorithm that takes as input the group public key gpk, a message $m \in \{0,1\}^*$ and a signature σ and outputs a bit $b \in \{0,1\}$ indicating if σ is valid.

- $\mathsf{Open}(\mathsf{osk}, m, \sigma) =: \mathsf{pk}$ is a polynomial time opening algorithm that takes as input the opening key osk, a message m and a signature σ on m and either outputs $\perp$ or the identity pk of the user that computed σ.

We model the adversary's capabilities in security definitions using the following oracles. The oracles are described using the keys $(\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk})$ generated by the experiment in the setup phase, and a set of corrupt users $\mathcal{I}$ and honest users $\mathcal{H}$ initialized by the experiment.

- $\mathsf{RegH}(\,)$: Allows the adversary to register honest users. When called, it runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\langle \mathsf{pk}, \mathsf{usk} \rangle \leftarrow \mathsf{Register}\langle S_{\mathsf{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{pk}, \mathsf{sk}) \rangle$, and returns pk to $\mathcal{A}$. It adds $(\mathsf{pk}, \mathsf{sk}, \mathsf{usk})$ to $\mathcal{H}$.

- $\mathsf{Reg}(\,)$: Allows the adversary to register corrupt users. When called, it plays the role of the registration server $S_{\mathsf{reg}}(\mathsf{rsk})$ in an execution of $\mathsf{Register}$ and adds the public key pk of the registered corrupt user to $\mathcal{I}$.

- $\mathsf{Cor}(\mathsf{pk})$: Allows the adversary to corrupt an honest user. When called with a user's public key pk, it returns $(\mathsf{sk}, \mathsf{usk})$ to the adversary and adds pk to $\mathcal{I}$, if $(\mathsf{pk}, \mathsf{sk}, \mathsf{usk}) \in \mathcal{H}$.

- $\mathsf{HSign}(\mathsf{pk}, m)$: Allows the adversary to obtain signatures from honest users. When called with a user's public key pk and a message m, it computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}, m)$ and returns σ to the adversary, if $(\mathsf{pk}, \mathsf{sk}, \mathsf{usk}) \in \mathcal{H}$.

- $\mathsf{Trace}(m, \sigma)$: Allows the adversary to open signatures. When called with a message m and signature σ, it computes $\mathsf{pk} := \mathsf{Open}(\mathsf{osk}, m, \sigma)$ and sends pk to the adversary.

We require that a group signature scheme satisfy the following properties.

- **Correctness:** For every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{gpk}, m, \sigma) = 1 \\ \wedge \\ \mathsf{Open}(\mathsf{osk}, m, \sigma) = \mathsf{pk} \end{array} : \begin{array}{r} (\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}, m) \leftarrow \mathcal{A}^{\mathsf{RegH}}(\mathsf{gpk}) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{usk}, m) \end{array}\right] = 1$$

where pk output by $\mathcal{A}$ is such that $(\mathsf{pk}, \mathsf{sk}, \mathsf{usk}) \in \mathcal{H}$.

- **Anonymity:** For every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\sigma \notin \Sigma \wedge b = b' \ : \begin{array}{r} (\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}_0, \mathsf{pk}_1, m, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{RegH}, \mathsf{Reg}, \mathsf{Cor}, \mathsf{HSign}, \mathsf{Trace}}(\mathsf{gpk}, \mathsf{rsk}) \\ b \leftarrow \{0,1\} \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{usk}_b, m) \\ b' \leftarrow \mathcal{A}^{\mathsf{RegH}, \mathsf{Reg}, \mathsf{Cor}, \mathsf{HSign}, \mathsf{Trace}}(\mathsf{st}, \sigma) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where Σ is the set of signatures queried to Trace, and $(\mathsf{pk}_0, \mathsf{pk}_1)$ output by $\mathcal{A}$ are such that both $(\mathsf{pk}_0, \mathsf{sk}_0, \mathsf{usk}_0)$ and $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{usk}_1)$ are in $\mathcal{H}$.

- **Traceability:** For every non-uniform polynomial time adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{gpk}, m, \sigma) = 1 \wedge \\ (\mathsf{pk}, m) \notin \mathcal{J} \ \wedge \ \mathsf{pk} \notin \mathcal{I} \end{array} : \begin{array}{r} (\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{RegH}, \mathsf{Reg}, \mathsf{Cor}, \mathsf{HSign}}(\mathsf{gpk}, \mathsf{osk}) \\ \mathsf{pk} \leftarrow \mathsf{Open}(\mathsf{osk}, m, \sigma) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{J}$ is the set of inputs queried to HSign.

- **Unframeability:** For every non-uniform polynomial time adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{gpk}, m, \sigma) = 1 \wedge \\ (\mathsf{pk}, m) \notin \mathcal{J} \ \wedge \ \mathsf{pk} \in \mathcal{H} \end{array} : \begin{array}{r} (\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m, \sigma) \leftarrow \mathcal{A}^{\mathsf{RegH}, \mathsf{HSign}}(\mathsf{gpk}, \mathsf{rsk}, \mathsf{osk}) \\ \mathsf{pk} \leftarrow \mathsf{Open}(\mathsf{osk}, m, \sigma) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathcal{J}$ is the set of inputs queried to HSign.

3.4 Policy-Based Signatures

Policy-based signature (PBS) schemes [BF14] allow computing signatures only on those messages that conform to an authority specified policy. Informally, PBS schemes guarantee that signatures are *unforgeable* and preserve the *privacy* of the policy. However, we will only require that signatures are unforgeable. In fact, compared to the definition of [BF14], we require weaker properties from PBS schemes where the policy language is always in P and unforgeability is only required against an adversary that obtains a single signing key. PBS schemes can be built generically from public-key encryption schemes secure against chosen plaintext attacks, digital signatures and NIZKs.

Definition 6 (Policy-Based Signatures)**.** Let $\ell \in \mathsf{poly}(\lambda)$. A policy-based signature scheme $\mathsf{PBS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is a tuple of algorithms with the following syntax.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$ is a PPT algorithm that outputs a master public key mpk and master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \nu) \to \mathsf{sk}_\nu$ is a PPT algorithm that takes as input the master secret key msk and an efficiently computable predicate ν, and outputs a signing key sk_ν.
- $\mathsf{Sign}(\mathsf{sk}_\nu, m) \to \sigma$ is a PPT algorithm that takes as input a signing key sk_ν and a message $m \in \{0,1\}^{\ell(\lambda)}$ and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{mpk}, m, \sigma) =: b$ is a polynomial time algorithm that takes as input the master public mpk, a message $m \in \{0,1\}^{\ell(\lambda)}$ and a signature σ and outputs a bit $b \in \{0,1\}$ denoting if σ is valid.

We require a PBS scheme to satisfy the following properties.

- **Correctness:** For all non-uniform polynomial time adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\begin{array}{c} \nu(m) = 1 \\ \wedge \\ \mathsf{Verify}(\mathsf{mpk}, m, \sigma) \neq 1 \end{array} : \begin{array}{r} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\nu, m, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{mpk}) \\ \mathsf{sk}_\nu \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \nu) \\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_\nu, m) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $m \in \{0,1\}^{\ell(\lambda)}$ and ν is efficiently computable.

- **Unforgeability:** For all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \nu(m) = 0 \\ \wedge \\ \mathsf{Verify}(\mathsf{mpk}, m, \sigma) = 1 \end{array} : \begin{array}{r} (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\nu, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{mpk}) \\ \mathsf{sk}_\nu \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \nu) \\ (\sigma, m) \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{sk}_\nu) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where ν is an efficiently computable predicate.

3.5 Extractable Witness Encryption

A witness encryption (WE) scheme [GGSW13] allows encrypting a message with respect to a statement such that decryption is only possible if one has the corresponding witness. In this work we use the stronger notion of extractable witness encryption [GKP+13, GGHW14].

While WE is known from well-founded assumptions [JLS21] or from variants of LWE [Tsa22, VWW22], extractable WE is not known from standard assumptions. However, known candidates for WE could be assumed to also be extractable. While there are conditional negative results for extractable WE [GGHW14], they are with respect to specific auxiliary input distributions and assuming strong forms of obfuscation. Recent work has shown that blockchains could help realize a primitive that is equivalent to extractable witness encryption [GKM+22].

Definition 7 (Extractable Witness Encryption)**.** Let $\mathcal{L}$ be an NP language with relation $\mathcal{R}$ and let $\ell \in \mathsf{poly}(\lambda)$. A witness encryption scheme $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ for $\mathcal{L}$ is a tuple of algorithms with the following syntax.

- $\mathsf{Enc}(1^\lambda, x, m) \to \mathsf{ct}$ is a PPT algorithm that takes as input a statement $x \in \mathcal{L}$ and a message $m \in \{0,1\}^{\ell(\lambda)}$ and outputs a ciphertext ct.

- $\mathsf{Dec}(w, \mathsf{ct}) \to m$ is a PPT algorithm that takes as input a witness w and a ciphertext ct and either outputs $\perp$ or a message $m \in \{0,1\}^{\ell(\lambda)}$.

We require that an Extractable WE scheme satisfy the following properties.

- **Correctness:** For every $\lambda \in \mathbb{N}$, $(x, w) \in \mathcal{R}$ and $m \in \{0,1\}^{\ell(\lambda)}$, we have

$$\Pr\Big[\mathsf{Dec}(w, \mathsf{ct}) = m : \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, x, m)\Big] = 1.$$

- **Extractability:** For all PPT adversaries $\mathcal{A}$ and polynomials p_a, there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ and a polynomial p_e such that for all sufficiently large $\lambda \in \mathbb{N}$, $x \in \mathcal{L}$, $m_0, m_1 \in \{0,1\}^{\ell(\lambda)}$, and polynomial length auxiliary input $z \in \{0,1\}^*$ we have

$$\Pr\left[\mathcal{A}(x, \mathsf{ct}, z) = b : \begin{array}{r} b \leftarrow \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, x, m_b) \end{array}\right] \geq \frac{1}{2} + \frac{1}{p_a(\lambda)}$$
$$\implies \Pr[(x, w) \in \mathcal{R} : \mathcal{E}_\mathcal{A}(x, z) = w] \geq \frac{1}{p_e(\lambda)}.$$

3.6 Incrementally Verifiable Computation

Incrementally verifiable computation (IVC), first introduced by Valiant [Val08], allows incrementally computing a proof for a computation so that any intermediate state can be verified and moreover, a proof for an intermediate state can be efficiently updated to compute a proof for subsequent steps of the computation. However, IVC as defined by Valiant, supports only deterministic computation. Specifically, a prover convinces a verifier that a Turing machine, the description of which is known to both, reaches a particular state after a specific number of steps of its execution. Chiesa and Tromer [CT10] generalized the notion of IVC to Proof Carrying Data (PCD) which supports proving that an output was obtained through a computation that is compliant. Specifically, given inputs, outputs, proofs of correct computation for each input and a compliance predicate for the computation mapping the inputs to the output, a PCD scheme allows computing a proof of correctness for the output. Our definition of IVC is adapted from the definition of PCD by Bünz et al. [BCMS20] and is a special case of a PCD where each step of the computation takes only one new input i.e., in the language used by Bünz et al., the transcript is a simple directed path. Alternatively, it is a generalization of Valiant's definition of IVC where each step of the Turing machine may take a private input.

Currently, we know instantiations of IVC for non-deterministic computations from knowledge assumptions (e.g., [BCCT13, BCMS20]) or from random oracles [Val08]. [HAN23] discuss the impossibility of constructing IVC with zero-knowledge assuming the existence of collision resistant hash functions or perfectly binding rerandomizable commitments. However, we remark that in this work we only make use of knowledge sound IVC schemes with non-black box extraction and do not do not require the IVC scheme to be zero-knowledge. Such IVC schemes are not ruled out by [HAN23].

We first recall the definition of compliance predicates and the relation defined by them on computations and then continue to define IVC.

Definition 8 (Relations From Compliance Predicates)**.** Let $m, n \in \mathsf{poly}(\lambda)$. Let $\phi : \{0,1\}^{n(\lambda)} \times \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)} \to \{0,1\}$ be an efficiently computable compliance predicate. Let $T = (\mathsf{st}_0, (\mathsf{st}_1, w_1), \ldots, (\mathsf{st}_\ell, w_\ell))$ be a computation trace where st_0 denotes the initial state and each intermediate state st_i is obtained as the output of computing on the previous state st_{i-1} and an input w_i. st_ℓ denotes the output of the computation. T is said to be ϕ-compliant if $\phi(\mathsf{st}_0, \perp, \perp) = 1$ and for each $i \in [\ell]$, $\phi(\mathsf{st}_{i-1}, \mathsf{st}_i, w_i) = 1$.

The language $\mathcal{L}_\phi$ and the corresponding witness relation $\mathcal{R}_\phi$ are defined as

$$\mathcal{R}_\phi = \{(\mathsf{st}, T) \mid \mathsf{st} \text{ is output of } T \wedge T \text{ is } \phi\text{-compliant}\}$$
$$\mathcal{L}_\phi = \{\mathsf{st} \mid \exists T \text{ such that } (\mathsf{st}, T) \in \mathcal{R}_\phi\}.$$

Definition 9 (Incrementally Verifiable Computation)**.** Let Φ be a set of compliance predicates. An IVC scheme $\mathsf{IVC} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for Φ is a tuple of algorithms with the following semantics.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$ is a PPT algorithm that outputs the common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, \phi, \mathsf{st}, \pi, w, \mathsf{st}') \to \pi'$ is a PPT algorithm that takes the common reference string crs, a predicate $\phi \in \Phi$, a state st, a proof of the correctness π for st, a witness w and the next state st' and outputs a proof of correctness π' for st'.

- $\mathsf{Verify}(\mathsf{crs}, \phi, \mathsf{st}, \pi) =: b$ is a polynomial time algorithm that takes the common reference string crs, a predicate $\phi \in \Phi$, a state st and a proof π and outputs a bit $b \in \{0,1\}$ denoting if the proof is accepted or rejected.

  We require an IVC scheme to satisfy the following properties.

- **Completeness:** For all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \left(\begin{array}{c} \phi \in \Phi \ \wedge \ \phi(\mathsf{st}', \mathsf{st}, w) = 1 \ \wedge \\ (\mathsf{st} = \perp \vee \mathsf{Verify}(\mathsf{crs}, \phi, \mathsf{st}, \pi) = 1) \end{array}\right) \\ \Downarrow \\ \mathsf{Verify}(\phi, \mathsf{st}', \pi') = 1 \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\phi, \mathsf{st}, \pi, w, \mathsf{st}') \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \pi' \leftarrow \mathsf{Prove}(\mathsf{crs}, \phi, \mathsf{st}, \pi, w, \mathsf{st}') \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Soundness:** For all PPT adversaries $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}_\mathcal{A}$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and polynomial length auxiliary input $z \in \{0,1\}^*$, we have

$$\Pr\left[\begin{array}{c} \phi \in \Phi \wedge \mathsf{st} = \mathsf{out}(T) \\ \wedge \\ \mathsf{Verify}(\mathsf{crs}, \phi, \mathsf{st}, \pi) = 1 \\ \wedge \\ T \text{ is not } \phi\text{-compliant} \end{array} : \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\phi, \mathsf{st}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}, z) \\ T \leftarrow \mathcal{E}_\mathcal{A}(\mathsf{crs}, z) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{out}(T)$ denotes the output of the computation corresponding to T.

- **Efficiency:** For all $\phi \in \Phi$, common reference strings crs, and $(\mathsf{st}, T) \in \mathcal{R}_\phi$, the size of the proof π, computed by repeatedly running Prove, is at most $|\pi| \in \mathsf{poly}(\lambda\ |\phi| \log |T|)$.

4 Hop Tracking Signatures

An HTS scheme allows a user to sign a message m, producing a signature σ that preserves the anonymity of the signer. The signature can then be *forwarded*, while keeping track of the number of unique users who have forwarded it, so that the source can be deanonymized once this count exceeds a specified threshold. As discussed in Section 2.2, HTS schemes can be naturally composed with E2EE messaging systems to allow tracing the source of *viral* messages, where virality here means that the message's forwarding graph satisfies the unique-forwarders-on-a-path predicate.

The syntax and properties of HTS schemes are described in Definition 10. Informally, the setup procedure outputs a master public key mpk available to all users, the registration server's secret key rsk, and the E2EE server's master secret key msk. New users run a registration protocol with the registration server to obtain their keypair, while the server records the user's public key. Registered users can then use their secret keys to sign messages and forward signatures via the Sign and $\mathsf{Forward}$ algorithms, respectively. Once a signature has been forwarded by t_{trace} distinct users, the E2EE server, using its secret key msk, can deanonymize the source by running the Open algorithm.

Crucially, deanonymizing the source requires the master secret key msk, which is *constrained*: it cannot be used to reveal the source identity if the signature has been forwarded by fewer than t_{anon} distinct users, and it always reveals the source identity once the signature has been forwarded by at least t_{trace} distinct users. This property, along with other guarantees—such as the signature preserving the anonymity of intermediate forwarders—is captured by the following properties.

Correctness: Correctness defines the behavior of the scheme when all users and servers are honest. It requires that a signature that has been honestly signed and forwarded satisfies the following properties: (1) it verifies successfully, (2) it does not reveal the identity of the source when input to the Open algorithm if the number of distinct forwarders is less than t_{anon}, (3) it always opens to the source identity if the number of distinct forwarders is at least t_{trace}, and (4) if it opens at any point when the number of distinct forwarders lies between t_{anon} and t_{trace}, it opens only to the identity of the source. To ensure that this property holds regardless of the order of registration and forwards, we formalize it through an experiment in which an adversary schedules user registrations and the sequence of forwards.

Source Anonymity: Source anonymity guarantees that a signature ensures anonymity of the signer against an adversary that either corrupts the E2EE server or the registration server. Source anonymity against a corrupt registration server is formally captured by the experiment described in Figure 3. In this experiment, the adversary is given the registration server's secret key and outputs a challenge message m along the identities of two honest users, upk_0 and upk_1. It must then determine whether a challenge signature σ on m was generated by upk_0 or upk_1. The only restriction[13] is that the adversary cannot query the Trace oracle on signatures of the challenge message m. Otherwise, the adversary could trivially break source anonymity by forwarding σ sufficiently many times and then invoking Trace to recover the source identity.

Source anonymity against a corrupt E2EE server is formally captured by the experiment described in Figure 2. It proceeds similarly to the source anonymity experiment for the registration server, except that the adversary is given the master secret key msk instead of the registration server's secret key. Importantly, the adversary is allowed to corrupt fewer than t_{anon} parties and is restricted from using honest parties to forward signatures on the challenge message m. This prevents the adversary from trivially breaking source anonymity by forwarding the signature sufficiently many times. Note that the definition captures the scenario where an adversary that corrupts $t < t_{\mathsf{anon}} - 1$ users receives a signature forwarded by $t_{\mathsf{anon}} - t - 1$ honest users; in the experiment, the adversary honestly forwards the signature using $t_{\mathsf{anon}} - t - 1$ of the users it corrupts. However, the definition only guarantees *selective* security with respect to the choice of message and the corruptions. Ideally, we would like to allow the adversary to select the challenge message m and adaptively corrupt honest users through the Cor oracle, as in Figure 3. Looking ahead, selective security is an artifact of our proof strategy and it is unclear whether the proof of security for our construction can be extended to establish the stronger adaptive security guarantee. Accordingly, we adopt the selective security definition when formalizing HTS schemes and elaborate further on this point in Section 6.

Forwarder Anonymity: Forwarder anonymity requires that signatures hide the identity of intermediate forwarders. However, defining this property against an adversary capable of creating arbitrary forwarding paths requires some care. Specifically, if a user forwards a signature and later forwards it again, the number of distinct forwarders does not increase, allowing the adversary to "link" signatures forwarded by the same user. To address this, we adopt a definition similar to that used for linkable ring signatures [BDH+19]. Specifically, the adversary

[13] The adversary is not given access to the Reg oracle, but it is unnecessary since it can locally generate keypairs for corrupt users using the registration server's secret key.

```
$\mathsf{ExpSrcAnon}^{\mathsf{open}}_{\mathsf{HTS},\mathcal{A}}(1^\lambda, m)$:
1: Initialize $\mathcal{I} := \emptyset$ and $\mathcal{H} := \emptyset$. Compute $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
2: Let
       $(\mathsf{upk}_0, \mathsf{upk}_1, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{RegH},\mathsf{Reg},\mathsf{HSign},\mathsf{HFwd}}(\mathsf{mpk}, \mathsf{msk})$.
   If there do not exist $\mathsf{usk}_0$ and $\mathsf{usk}_1$ such that $(\mathsf{upk}_0, \mathsf{usk}_0)$ and $(\mathsf{upk}_1, \mathsf{usk}_1)$ are in $\mathcal{H}$ then output $0$.
3: Sample $b \leftarrow \{0,1\}$ uniformly at random and compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}_b, m)$.
4: Let $b' \leftarrow \mathcal{A}^{\mathsf{RegH},\mathsf{Reg},\mathsf{HSign},\mathsf{HFwd}}(\mathsf{st}, \sigma)$.
5: Output 1 if
       $|\mathcal{I}| < t_{\mathsf{anon}} \ \wedge \ m \notin \mathcal{M}_{\mathsf{HFwd}} \ \wedge \ b = b'$,
   where $\mathcal{M}_{\mathsf{HFwd}}$ is the set of messages queried to HFwd. Else, output $0$.
```

Figure 2: HTS source anonymity experiment for opener. Key differences from Figure 3 are highlighted in red.

```
$\mathsf{ExpSrcAnon}^{\mathsf{reg}}_{\mathsf{HTS},\mathcal{A}}(1^\lambda)$:
1: Initialize $\mathcal{I} := \emptyset$ and $\mathcal{H} := \emptyset$. Compute $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
2: Let
       $(m, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{RegH},\mathsf{Cor},\mathsf{HSign},\mathsf{HFwd},\mathsf{Trace}}(\mathsf{mpk}, \mathsf{rsk})$.
   If there do not exist $\mathsf{usk}_0$ and $\mathsf{usk}_1$ such that $(\mathsf{upk}_0, \mathsf{usk}_0)$ and $(\mathsf{upk}_1, \mathsf{usk}_1)$ are in $\mathcal{H}$ then output $0$.
3: Sample $b \leftarrow \{0,1\}$ uniformly at random and compute $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}_b, m)$.
4: Let $b' \leftarrow \mathcal{A}^{\mathsf{RegH},\mathsf{Cor},\mathsf{HSign},\mathsf{HFwd},\mathsf{Trace}}(\mathsf{st}, \sigma)$.
5: Output 1 if
       $m \notin \mathcal{M}_{\mathsf{Trace}} \ \wedge \ b = b'$,
   where $\mathcal{M}_{\mathsf{Trace}}$ is the set of messages queried to Trace. Else, output $0$.
```

Figure 3: HTS source anonymity experiment for registration server. Key differences from Figure 2 are highlighted in red.

outputs a challenge message m and the identities of two honest users, upk_0 and upk_1. It must then determine whether requesting a forward by upk_0 for a signature on m results in upk_0 forwarding the signature or in upk_1 forwarding the signature, and similarly for upk_1. The only restriction is that the adversary must output the challenge message and the identities of upk_0 and upk_1 *before* issuing any forwarding queries on m involving these users via the HFwd oracle. This prevents the adversary from trivially breaking forwarder anonymity by linking multiple forwards made by the same user.

Traceability and Unframeability: Traceability requires that any valid signature generated by an adversary that corrupts the E2EE server must open to the identity of a corrupt user—specifically, one it obtains through the Reg or Cor oracles—after it has been forwarded by at least t_{trace} distinct honest users. Unframeability is similar, except that the adversary additionally corrupts the registration server. In this case, we require that the signature does not open to the identity of an honest user i.e., while the adversary may locally generate keypairs and produce signatures that open to unregistered users, it must be unable to frame any honest user. Note that the definition requires *consecutively* forwarding by t_{trace} honest users to prevent the adversary from mounting refresh attacks, as discussed in Section 2.1.1.

Definition 10 (Hop Tracking Signature). Let $t_{\mathsf{anon}}, t_{\mathsf{trace}} \in \mathsf{poly}(\lambda)$. A $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-secure Hop Tracking Signature scheme $\mathsf{HTS} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{Sign}, \mathsf{Forward}, \mathsf{Verify}, \mathsf{Open})$ is a tuple with the following semantics.

```
$\mathsf{ExpTrace}_{\mathsf{HTS},\mathcal{A}}(1^\lambda)$:
1: Initialize $\mathcal{I} := \emptyset$ and $\mathcal{H} := \emptyset$. Compute (mpk, msk, rsk) $\leftarrow \mathsf{Setup}(1^\lambda)$.
2: Let
       $(m, \sigma_0, \mathsf{stop}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Reg}}(\mathsf{mpk}, \mathsf{msk})$,
   where $\mathcal{O}$ denotes the oracles (RegH, Cor, HSign, HFwd). If $\sigma_0$ does not verify successfully, output 0.
3: Initialize $i := 0$. While $\mathsf{stop} = 0$, run
       $(\mathsf{upk}_{i+1}, \mathsf{stop}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O},\mathsf{Reg}}(\mathsf{st}, \sigma_i)$
           $\sigma_{i+1} \leftarrow \mathsf{Forward}(\mathsf{usk}_{i+1}, m, \sigma_i)$
           $i := i + 1$
   where $(\mathsf{upk}_{i+1}, \mathsf{usk}_{i+1}) \in \mathcal{H}$ for each $i$. If for any $i$ there does not exist $(\mathsf{upk}_{i+1}, \mathsf{usk}_{i+1})$ in $\mathcal{H}$, then output 0. Let the counter $i$ be equal to $\ell$ when $\mathsf{stop} = 1$.
4: Output 1 if
       $m \notin \mathcal{M}_{\mathsf{HSign}} \ \wedge \ m \notin \mathcal{M}_{\mathsf{HFwd}} \ \wedge$
           $L \geq t_{\mathsf{trace}} \ \wedge \ \mathsf{Open}(\mathsf{msk}, m, \sigma_\ell) \notin \mathcal{I}$
   where $\mathcal{M}_{\mathsf{HSign}}$ and $\mathcal{M}_{\mathsf{HFwd}}$ are the set of messages queried to HSign and HFwd respectively, and $L = \left|\{\mathsf{upk}_i\}_{i=1}^{\ell}\right|$ is the number of distinct public keys in $(\mathsf{upk}_1, \ldots, \mathsf{upk}_\ell)$.
```

Figure 4: HTS traceability experiment. Differences from Figure 5 are highlighted in red.

```
$\mathsf{ExpFrame}_{\mathsf{HTS},\mathcal{A}}(1^\lambda)$:
1: Initialize $\mathcal{I} := \emptyset$ and $\mathcal{H} := \emptyset$. Compute (mpk, msk, rsk) $\leftarrow \mathsf{Setup}(1^\lambda)$.
2: Let
       $(m, \sigma_0, \mathsf{stop}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk})$,
   where $\mathcal{O}$ denotes the oracles (RegH, Cor, HSign, HFwd). If $\sigma_0$ does not verify successfully, output 0.
3: Initialize $i := 0$. While $\mathsf{stop} = 0$, run
       $(\mathsf{upk}_{i+1}, \mathsf{stop}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{st}, \sigma_i)$
           $\sigma_{i+1} \leftarrow \mathsf{Forward}(\mathsf{usk}_{i+1}, m, \sigma_i)$
           $i := i + 1$
   where $(\mathsf{upk}_{i+1}, \mathsf{usk}_{i+1}) \in \mathcal{H}$ for each $i$. If for any $i$ there does not exist $(\mathsf{upk}_{i+1}, \mathsf{usk}_{i+1})$ in $\mathcal{H}$, then output 0. Let the counter $i$ be equal to $\ell$ when $\mathsf{stop} = 1$.
4: Output 1 if
       $m \notin \mathcal{M}_{\mathsf{HSign}} \ \wedge \ m \notin \mathcal{M}_{\mathsf{HFwd}} \ \wedge$
           $L \geq t_{\mathsf{trace}} \ \wedge \ \mathsf{Open}(\mathsf{msk}, m, \sigma_\ell) \in \mathcal{H}$
   where $\mathcal{M}_{\mathsf{HSign}}$ and $\mathcal{M}_{\mathsf{HFwd}}$ are the set of messages queried to HSign and HFwd respectively, and $L = \left|\{\mathsf{upk}_i\}_{i=1}^{\ell}\right|$ is the number of distinct public keys in $(\mathsf{upk}_1, \ldots, \mathsf{upk}_\ell)$.
```

Figure 5: HTS unframeability experiment. Differences from Figure 4 are highlighted in red.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk})$ is a PPT setup algorithm, run by a trusted party, that outputs the master public key mpk, the master secret key msk and the registration key rsk.
- $\mathsf{Register}\langle S_{\mathsf{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{mpk})\rangle \to \langle \mathsf{upk}, (\mathsf{upk}, \mathsf{usk})\rangle$ is a registration protocol between the registration server S_{reg} with input rsk and a user U with input mpk. At the end of the protocol, the server and the user obtain the user's public key upk and the user additionally obtains its secret key usk.
- $\mathsf{Sign}(\mathsf{usk}, m) \to \sigma$ is a PPT signing algorithm run by registered users that takes a user's secret key usk and a message $m \in \{0,1\}^*$ as input and outputs a signature σ.
- $\mathsf{Forward}(\mathsf{usk}, m, \sigma) \to \sigma'$ is a PPT algorithm that takes a user's secret key usk, a message m and a signature σ on m and outputs a new signature σ' on m.
- $\mathsf{Verify}(\mathsf{mpk}, m, \sigma) =: b$ is a polynomial time algorithm that takes the master public key mpk, a message m and a signature σ as input and outputs a bit $b \in \{0,1\}$ denoting if σ is valid.

- $\mathsf{Open}(\mathsf{msk}, m, \sigma) =: \mathsf{upk}$ is a polynomial time algorithm that takes the master secret key msk, a message m and a signature σ on m as input and either outputs $\perp$ or a public key upk.

We model the adversary's capabilities in security definitions using the following oracles. The oracles are described using the keys $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk})$ generated by the experiment in the setup phase, and a set of corrupt users $\mathcal{I}$ and honest users $\mathcal{H}$ initialized by the experiment.

- $\mathsf{RegH}()$: Allows the adversary to register honest users. When called, it locally runs the registration protocol $\mathsf{Register}\langle S_{\mathsf{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{mpk})\rangle$ to compute $(\mathsf{upk}, \mathsf{usk})$ which it then adds to $\mathcal{H}$. It returns upk to the adversary.
- $\mathsf{Reg}()$: Allows the adversary to register corrupt users. When called, it acts as the registration server $S_{\mathsf{reg}}(\mathsf{rsk})$ in an execution of $\mathsf{Register}$ and adds the public key upk of the registered corrupt user to $\mathcal{I}$.
- $\mathsf{Cor}(\mathsf{upk})$: Allows the adversary to corrupt honest users. When called with a user's public key upk, if $(\mathsf{upk}, \mathsf{usk}) \in \mathcal{H}$ it returns usk to the adversary and adds upk to $\mathcal{I}$.
- $\mathsf{HSign}(\mathsf{upk}, m)$: Allows the adversary to obtain signatures by honest users on messages of its choice. When called with a user's public key upk and a message m, if $(\mathsf{upk}, \mathsf{usk}) \in \mathcal{H}$, it computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{usk}, m)$ and returns σ to the adversary.
- $\mathsf{HFwd}(\mathsf{upk}, m, \sigma)$: Allows the adversary to compute forwards on behalf of honest users. When called with a user's public key upk, a message m and a signature σ, if $(\mathsf{upk}, \mathsf{usk}) \in \mathcal{H}$, it computes $\mathsf{Forward}(\mathsf{usk}, m, \sigma)$ to obtain σ' and returns σ' to the adversary.
- $\mathsf{Trace}(m, \sigma)$: Allows the adversary to trace the source of signatures. When called with a message m and signature σ, it runs $\mathsf{upk} := \mathsf{Open}(\mathsf{msk}, m, \sigma)$ and returns upk to the adversary.

We require that a $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-secure HTS scheme HTS satisfy the following properties.

- **Correctness:** HTS is $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-correct if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{mpk}, m, \sigma_\ell) \neq 1 \\ \vee\ \mathsf{upk}' \notin \{\perp, \mathsf{upk}_0\} \\ \vee\ \left(L < t_{\mathsf{anon}}\ \wedge\ \mathsf{upk}' \neq \perp\right) \\ \vee\ \left(L \geq t_{\mathsf{trace}}\ \wedge\ \mathsf{upk}' \neq \mathsf{upk}_0\right) \end{array} : \begin{array}{r} (\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m, \mathsf{upk}_0, \ldots, \mathsf{upk}_\ell) \leftarrow \mathcal{A}^{\mathsf{RegH}}(\mathsf{mpk}) \\ \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{usk}_0, m) \\ \sigma_i \leftarrow \mathsf{Forward}(\mathsf{usk}_i, m, \sigma_{i-1}),\ \forall i \in [\ell] \\ \mathsf{upk}' := \mathsf{Open}(\mathsf{msk}, m, \sigma_\ell) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $L = \left|\{\mathsf{upk}_i\}_{i=1}^{\ell}\right|$ is the number of distinct public keys in $(\mathsf{upk}_1, \ldots, \mathsf{upk}_\ell)$, and $(\mathsf{upk}_i, \mathsf{usk}_i)$ is in $\mathcal{H}$ for each $i \in [0, \ell]$.

- **Source Anonymity:** HTS is t_{anon}-source anonymous against the *opener* if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\ell_{\mathsf{msg}} \in \mathsf{poly}(\lambda)$ and messages $m \in \{0,1\}^{\ell_{\mathsf{msg}}}$,

$$\Pr\left[\mathsf{ExpSrcAnon}^{\mathsf{open}}_{\mathsf{HTS},\mathcal{A}}(1^\lambda, m) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{ExpSrcAnon}^{\mathsf{open}}_{\mathsf{HTS},\mathcal{A}}$ is defined in Figure 2.

HTS is source anonymous against the *registration server* if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\Big[\mathsf{ExpSrcAnon}^{\mathsf{reg}}_{\mathsf{HTS},\mathcal{A}}(1^\lambda) = 1\Big] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathsf{ExpSrcAnon}^{\mathsf{reg}}_{\mathsf{HTS},\mathcal{A}}$ is defined in Figure 3.

- **Forwarder Anonymity:** HTS is forwarder anonymous if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\left[\begin{array}{c} \mathsf{upk}_0, \mathsf{upk}_1 \notin \mathcal{I} \wedge \\ (\mathsf{upk}_0, m) \notin \mathcal{J} \wedge \\ (\mathsf{upk}_1, m) \notin \mathcal{J} \wedge \\ b = b' \end{array} : \begin{array}{r} (\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (m, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{st}) \leftarrow \mathcal{A}^{*,\mathsf{HFwd}}(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \\ b \leftarrow \{0,1\} \\ b' \leftarrow \mathcal{A}^{*,\mathsf{HFwd}_b}(\mathsf{st}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

  where the secret keys corresponding to upk_0 and upk_1 are in $\mathcal{H}$, $\mathcal{J}$ is the set of message and user identity pairs queried to HFwd, HFwd_0 is identical to HFwd, and HFwd_1 is identical to HFwd except that $\mathsf{HFwd}_1(\mathsf{upk}_0, m, \sigma)$ returns $\mathsf{Forward}(\mathsf{usk}_1, m, \sigma)$ and $\mathsf{HFwd}_1(\mathsf{upk}_1, m, \sigma)$ returns $\mathsf{Forward}(\mathsf{usk}_0, m, \sigma)$ for any signature σ. $\mathcal{A}^*$ is used to denote that $\mathcal{A}$ has oracle access to RegH, Cor, and HSign.

- **Traceability:** HTS is t_{trace}-traceable if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\Big[\mathsf{ExpTrace}_{\mathsf{HTS},\mathcal{A}}(1^\lambda) = 1\Big] \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{ExpTrace}_{\mathsf{HTS},\mathcal{A}}$ is defined in Figure 4.

- **Unframeability:** HTS is t_{trace}-unframeable if for every non-uniform polynomial time adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ we have

$$\Pr\Big[\mathsf{ExpFrame}_{\mathsf{HTS},\mathcal{A}}(1^\lambda) = 1\Big] \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{ExpFrame}_{\mathsf{HTS},\mathcal{A}}$ is defined in Figure 5.

When the tracing and anonymity thresholds are equal, we simply say that the HTS scheme is t-secure.

Succinct HTS schemes. We say an HTS scheme is *succinct* if the size of signatures is sublinear in the number of forwards. Non-succinct HTS schemes, where the size of the signature grows linearly with the number of forwards, are infeasible for applications like source tracing of viral messages where the number of forwards are expected to be relatively large.

Definition 11 (Succinct Hop Tracking Signature Scheme)**.** An HTS scheme HTS is said to be succinct if there exists a sublinear function $g(x) \in \mathsf{o}(x)$ and a polynomial $p(x) \in \mathsf{poly}(x)$ such that for all $\ell_{\mathsf{msg}} \in \mathsf{poly}(\lambda)$, all $\lambda, \ell \in \mathbb{N}$, all messages $m \in \{0,1\}^{\ell_{\mathsf{msg}}}$, all registration keys rsk in the support of $\mathsf{Setup}(1^\lambda)$ and user secret keys $(\mathsf{usk}_0, \ldots, \mathsf{usk}_\ell)$ in the support of Register, we have,

$$|\sigma_\ell| = g(\ell)p(\lambda)$$

with probability 1 where $\sigma_0 \leftarrow \mathsf{Sign}(\mathsf{usk}_0, m)$, $\sigma_i \leftarrow \mathsf{Forward}(\mathsf{upk}_i, m, \sigma_{i-1})$ for each $i \in [\ell]$, and the probability is over the randomness used for signing and forwarding the signature.

4.1 Discussion on the Unique Forwarders on a Path Predicate

In this work, we consider primitives that allow source tracing for messages where the corresponding forwarding graph satisfies the unique-forwarders-on-a-path predicate i.e., the number of forwarders between the source and the final recipient is at least a threshold t. However, as discussed in Section 2.1.1, corrupt parties can always prevent traceability by acting as a new source and sending the message anew or resorting to "out-of-band" communication with other corrupt parties. As a result, traceability is only possible if t honest users forward a message sequentially. In this section, we explore whether such a traceability guarantee is meaningful in practice.

In practice, the adversary corrupts a set of parties in the system. Then a certain message is signed and forwarded throughout users. We model this as a *random graph* where the vertices correspond to the users and the edges correspond to the message path in the system. In a random graph $G_{N,p}$, given N vertices, the edges are sampled according to some probability, say $p = c_0/N$ where $c_0 = \mathcal{O}(1)$ is a constant. This provides a simple yet easy-to-analyze model of how a message propagates through the system: we can think of c_0 as the size of a user's contact list, with the user forwarding the message to all of its contacts. It is well-known that a random graph $G_{N,p}$ has a path of length $c_1 N$ except with negligible probability in N, where $c_1 < 1$ is a constant depending on c_0 [FK15].

Additionally, we show that a uniformly chosen subgraph H with $(1 - \epsilon)N$ vertices *still* has a path of length $c_2 N$ except with negligible probability in N, where $\epsilon < 1$ is a constant. We can think of the subgraph H as the set of honest users in the system. This means that, even if the adversary corrupts a uniformly random set of users of size ϵN, there is a sufficiently long path containing *only* honest users, except with negligible probability, and thus we can trace the source identity. In other words, we can guarantee that even if the adversary corrupts a *fraction* of the entire system, we can still deanonymize the source of a viral message.

Paths In Random Graphs. A random graph $G_{N,p}$ is a graph with N vertices and where each edge is sampled independently with probability p. It is well-known that random sparse graphs have long paths. Let G be a graph and let H be a subgraph, we denote by $G \setminus H$ the subgraph of G where the vertices of H are removed.

Lemma 1 ([FK15])**.** *Let $p = c/N$ where $c = \mathcal{O}(\log N)$. Then $G_{N,p}$ has a path of length at least*

$$\left(1 - \frac{6 \log c}{c}\right) N$$

except with negligible probability in N.

Let G be a random graph a let H be a random subgraph. We show that the graph $G \setminus H$ still has a long path.

Lemma 2. *Let $p = c/N$ where $c = \mathcal{O}(1)$ is a constant and $G = G_{N,p}$ be a random graph. Let H be a uniformly chosen random subgraph of size $k = \varepsilon \cdot N$ where $\varepsilon < 1$. Then the graph $G \setminus H$ has a path of length $d \cdot N$ where d is a constant, except with negligible probability in N.*

Proof. Let $F = G \setminus H$. F has $(1 - \varepsilon)N$ vertices and it is a random graph with $p = c/N$. Set $c' = (1 - \varepsilon) \cdot c$. Then, $F = F_{(1-\varepsilon)N,p}$ where $p = c'/((1 - \varepsilon)N)$. To see this note that

$$p = \frac{c}{N} = \frac{c'}{(1 - \varepsilon)N}.$$

We can resort to Lemma 1 to argue that F has a path of length

$$\left(1 - \frac{6 \log c'}{c'}\right) \cdot (1 - \varepsilon)N \in d \cdot N$$

except with negligible probability in N, for some constant d. ■

Example. As a concrete example, set $N = 2B$ (two billion, which is roughly the number of Whatsapp users) and let H be the nodes that an adversary corrupts. Let εN be the size of H. Lemma 2 states that, in this case, $G \setminus H$ will have a path of length $d \cdot N$ for some constant d, except with negligible probability. If we set the tracing threshold to $1M$ (one million forwards), then the size of H can be roughly $\approx 0.4 \cdot N$ and $G \setminus H$ will have a path of size $1M$ except with negligible probability. This means that, even a very powerful adversary corrupting $\approx 0.4 \cdot N = 0.8B$ can be traced except with negligible probability.

5 Impossibility Of t-secure Succinct HTS

In this section we show that t-secure *succinct* HTS schemes cannot exist using a well known lower bound in communication complexity. We first recall relevant details in communication complexity; specifically the ϵ-error probabilistic communication complexity $C_\epsilon(f)$ of a boolean function f, introduced by Yao [Yao79]. Consider two computationally unbounded parties, each holding an n bit input x and y such that neither party knows the other's input. The parties wish to compute the output of a function $f : \{0,1\}^{2n} \to \{0,1\}$ using a pre-determined protocol. To compute the output, they are allowed to make random decisions as well as send messages to each other. At the end of the protocol, the parties must output $f(x, y)$ with probability at least $1 - \epsilon$ for any $x, y \in \{0,1\}^n$. $C^A(f)$ is defined as the expected number of bits communicated between the two parties when computing f on the worst case input under the protocol A and $C_\epsilon(f)$ is defined as the infimum of the set of all $C^A(f)$ where the output of protocol A is correct with probability at least $1 - \epsilon$. In other words, $C_\epsilon(f)$ is the expected communication on the worst case input for any protocol that computes f with at most ϵ error.

Let $\mathsf{DIS}_n : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ denote the disjointness function such that $\mathsf{DIS}_n(x, y) = 1$ iff for all $i \in [n]$, either $\mathbf{x}_i \neq 1$ or $\mathbf{y}_i \neq 1$. Alternatively, let $X, Y \subseteq [n]$ denote the set of indices where x and y contain 1, respectively. Then, $\mathsf{DIS}_n(x, y) = 1$ iff $|X \cap Y| = 0$. Our main result for this section follows from a lower bound established by Kalyanasundaram and Schnitger [KS92] who showed that for any fixed $\epsilon < 1/2$, $C_\epsilon(\mathsf{DIS}_n) \in \Omega(n)$.

Informally, we show that any (t, t)-secure HTS signature that has been forwarded $\ell = \mathcal{O}(t)$ times has expected size linear in t. In other words, the signature size grows linearly with the number of forwards and is therefore not succinct. Our result holds for non-constant thresholds t—so the only parameter regime where we can hope to have a succinct t-secure HTS scheme is for a constant t, where succinctness follows trivially from the fact that the signature needs to keep track of only a constant number of forwarders. However, such a scheme does not provide any meaningful notion of privacy since the source can be de-anonymized after a constant number of forwards. Looking ahead, we show in Section 6 that having a gap in the anonymity and traceability thresholds suffices to circumvent the impossibility result and realize a succinct HTS scheme where the signature size is polylogarithmic in the number of forwards.

Theorem 3. *Let $t \in \mathsf{poly}(\lambda)$ be a non-constant polynomial. Then, any (t,t)-secure HTS scheme cannot be succinct (Definition 11).*

Proof. We will first use a (t,t)-correct HTS scheme and a PRG to construct a protocol for computing the disjointness function, where the communication complexity of the protocol depends on the size of the signature. We will then leverage known lower bounds on the communication complexity of any protocol that computes the disjointness function, to derive a lower bound on the size of the signature. This lower bound on the signature size is conditioned on the existence of one-way functions and relies solely on the correctness property of the HTS scheme. We conclude the proof by showing that the source anonymity property implies the existence of one-way functions, which in turn unconditionally establishes the impossibility of succinct (t,t)-secure HTS schemes.

Claim. *Assuming one-way functions exist, any (t,t)-correct HTS scheme cannot be succinct.*

Proof. Assume for the sake of contradiction that there exists a (t,t)-correct succinct HTS scheme HTS for some non-constant polynomial t. Let λ denote the security parameter of HTS and let $n = t(\lambda) - 1$. Let G be a PRG and let $\lambda_1 = \sqrt{t(\lambda)}$ be the security parameter of the PRG. Consider two parties, Alice and Bob, with inputs $x, y \in \{0,1\}^n$ that run the following protocol to compute $\mathsf{DIS}_n(x,y)$.

1. Alice samples $s \leftarrow \{0,1\}^{\lambda_1}$ uniformly at random.
2. Alice uses $G(s)$ as the random tape to compute $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{HTS.Setup}(1^\lambda)$ and user keys $\{(\mathsf{upk}_i, \mathsf{usk}_i)\}_{i=1}^{2n}$ using $\mathsf{HTS.Register}$.
3. Alice computes $\sigma_0 \leftarrow \mathsf{HTS.Sign}(\mathsf{usk}_0, 0)$.
4. For each $i \in [n]$, Alice does one of the following.
   - If $\mathbf{x}_i = 1$ then it computes $\sigma_i \leftarrow \mathsf{HTS.Forward}(\mathsf{usk}_i, 0, \sigma_{i-1})$.
   - If $\mathbf{x}_i = 0$ then it computes $\sigma_i \leftarrow \mathsf{HTS.Forward}(\mathsf{usk}_{n+i}, 0, \sigma_{i-1})$.
5. Alice sends (s, σ_n) to Bob.
6. Bob uses $G(s)$ to compute the setup $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk})$ and the user keys $\{(\mathsf{upk}_i, \mathsf{usk}_i)\}_{i=1}^{2n}$.
7. Bob initializes $z := 1$ and does the following for every $i \in [n]$ with $\mathbf{y}_i = 1$.
   - It computes $\sigma_{n+i} \leftarrow \mathsf{HTS.Forward}(\mathsf{usk}, 0, \sigma_n)$.
   - If $\mathsf{HTS.Open}(\mathsf{msk}, 0, \sigma_{n+i}) = \bot$ then it sets $z := 0$.
8. Bob sends z to Alice and both parties output z.

Correctness of the protocol follows from the security of PRG and the (t,t)-correctness of HTS. In more detail, since t is a non-constant polynomial in λ and $\lambda_1 = \sqrt{t(\lambda)}$, it follows from the security of the PRG that the correctness of HTS holds with overwhelming probability when $G(s)$ is used as the random tape for computing the setup and the user secret keys. Alice computes σ_n by forwarding σ_0 through n distinct forwarders, with public keys $(\mathsf{upk}'_1, \ldots, \mathsf{upk}'_n)$ where for each $i \in [n]$, $\mathsf{upk}'_i = \mathsf{upk}_i$ if $x_i = 1$ and $\mathsf{upk}'_i = \mathsf{upk}_{n+i}$ otherwise. Bob checks if $x_i = y_i = 1$ (in which case the inputs are not disjoint) by attempting to trace the signature σ_{n+i} obtained by forwarding σ_n

through the user with public key upk_i. Observe that if $x_i = 1$ then $\mathsf{upk}'_i = \mathsf{upk}_i$ which implies that σ_{n+i} has been forwarded through $n = t(\lambda) - 1$ distinct users and so $\mathsf{HTS.Open}(\mathsf{msk}, 0, \sigma_{n+i}) = \perp$ with overwhelming probability due to the correctness of HTS. Conversely, if $x_i = 0$ then σ_{n+i} has been forwarded through $n + 1 = t(\lambda)$ distinct users in which case $\mathsf{HTS.Open}(\mathsf{msk}, 0, \sigma_{n+i})$ outputs upk_0 with overwhelming probability. Thus, it follows that the parties output $\mathsf{DIS}_n(x, y)$ with all but negligible probability at the end of the protocol.

Next, observe that the parties communicate $|\sigma_n| + \lambda_1 + 1$ bits during the protocol. Since HTS is succinct and $\lambda_1 \in \mathsf{o}(t(\lambda))$ the expected communication complexity of the protocol for all inputs $x, y \in \{0,1\}^n$ is $\mathsf{o}(t(\lambda))$. However, for large enough λ, this is a contradiction to the fact that for any $\epsilon < \frac{1}{2}$, $C_\epsilon(\mathsf{DIS}_n) \in \Omega(n)$ [KS92]. This implies that HTS cannot be (t,t)-correct. $\square$

Claim. *If there exists a t-source anonymous HTS scheme then one-way functions exist.*

Proof Sketch. Let HTS be a t-source anonymous HTS scheme. Consider a function f such that $f(1^\lambda, x)$ runs $(\mathsf{mpk}, \mathsf{msk}, \mathsf{rsk}) \leftarrow \mathsf{HTS.Setup}(1^\lambda; x)$ and outputs $y = (\mathsf{mpk}, \mathsf{msk})$. It is easy to see that f is one-way. Specifically, if f is not one-way then an adversary for the $\mathsf{ExpSrcAnon}^{\mathsf{open}}_{\mathsf{HTS},\mathcal{A}}$ experiment can invert f to compute x and thus rsk. It can then break source anonymity by using rsk to locally compute t user key-pairs and forward the challenge signature through these users to de-anonymize the source. However, since HTS is t-source anonymous, it follows that f is a one-way function. $\square$

This concludes the proof of the theorem. $\blacksquare$

We note that the proof of Theorem 3 can be extended to argue the impossibility of succinct t-secure HTS schemes, even if a larger correctness error of up to $1/2$ is allowed, since the lower bound in [KS92] requires $\epsilon < 1/2$. Similarly, it can be shown that t-secure HTS schemes that satisfy a weaker form of succinctness, where the signature size is sublinear in the number of forwards only in expectation, are also impossible.

6 Succinct HTS Scheme

We present our construction of a succinct HTS scheme in Section 6.3. Before describing our construction, we first introduce two key building blocks: a streaming algorithm for counting distinct elements in Section 6.1 and Puncturable NIZKs in Section 6.2.

6.1 Streaming Algorithm

In this section, we define the syntax and properties of streaming algorithms for counting distinct elements. We then present the streaming algorithm of Bar-Yossef et al. [BYJK+02] and show that it satisfies the required properties. We refer the reader to Section 2.6 for background on streaming algorithms and a detailed discussion of how we address the challenges of using them in an adversarial setting.

We formalize the syntax and properties of streaming algorithms for counting distinct elements below. Our HTS construction requires three properties from streaming algorithms: correctness, insertion robustness and deletion robustness. Correctness ensures that an adversary cannot affect the algorithm's estimate by reordering or repeating uniformly random inputs. Insertion and deletion robustness, on the other hand, require that adversarial insertions do not decrease the estimate and deletions do not increase it, even when the inputs are not uniformly random. Note that while these robustness properties are fairly natural, they do not follow from correctness: since correctness is defined only for uniformly random inputs, it does not rule out the possibility that adversarial insertions (or deletions) could decrease (or increase) the estimate in other cases.

Definition 12 (Streaming Algorithm For Distinct Elements)**.** Let ℓ_{uniq} be an integer-valued polynomial. A streaming algorithm for distinct elements $\mathsf{Uniq} = (\mathsf{Update}, \mathsf{Query})$ is a tuple of algorithms with the following syntax.

- $\mathsf{Update}(\epsilon, \mathsf{st}, x) =: \mathsf{st}'$ is a deterministic algorithm that takes an error bound ϵ, the current state st and a value $x \in \{0,1\}^{\ell_{\mathsf{uniq}}(\lambda)}$ as input and outputs the next state st'. It runs in time $\mathsf{poly}(\lambda, 1/\epsilon)$. $\mathsf{st} = \bot$ denotes the initial state before any input is processed.
- $\mathsf{Query}(\epsilon, \mathsf{st}) =: q$ is a deterministic algorithm that takes the error bound ϵ and a state st as input and outputs the number of distinct elements q in the input stream used to compute st. It runs in time $\mathsf{poly}(\lambda, 1/\epsilon)$.

We require the streaming algorithm to satisfy the following properties.

- **Correctness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all adversaries $\mathcal{A}$, $\lambda \in \mathbb{N}$, $\epsilon \le \frac{1}{2}$ and $L \in \mathsf{poly}(\lambda)$, we have

$$\Pr\left[\begin{array}{c} \{x_i\}_{i=1}^L = \{y_i\}_{i=1}^\ell \\ \wedge \\ |\mathsf{Query}(\epsilon, \mathsf{st}_\ell) - L| \ge \epsilon L \end{array} : \begin{array}{r} x_1, \ldots, x_L \leftarrow \{0,1\}^{\ell_{\mathsf{uniq}}(\lambda)} \\ y_1, \ldots, y_\ell \leftarrow \mathcal{A}(\{x_i\}_{i=1}^L) \\ \mathsf{st}_0 := \bot \\ \mathsf{st}_i := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}, y_i),\ \forall i \in [\ell] \end{array}\right] \le \mathsf{negl}(\lambda).$$

- **Insertion Robustness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all adversaries $\mathcal{A}$, $\lambda \in \mathbb{N}$, all $\epsilon \le \frac{1}{2}$, all $L \in \mathsf{poly}(\lambda)$, and all $x_1, \ldots, x_L \in \{0,1\}^{\ell_{\mathsf{uniq}}(\lambda)}$, we have

$$\Pr\left[\begin{array}{c} \{x_i\}_{i=1}^L \subseteq \{y_i\}_{i=1}^\ell \\ \wedge \\ \mathsf{Query}(\epsilon, \mathsf{st}_\ell^x) \ge \mathsf{Query}(\epsilon, \mathsf{st}_L^y) \end{array} : \begin{array}{r} y_1, \ldots, y_\ell \leftarrow \mathcal{A}(\{x_i\}_{i=1}^L) \\ \mathsf{st}_i^x := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}^x, x_i),\ \forall i \in [L], \\ \mathsf{st}_i^y := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}^y, y_i),\ \forall i \in [\ell] \end{array}\right] \quad \le \quad \mathsf{negl}(\lambda),$$

where $\mathsf{st}_0^x := \bot$ and $\mathsf{st}_0^y := \bot$.

- **Deletion Robustness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all adversaries $\mathcal{A}$, $\lambda \in \mathbb{N}$, $\epsilon \le \frac{1}{2}$ and $L \in \mathsf{poly}(\lambda)$, and all $x_1, \ldots, x_L \in \{0,1\}^{\ell_{\mathsf{uniq}}(\lambda)}$, we have

$$\Pr\left[\begin{array}{c} \{x_i\}_{i=1}^L \supseteq \{y_i\}_{i=1}^\ell \\ \wedge \\ \mathsf{Query}(\epsilon, \mathsf{st}_\ell^x) \le \mathsf{Query}(\epsilon, \mathsf{st}_L^y) \end{array} : \begin{array}{r} y_1, \ldots, y_\ell \leftarrow \mathcal{A}(\{x_i\}_{i=1}^L) \\ \mathsf{st}_i^x := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}^x, x_i),\ \forall i \in [L], \\ \mathsf{st}_i^y := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}^y, y_i),\ \forall i \in [\ell] \end{array}\right] \quad \le \quad \mathsf{negl}(\lambda),$$

where $\mathsf{st}_0^x := \bot$ and $\mathsf{st}_0^y := \bot$.

- **Succinctness:** For all polynomials p_1, there exists a polynomial p_2 and a sublinear function $g(x) \in \mathsf{o}(x)$ such that for all $\lambda, L \in \mathbb{N}$ and $\epsilon \le 1/2$, where $L < p_1(\lambda)$, the size of the state st obtained upon processing L values with error bound ϵ is at most $p_2(\lambda \cdot 1/\epsilon) \cdot g(L)$.

Streaming Algorithm for Distinct Elements

Public parameters. Let $\ell_{\mathsf{uniq}} \in \Omega(\lambda^2)$ denote the length of each input and let $\ell_{\mathsf{sub}}(\lambda) = \ell_{\mathsf{uniq}}(\lambda)/\lambda$.

```
Update($\epsilon$, st, $x$)
1 : parse $x = (x_1, \ldots, x_\lambda)$
      where each $x_i \in [0, 2^{\ell_{\mathsf{sub}}(\lambda)}]$
2 : $\gamma := \lceil 96/\epsilon^2 \rceil$
3 : if st $\neq \perp$
4 :   parse st $= (\mathcal{M}_1, \ldots, \mathcal{M}_\lambda)$
5 :   else $\forall i \in [\lambda],\ \mathcal{M}_i := \emptyset$
6 : for $i \in [\lambda]$
7 :   $x_i^* = \max(\mathcal{M}_i)$
8 :   if $|\mathcal{M}_i| < \gamma$
9 :     append $x_i$ to $\mathcal{M}_i$
10 :  elseif $x_i < x_i^*$
11 :    replace $x_i^*$ with $x_i$ in $\mathcal{M}_i$
12 : return st$' := (\mathcal{M}_1, \ldots, \mathcal{M}_\lambda)$
```

```
Query($\epsilon$, st)
1 : if st $= \perp$ then return $0$
2 : parse st $= (\mathcal{M}_1, \ldots, \mathcal{M}_\lambda)$
3 : $\gamma := \lceil 96/\epsilon^2 \rceil$
4 : $\forall i \in [\lambda],\ q_i := \frac{\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}}{\max(\mathcal{M}_i)}$
5 : return $q :=$ median$(q_1, \ldots, q_\lambda)$
```

Figure 6: Streaming algorithm for distinct elements [BYJK+02].

Construction. The streaming algorithm of Bar-Yossef et al. [BYJK+02] is described in Figure 6. Intuitively, the streaming algorithm works by keeping track of the γ most minimum values among all its inputs, each of which are uniformly random. It then uses the γ-th most minimum value to estimate the number of distinct elements in the input stream. However, since a single estimate only has a constant probability of being within the required range, the algorithm is repeated λ times to amplify the probability.

Theorem 4. *For all polynomials $\ell_{\mathsf{uniq}} \in \Omega(\lambda^2)$, the algorithm described in Figure 6 is a streaming algorithm for distinct elements (Definition 12).*

Proof. We first show that when the input stream consists of L uniformly random elements, the algorithm's output is, with overwhelming probability, at most ϵL away from L. The correctness of the algorithm, as required by Definition 12, follows from the fact that the algorithm's output depends only on the set of distinct elements in the input stream and is independent of the order in which the elements are processed. Specifically, the algorithm parses each input as a tuple of λ integers and maintains the γ smallest values at each index separately, across all inputs. Since the algorithm's output is determined solely by the $\gamma\lambda$ smallest values it maintains, the output remains unchanged even if elements in the input stream are repeated or reordered.

Insertion robustness follows from the fact that the algorithm's output is inversely proportional to the maximum value among the $\gamma\lambda$ smallest values it maintains. In particular, inserting more elements into the stream cannot increase this maximum value, which means that the algorithm's output cannot decrease by adding new elements into the stream. Similarly, removing elements from the stream cannot decrease this maximum value, which in turn implies that the algorithm's output cannot increase by deleting elements, thereby ensuring deletion robustness.

Claim. *For all polynomials $\ell_{\mathsf{uniq}} \in \Omega(\lambda^2)$ and error bounds $\epsilon \leq 1/2$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ and $L \in \mathsf{poly}(\lambda)$, the algorithm described in Figure 6 satisfies*

$$\Pr\left[|\mathsf{Query}(\epsilon, \mathsf{st}_L) - L| \geq \epsilon L \ : \ \begin{array}{r} x_1, \ldots, x_L \leftarrow \{0,1\}^{\ell_{\mathsf{uniq}}(\lambda)} \\ \mathsf{st}_0 := \perp \\ \mathsf{st}_i := \mathsf{Update}(\epsilon, \mathsf{st}_{i-1}, x_i), \ \forall i \in [L] \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Proof. The proof follows the same approach as [BYJK+02, Theorem 1]. Observe that the algorithm outputs the median of $q_1, \ldots, q_\lambda$. We bound the probability that the median is more than ϵL far from L by bounding the probability that $|q_j - L| \geq \epsilon L$, for each $j \in [\lambda]$. Let $x_{i,j}$ denote the j-th $\ell_{\mathsf{sub}}(\lambda)$ length substring in x_i where $i \in [L]$. Consider any arbitrary $j \in [\lambda]$. We have $q_j = \gamma 2^{\ell_{\mathsf{sub}}(\lambda)} / \max(\mathcal{M}_j)$ where $\mathcal{M}_j$ contains the γ smallest values in $\{x_{i,j}\}_{i=1}^L$.

We first consider the case when $q_j \geq (1+\epsilon)L$ which in turn implies that at least γ values in the set $\{x_{i,j}\}_{i=1}^L$ are lesser than or equal to

$$\frac{\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}}{(1+\epsilon)L} \leq \left(1 - \frac{\epsilon}{2}\right) \frac{\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}}{L}$$

where the second inequality follows from the fact that $\epsilon \leq 1$. For every $i \in [L]$, let X_i be the indicator random variable for the event $x_{i,j} \leq \left(1 - \frac{\epsilon}{2}\right) \frac{\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}}{L}$. Since $x_{i,j}$ is sampled uniformly at random from $[2^{\ell_{\mathsf{sub}}(\lambda)}]$, for each $i \in [L]$ we have

$$\mathbb{E}[X_i] \leq \left(1 - \frac{\epsilon}{2}\right) \frac{\gamma}{L} + \frac{1}{2^{\ell_{\mathsf{sub}}(\lambda)}} < \left(1 - \frac{\epsilon}{4}\right) \frac{\gamma}{L}$$

where the second inequality follows from the fact that for large enough λ, $2^{\ell_{\mathsf{sub}}(\lambda)} > (4L)/(\epsilon\gamma)$ since $L \in \mathsf{poly}(\lambda)$. Additionally,

$$\mathbb{V}[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 = \mathbb{E}[X_i] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i] < \left(1 - \frac{\epsilon}{4}\right) \frac{\gamma}{L}$$

where the second equality follows from the fact that X_i is a Bernoulli random variable. Let $X = \sum_{i=1}^L X_i$. From linearity of expectations we have $\mathbb{E}[X] < \left(1 - \frac{\epsilon}{4}\right)\gamma$ and $\mathbb{V}[X] < \left(1 - \frac{\epsilon}{4}\right)\gamma$. From Chebyshev's inequality we have

$$\Pr[X \geq \gamma] \leq \Pr\left[|X - \mathbb{E}[X]| > \frac{\epsilon\gamma}{4}\right] \leq \frac{16\mathbb{V}[X]}{\epsilon^2\gamma^2} < \frac{16(1-\epsilon/4)}{\epsilon^2\gamma} \leq \frac{16}{\epsilon^2\gamma} \leq \frac{1}{6}$$

where the last inequality follows because $\gamma = \lceil 96/\epsilon^2 \rceil$. Thus, the probability that at least γ values in the set $\{x_{i,j}\}_{i=1}^L$ are lesser than or equal to $\gamma \cdot 2^{\ell_{\mathsf{sub}}(\lambda)} / ((1+\epsilon)L)$ is at most 1/6, which implies that $\Pr[q_j \geq (1+\epsilon)L] < 1/6$.

We next consider the case when $q_j \leq (1-\epsilon)L$. This occurs when at most γ values in the set $\{x_{i,j}\}_{i=1}^L$ are lesser than or equal to $\gamma \cdot 2^{\ell_{\mathsf{sub}}(\lambda)} / ((1-\epsilon)L)$. For each $i \in [L]$, let X_i be the indicator random variable for the event $x_{i,j} \leq \frac{\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}}{(1-\epsilon)L}$. For every $i \in [L]$ we have

$$\mathbb{E}[X_i] \leq \frac{\gamma}{(1-\epsilon)L} \leq \frac{2\gamma}{L}$$

since $\epsilon \leq 1/2$, and

$$\mathbb{E}[X_i] \geq \frac{\gamma}{(1-\epsilon)L} - \frac{1}{2^{\ell_{\mathsf{sub}}(\lambda)}} \geq \frac{(1+\epsilon)\gamma}{L} - \frac{1}{2^{\ell_{\mathsf{sub}}(\lambda)}} \geq \left(1+\frac{\epsilon}{2}\right)\frac{\gamma}{L}.$$

As in the previous case, we have $\mathbb{V}[X_i] \leq \mathbb{E}[X_i]$ which implies that $\mathbb{V}[X_i] \leq (2\gamma)/L$. Let $X = \sum_{i=1}^{L} X_i$. From linearity of expectations we have $\mathbb{E}[X] \leq \left(1+\frac{\epsilon}{2}\right)\gamma$ and $\mathbb{V}[X] \leq 2\gamma$. Thus, from Chebyshev's inequality we have

$$\Pr[X \leq \gamma] \leq \Pr\left[|X - \mathbb{E}[X]| \leq \frac{\epsilon\gamma}{2}\right] \leq \frac{\mathbb{V}[X]}{((\epsilon\gamma)/2)^2} \leq \frac{1}{12} < \frac{1}{6}.$$

Thus, the probability that at most γ values in the set $\{x_{i,j}\}_{i=1}^{L}$ are lesser than or equal to $\gamma \cdot 2^{\ell_{\mathsf{sub}}(\lambda)}/\left((1-\epsilon)L\right)$ is at most $1/6$ which implies that $\Pr[q_j \leq (1-\epsilon)L] < 1/6$.

Thus, for every $j \in [\lambda]$, we have $\Pr[|q_j - L| \geq \epsilon L] < 1/6$. Moreover, since $q_1, \ldots, q_\lambda$ are independent, it follows from a Chernoff bound that

$$\Pr[|\mathsf{Query}(\mathsf{st}_L) - L| \geq \epsilon L] \leq \mathsf{negl}(\lambda).$$

□

Next, we argue that the algorithm has the correctness, insertion robustness and deletion robustness properties.

Let $\mathbf{x} = (x_i)_{i=1}^{L}$ and let $\mathbf{y} = (y_i)_{i=1}^{\ell}$ be the tuple output by an adversary $\mathcal{A}$ when run on input $\mathbf{x}$. Let q_x and q_y denote the algorithm's output for input streams $\mathbf{x}$ and $\mathbf{y}$ respectively. Let x_j^* and y_j^* be the γ-th minimum value in $\{x_{i,j}\}_{i=1}^{L}$ and $\{y_{i,j}\}_{i=1}^{\ell}$ respectively, for every $j \in [\lambda]$.

Observe that q_x and q_y are completely determined by $(x_1^*, \ldots, x_\lambda^*)$ and $(y_1^*, \ldots, y_\lambda^*)$ respectively. Thus, if $\mathbf{x}$ and $\mathbf{y}$ have the same set of distinct elements, in other words when $\{x_i\}_{i=1}^{L} = \{y_i\}_{i=1}^{\ell}$, then $x_j^* = y_j^*$ for every $j \in [\lambda]$. This implies that $q_x = q_y$. However, when each element of $\mathbf{x}$ is sampled uniformly at random, it follows from the previous claim that

$$\Pr[|q_y - L| \geq \epsilon \cdot L] = \Pr[|q_x - L| \geq \epsilon \cdot L] \leq \mathsf{negl}(\lambda).$$

Thus, the algorithm satisfies the correctness property.

Conversely, when $\{x_i\}_{i=1}^{L} \subseteq \{y_i\}_{i=1}^{\ell}$, we have $x_j^* \geq y_j^*$ for every $j \in [\lambda]$. This implies that $q_x \leq q_y$, since q_x and q_y are computed as the median value of $\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}/y_j^*$ and $\gamma 2^{\ell_{\mathsf{sub}}(\lambda)}/x_j^*$ across all $j \in [\lambda]$ respectively. Thus, $\Pr[q_x \leq q_y] = 1$ and it follows that the algorithm has insertion robustness.

Similarly, when $\{x_i\}_{i=1}^{L} \supseteq \{y_i\}_{i=1}^{\ell}$, we have $x_j^* \leq y_j^*$ for every $j \in [\lambda]$. This implies that $\Pr[q_x \geq q_y] = 1$ which in turn implies that the algorithm has deletion robustness.

We have thus shown that the algorithm is an ϵ-approximate streaming algorithm for distinct elements. We are left to show that it is succinct. Observe that the algorithm's state consists of λ sets, each of which contains at most γ integers, where each integer is $\ell_{\mathsf{sub}}(\lambda)$ bits in length. Thus, the algorithm's state size is $\Theta(\lambda \cdot \gamma \cdot \ell_{\mathsf{sub}}(\lambda)) = \Theta(\mathsf{poly}(\lambda)/\epsilon^2)$. ■

6.2 Puncturable NIZK

In this section we define and construct Puncturable NIZKs.

Informally, given a predicate ν, a Puncturable NIZK guarantees zero-knowledge for statements x satisfying $\nu(x) = 1$, and ensures knowledge soundness for statements x satisfying $\nu(x) = 0$. More precisely, it allows puncturing the trapdoor td on a predicate ν so that the resulting trapdoor td_ν can be used to simulate proofs on any statement x such that $\nu(x) = 1$. On the other hand, it should be infeasible for the adversary, even given the punctured trapdoor td_ν, to prove false statements x such that $\nu(x) = 0$. This notion can be seen as a generalization of ID-based NIZKs [BJPY18]. Looking ahead, we use the ability to puncture the simulation trapdoor to prove source anonymity of our HTS construction.

Definition 13 (Puncturable NIZK). Let $\mathcal{L}$ be an NP language with relation $\mathcal{R}$. A Puncturable Non-Interactive Zero Knowledge (PNIZK) argument of knowledge for $\mathcal{L}$ is a tuple of algorithms $\mathsf{PNIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Puncture}, \mathsf{SimProve})$ with the following semantics.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$ is a PPT algorithm that outputs a common reference string and a trapdoor td.

- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$ is a PPT algorithm that takes as input a common reference string crs, a statement $x \in \mathcal{L}$ and a witness w, and outputs a proof π.

- $\mathsf{Verify}(\mathsf{crs}, x, \pi) =: b$ is a polynomial time algorithm that takes as input a common reference string crs, a statement x and a proof π and outputs a bit $b \in \{0, 1\}$.

- $\mathsf{Puncture}(\mathsf{td}, \nu) \to \mathsf{td}_\nu$ is a PPT algorithm that takes the trapdoor td and an efficiently computable predicate ν and outputs a punctured trapdoor td_ν.

- $\mathsf{SimProve}(\mathsf{crs}, x, \mathsf{td}_\nu) \to \pi$ is a PPT algorithm that takes the crs, a trapdoor td_ν and a statement x as input and outputs a simulated proof π.

We require a PNIZK argument of knowledge to satisfy the following properties.

- **Completeness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and $(x, w) \in \mathcal{R}$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) \neq 1 : \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Puncturable Knowledge Soundness:** There exists an extractor $\mathcal{E}$, which is a PPT algorithm with the following properties. Firstly,
$$\left\{(\mathsf{crs}, \mathsf{td}) : (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)\right\}_\lambda \overset{c}{\approx} \left\{(\mathsf{crs}, \mathsf{td}) : (\mathsf{crs}, \mathsf{td}, \mathsf{st}) \leftarrow \mathcal{E}(1^\lambda)\right\}_\lambda.$$
Moreover, for all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have
$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge \\ \nu(x) = 0 \\ \wedge \\ (x, w) \notin \mathcal{R} \end{array} : \begin{array}{r} (\mathsf{crs}, \mathsf{td}, \mathsf{st}_e) \leftarrow \mathcal{E}(1^\lambda) \\ (\nu, \mathsf{st}_a) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \mathsf{td}_\nu \leftarrow \mathsf{Puncture}(\mathsf{td}, \nu) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{st}_a, \mathsf{td}_\nu) \\ w \leftarrow \mathcal{E}(\mathsf{st}_e, \nu, \mathsf{td}_\nu, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda),$$
where ν is an efficiently computable predicate.

- **Puncturable Zero Knowledge:** For all non-uniform polynomial time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have

$$\Pr\left[\begin{array}{c} (x,w) \in \mathcal{R} \\ \wedge \\ \nu(x) = 1 \\ \wedge \\ b = b' \end{array} : \begin{array}{r} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\nu, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \mathsf{td}_\nu \leftarrow \mathsf{Puncture}(\mathsf{td}, \nu) \\ (x, w, \mathsf{st}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{td}_\nu) \\ \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \\ \pi_1 \leftarrow \mathsf{SimProve}(\mathsf{crs}, x, \mathsf{td}_\nu) \\ b \leftarrow \{0,1\} \\ b' \leftarrow \mathcal{A}(\mathsf{st}, \pi_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where ν is an efficiently computable predicate.

Remark 1 (Multi-theorem ZK). While puncturable zero knowledge is defined for a single statement in Definition 4, the definition readily extends to a multi-theorem setting (for every statement x satisfying $\nu(x) = 1$) via a standard hybrid argument. This holds because the adversary receives the punctured trapdoor and can generate simulated proofs on its own.

Remark 2 (PNIZK implies NIZK). Note that when the predicate is the null predicate, that is false on every input, puncturable knowledge soundness is equivalent to the standard NIZK knowledge soundness. On the other hand, when the predicate is the "all" predicate, that is true on every input, puncturable zero knowledge is equivalent to standard NIZK zero knowledge.

Construction. We describe our construction of Puncturable NIZKs from policy-based signature schemes (Definition 6) and NIZKs (Definition 4) in Figure 7. Informally, given a relation $\mathcal{R}$ and a statement $x \in \mathcal{R}$, the Puncturable NIZK computes a NIZK proof under an extended relation $\mathcal{R}_{\mathsf{nzk}}$ that accepts either a valid witness for x under $\mathcal{R}$ or a valid PBS signature on x. The PBS signing key serves as the trapdoor td. Completeness of the Puncturable NIZK follows directly from the completeness of the underlying NIZK. Puncturable knowledge soundness follows from the fact that an adversary that receives td_ν—the PBS signing key constrained by a policy ν—cannot produce signatures on any x such that $\nu(x) = 0$. Consequently, any witness w extracted from the NIZK proof must satisfy $(x, w) \in \mathcal{R}$. Conversely, the punctured trapdoor td_ν does allow computing signatures on statements x satisfying $\nu(x) = 1$, which in turn allows producing valid proofs for such statements. The zero-knowledge property of the NIZK ensures that the proof reveals nothing about whether it was generated using the trapdoor or with a valid witness under $\mathcal{R}$.

Theorem 5. *For all* NP *relations $\mathcal{R}$, if* PBS *is a policy-based signature scheme (Definition 6) and* NIZK *is a NIZK argument of knowledge (Definition 4) for the relation $\mathcal{R}_{\mathsf{nzk}}$, then the scheme described in Figure 7 is a Puncturable NIZK scheme for $\mathcal{R}$.*

Proof. Completeness of the Puncturable NIZK follows directly from the completeness of the NIZK and we mainly focus on proving puncturable knowledge soundness and puncturable zero-knowledge.

Claim. *If* PBS *is unforgeable and* NIZK *is knowledge sound, then the scheme described in Figure 7 has puncturable knowledge soundness.*

Proof. There exists an extractor $\mathcal{E}_{\mathsf{nzk}} = (\mathsf{Setup}, \mathsf{Extract})$ for NIZK, from its knowledge soundness property. We will use $\mathcal{E}_{\mathsf{nzk}}$ to build an extractor $\mathcal{E}$ for the Puncturable NIZK scheme as follows.

Puncturable NIZK Scheme

Public parameters. The relation $\mathcal{R}$ for the Puncturable NIZK, a policy-based signature scheme PBS, and a NIZK argument of knowledge for the relation

$$\mathcal{R}_{\mathsf{nzk}} = \{(x, \mathsf{mpk}; w) \mid (x, w) \in \mathcal{R} \vee \mathsf{PBS.Verify}(\mathsf{mpk}, x) = 1\}.$$

```
Setup(1^λ)
1 : $\mathsf{crs}_{\mathsf{nzk}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
2 : $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PBS.Setup}(1^\lambda)$
3 : $\mathsf{crs} := (\mathsf{crs}_{\mathsf{nzk}}, \mathsf{mpk})$
4 : $\mathsf{td} := \mathsf{msk}$
5 : return $(\mathsf{crs}, \mathsf{td})$
```

```
Prove(crs, x, w)
1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{nzk}}, \mathsf{mpk})$
2 : $\pi \leftarrow \mathsf{NIZK.Prove}($
        $\mathsf{crs}_{\mathsf{nzk}}, (x, \mathsf{mpk}), w)$
3 : return $\pi$
```

```
Verify(crs, x, π)
1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{nzk}}, \mathsf{mpk})$
2 : $b := \mathsf{NIZK.Verify}($
        $\mathsf{crs}_{\mathsf{nzk}}, (x, \mathsf{mpk}), \pi)$
3 : return $b$
```

```
Puncture(td, ν)
1 : parse $\mathsf{td} = \mathsf{msk}$
2 : $\mathsf{td}_\nu = \mathsf{PBS.KeyGen}(\mathsf{msk}, \nu)$
3 : return $\mathsf{td}_\nu$
```

```
SimProve(crs, x, td_ν)
1 : parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{nzk}}, \mathsf{mpk})$
2 : $\sigma \leftarrow \mathsf{PBS.Sign}(\mathsf{td}_\nu, x)$
3 : $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{nzk}}, (x, \mathsf{mpk}), \sigma)$
4 : return $\pi$
```

Figure 7: Puncturable NIZK construction from policy-based signatures and NIZK arguments of knowledge.

$\mathcal{E}$ first runs $(\mathsf{crs}_{\mathsf{nzk}}, \mathsf{td}') \leftarrow \mathcal{E}_{\mathsf{nzk}}.\mathsf{Setup}(1^\lambda)$ and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{PBS.Setup}(1^\lambda)$, and outputs $\mathsf{crs} = (\mathsf{crs}_{\mathsf{nzk}}, \mathsf{mpk})$, $\mathsf{td} = \mathsf{msk}$, and $\mathsf{st} = \mathsf{td}'$. Then, upon being called with input $(\mathsf{st}, \nu, \mathsf{td}_\nu, x, \pi)$, it computes and outputs $w \leftarrow \mathcal{E}_{\mathsf{nzk}}.\mathsf{Extract}(\mathsf{td}', x, \pi)$.

$(\mathsf{crs}, \mathsf{td})$ as output by $\mathcal{E}$ is indistinguishable from $(\mathsf{crs}, \mathsf{td})$ computed using the Puncturable NIZK's Setup algorithm since from the NIZK's knowledge soundness property, $\mathsf{crs}_{\mathsf{nzk}}$ computed using NIZK.Setup is indistinguishable from $\mathsf{crs}_{\mathsf{nzk}}$ computed using $\mathcal{E}_{\mathsf{nzk}}$. We next use a hybrid argument to show that any polynomial sized adversary $\mathcal{A}$ wins the puncturable knowledge soundness experiment with at most negligible probability. Let the output of the experiment be defined as 1 if π verifies successfully, $\nu(x) = 0$ and $(x, w) \notin \mathcal{R}$, and be defined as 0 otherwise.

- Hyb_0: This is the output of the puncturable knowledge soundness experiment when run with $\mathcal{A}$.
- Hyb_1: This hybrid is identical to the previous hybrid, except that the experiment aborts and outputs $\perp$ if $(x, w) \notin \mathcal{R}_{\mathsf{nzk}}$.

  $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$ since the knowledge soundness property of NIZK implies that the witness w output by $\mathcal{E}_{\mathsf{nzk}}.\mathsf{Extract}$ is such that $(x, w) \in \mathcal{R}_{\mathsf{nzk}}$, except with negligible probability; which in turn implies that the experiment aborts in Hyb_1 with at most negligible probability.
- Hyb_2: This hybrid is identical to the previous hybrid, except that the experiment aborts and outputs $\perp$ if $(x, w) \notin \mathcal{R}$.

The only difference between Hyb_1 and Hyb_2 is that the experiment additionally aborts in Hyb_2 when $(x, w) \notin \mathcal{R}$. This means that when Hyb_1 is not equal to Hyb_2, then $(x, w) \in \mathcal{R}_{\mathsf{nzk}}$ but $(x, w) \notin \mathcal{R}$. It follows that w must then be a valid signature on x under mpk, from the definition of $\mathcal{R}_{\mathsf{nzk}}$. However, since $\nu(x) = 0$, it follows that $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$, where the indistinguishability between the hybrids reduces directly to the unforgeability of PBS.

Observe that the experiment never outputs 1 in Hyb_2 since it aborts and outputs $\perp$ when $(x, w) \notin \mathcal{R}$. Thus, the probability that $\mathsf{Hyb}_2 = 1$ is 0. Since $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_2$, it follows that the experiment outputs 1 in Hyb_0 with at most negligible probability, which in turn implies that $\mathcal{A}$ wins the puncturable knowledge soundness experiment with at most negligible probability. $\square$

Claim. *If* PBS *is correct and* NIZK *is zero-knowledge, then the scheme described in Figure 7 has puncturable zero-knowledge.*

Proof. Let the output of the puncturable zero-knowledge experiment be defined as 1 if $b' = b$ and be defined as 0 otherwise. We use a hybrid argument to show that for any polynomial sized adversary $\mathcal{A}$, the output of the experiment is 1 with a probability of at most $1/2 + \mathsf{negl}(\lambda)$.

- Hyb_0: This is the output of the puncturable zero-knowledge experiment when run with $\mathcal{A}$.
- Hyb_1: This hybrid is identical to the previous hybrid, except that the experiment aborts and outputs $\perp$ if σ computed in $\mathsf{SimProve}$ is such that $\mathsf{PBS.Verify}(\mathsf{mpk}, \sigma, x) \neq 1$.

  Since $\nu(x) = 1$, it follows directly from the correctness of PBS that $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$.
- Hyb_2: This hybrid is identical to the previous hybrid, except that $\mathsf{crs}_{\mathsf{nzk}}$ and the proof π_b send to $\mathcal{A}$ are simulated using the NIZK simulator.

  When $b = 1$ and the experiments do not abort, we have $((x, \mathsf{mpk}), \sigma) \in \mathcal{R}_{\mathsf{nzk}}$ since σ verifies successfully under mpk. It follows that when $(x, w) \in \mathcal{R}$, the witness used to compute π_b is valid, irrespective of if b is equal to 0 or 1. Thus, $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$ from the zero-knowledge property of NIZK.

Observe that the proof sent to $\mathcal{A}$ is independent of b in Hyb_2 since it is always simulated. It follows that $b' = b$ and $\mathsf{Hyb}_2 = 1$ with probability exactly $1/2$. Since $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_2$, $\mathcal{A}$ wins the puncturable zero-knowledge experiment with a probability of at most $1/2 + \mathsf{negl}(\lambda)$. $\square$

It follows from the above argument that the scheme described in Figure 7 is a Puncturable NIZK. $\blacksquare$

6.3 HTS Construction

In this section, we present our succinct HTS scheme. We first list the required primitives and then describe the scheme.

6.3.1 Required primitives

We require the following primitives for our succinct HTS construction.

- A digital signature scheme Sig (Definition 3).
- A group signature scheme GSig (Definition 5).
- A streaming algorithm Uniq for counting distinct elements with input length ℓ_{uniq} (Definition 12).
- A pseudorandom permutation PRP (Definition 2) and a pseudorandom function PRF (Definition 1) with output lengths $\ell_{\mathsf{uniq}}(\lambda)$.
- A pseudorandom permutation PRP (Definition 2) and a pseudorandom function PRF (Definition 1) with output lengths $\ell_{\mathsf{uniq}}(\lambda)$.
- A protocol Π_{reg} that realizes the functionality $\mathcal{F}_{\mathsf{reg}}^{\mathsf{Sig}}$ (Functionality 1).
- A Puncturable NIZK scheme PNIZK (Definition 13) for the relation $\mathcal{R}_{\mathsf{pzk}}$ defined as

$$\mathcal{R}_{\mathsf{pzk}} = \left\{ (\mathsf{st}, \mathsf{st}', \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m \ ; \ \mathsf{fk}, \mathsf{cert}) \ \middle| \ \begin{array}{l} \mathsf{Sig.Verify}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{fk}, \mathsf{cert}) \wedge \\ \mathsf{st}' = \mathsf{Uniq.Update}(\epsilon, \mathsf{st}, f) \end{array} \right\}$$

  where $f = \mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$.
- An IVC scheme IVC (Definition 9) for the set of compliance predicates Φ where each predicate ϕ_{par} in Φ is parameterized by $\mathsf{par} = (\mathsf{crs}_{\mathsf{pzk}}, \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m)$ and is of the form

$$\phi_{\mathsf{par}}(\mathsf{st}, \mathsf{st}', \pi_{\mathsf{fwd}}) = \begin{cases} 1 & \mathsf{st}' = \bot \wedge \mathsf{st} = \bot \\ 1 & \mathsf{PNIZK.Verify}\left(\mathsf{crs}_{\mathsf{pzk}}, x_{\mathsf{pzk}}, \pi_{\mathsf{fwd}}\right) = 1 \\ 0 & \text{otherwise} \end{cases}$$

  where $x_{\mathsf{pzk}} = \left(\mathsf{st}, \mathsf{st}', \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m\right)$.
- An extractable witness encryption WE (Definition 7) for the relation $\mathcal{R}_{\mathsf{we}}$ defined as

$$\mathcal{R}_{\mathsf{we}} = \left\{ \left(\begin{array}{c} t, \epsilon, \mathsf{crs}_{\mathsf{ivc}}, \mathsf{crs}_{\mathsf{pzk}}, \mathsf{pk}_{\mathsf{sig}}, \\ k_{\mathsf{msg}}, m \ ; \ \mathsf{st}, \pi \end{array} \right) \ \middle| \ \begin{array}{c} \mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq t \wedge \\ \mathsf{IVC.Verify}(\mathsf{crs}_{\mathsf{ivc}}, \phi_{\mathsf{par}}, \mathsf{st}, \pi) = 1 \end{array} \right\}$$

  where $\mathsf{par} = \left(\mathsf{crs}_{\mathsf{pzk}}, \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m\right)$.
- A NIZK argument of knowledge NIZK (Definition 4) for the relation $\mathcal{R}_{\mathsf{nzk}}$ defined as

$$\mathcal{R}_{\mathsf{nzk}} = \left\{ \left(\begin{array}{c} \mathsf{gpk}_{\mathsf{gs}}, t, \epsilon, \mathsf{crs}_{\mathsf{ivc}}, \mathsf{crs}_{\mathsf{pzk}}, \\ \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m \ ; \ \sigma_{\mathsf{gs}}, r_{\mathsf{we}} \end{array} \right) \ \middle| \ \begin{array}{c} \mathsf{GSig.Verify}(\mathsf{gpk}_{\mathsf{gs}}, m, \sigma_{\mathsf{gs}}) = 1 \wedge \\ \mathsf{ct}_{\mathsf{we}} := \mathsf{WE.Enc}\left(x_{\mathsf{we}}, \sigma_{\mathsf{gs}} \ ; \ r_{\mathsf{we}}\right) \end{array} \right\}$$

  where $x_{\mathsf{we}} = \left(t, \epsilon, \mathsf{crs}_{\mathsf{ivc}}, \mathsf{crs}_{\mathsf{pzk}}, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m\right)$.

Functionality $\mathcal{F}_{\text{reg}}^{\text{Sig}}$

The functionality is parameterized by a signature scheme Sig, and proceeds as follows with a user U and the registration server S_{reg}, where either might be corrupt by an adversary $\mathcal{S}$.

```
1: The functionality receives $\mathsf{sk}_{\mathsf{sig}}$ from $S_{\text{reg}}$ and $\mathsf{mpk}$ from U. It parses $\mathsf{pk}_{\mathsf{sig}}$ from $\mathsf{mpk}$ and sends abort to $S_{\text{reg}}$ and U if $\mathsf{sk}_{\mathsf{sig}}$ is not a valid signing key for $\mathsf{pk}_{\mathsf{sig}}$.
2: The functionality samples $\mathsf{fk} \leftarrow \{0,1\}^\lambda$ uniformly at random and computes $\mathsf{cert} \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{fk})$.
3: If U is corrupt, the functionality sends $(\mathsf{fk}, \mathsf{cert})$ to $\mathcal{S}$. In all cases, it receives status from $\mathcal{S}$.
4: If $\mathsf{status} = \mathsf{abort}$ it sends abort to $S_{\text{reg}}$ and U. Else, it sends $(\mathsf{fk}, \mathsf{cert})$ to U and $\top$ to $S_{\text{reg}}$.
```

Functionality 1: Registration functionality.

Succinct HTS Scheme

Public parameters. Threshold parameter $t \in \mathsf{poly}(\lambda)$, error parameter $\epsilon \in \mathbb{R}$, and the primitives listed in Section 6.3.1.

```
Setup$(1^\lambda)$
1 : $(\mathsf{gpk}_{\mathsf{gs}}, \mathsf{rsk}_{\mathsf{gs}}, \mathsf{osk}_{\mathsf{gs}}) \leftarrow \mathsf{GSig.Setup}(1^\lambda)$
2 : $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$
3 : $\mathsf{crs}_{\mathsf{nzk}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
4 : $(\mathsf{crs}_{\mathsf{pzk}}, \cdot) \leftarrow \mathsf{PNIZK.Setup}(1^\lambda)$
5 : $\mathsf{crs}_{\mathsf{ivc}} \leftarrow \mathsf{IVC.Setup}(1^\lambda)$
6 : $\mathsf{mpk} := (\mathsf{gpk}_{\mathsf{gs}}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{crs}_{\mathsf{nzk}}, \mathsf{crs}_{\mathsf{pzk}}, \mathsf{crs}_{\mathsf{ivc}})$
7 : $\mathsf{msk} := (\mathsf{mpk}, \mathsf{osk}_{\mathsf{gs}})$
8 : $\mathsf{rsk} := (\mathsf{mpk}, \mathsf{rsk}_{\mathsf{gs}}, \mathsf{sk}_{\mathsf{sig}})$
9 : return $(\mathsf{crs}, \mathsf{msk}, \mathsf{rsk})$
```

```
Register$\langle S_{\text{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{mpk})\rangle$
1 : $S_{\text{reg}}$ parses $\mathsf{rsk}_{\mathsf{gs}}$, $\mathsf{sk}_{\mathsf{sig}}$ from rsk
2 : U runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GSig.KeyGen}(1^\lambda)$
3 : $\langle \mathsf{pk}, \mathsf{usk}_{\mathsf{gs}}\rangle \leftarrow$
        $\mathsf{GSig.Register}\langle S_{\text{reg}}(\mathsf{rsk}_{\mathsf{gs}}), \mathsf{U}(\mathsf{pk}, \mathsf{sk})\rangle$
4 : $\langle \cdot, (\mathsf{fk}, \mathsf{cert})\rangle \leftarrow$
        $\Pi_{\text{reg}}\langle S_{\text{reg}}(\mathsf{rsk}), \mathsf{U}(\mathsf{mpk})\rangle$
5 : $\mathsf{upk} := \mathsf{pk}$
6 : $\mathsf{usk} := (\mathsf{mpk}, \mathsf{usk}_{\mathsf{gs}}, \mathsf{fk}, \mathsf{cert})$
7 : $S_{\text{reg}}$ outputs upk
8 : U outputs (upk, usk)
```

Figure 8: Setup and registration procedures for the succinct HTS scheme. The remaining algorithms are described in Figure 9.

6.3.2 Construction

We describe our construction of succinct HTS signatures in Figures 8 and 9. The scheme is parameterized by a threshold $t \in \mathsf{poly}(\lambda)$ and the streaming algorithm error parameter $\epsilon \in \mathbb{R}$ so that the resulting HTS thresholds are $t_{\text{anon}} = t/(1+\epsilon)$ and $t_{\text{trace}} = t/(1-\epsilon)$.

Informally, the setup algorithm generates the setup of the underlying primitives, provides the group signature opening key osk to the E2EE server, and provides the group signature registration key rsk along with a signing key $\mathsf{sk}_{\mathsf{sig}}$ (used to authenticate forwarding keys) to the registration server. During registration, the user receives its secret key usk for the group signature scheme, along with a forwarding key fk that is authenticated via a signature under $\mathsf{pk}_{\mathsf{sig}}$. A signature in this

Succinct HTS Scheme (continued)

```
Sign(usk, $m$)
1 : parse mpk, $\mathsf{usk_{gs}}$ from usk
2 : parse mpk =
        $(\mathsf{gpk_{gs}}, \mathsf{pk_{sig}}, \mathsf{crs_{nzk}}, \mathsf{crs_{pzk}}, \mathsf{crs_{ivc}})$
3 : $\sigma_{\mathsf{gs}} \leftarrow \mathsf{GSig.Sign}(\mathsf{usk_{gs}}, m)$
4 : $k_{\mathsf{msg}}, r_{\mathsf{we}} \leftarrow \{0,1\}^\lambda$
5 : $x_{\mathsf{we}} := (t, \epsilon, \mathsf{crs_{ivc}}, \mathsf{crs_{pzk}}, \mathsf{pk_{sig}}, k_{\mathsf{msg}}, m)$
6 : $\mathsf{ct_{we}} := \mathsf{WE.Enc}(x_{\mathsf{we}}, \sigma_{\mathsf{gs}} \; ; \; r_{\mathsf{we}})$
7 : $x_{\mathsf{nzk}} := \mathsf{gpk_{gs}} \| x_{\mathsf{we}}$
8 : $\pi_{\mathsf{nzk}} \leftarrow$
        $\mathsf{NIZK.Prove}(\mathsf{crs_{nzk}}, x_{\mathsf{nzk}}, (\sigma_{\mathsf{gs}}, r_{\mathsf{we}}))$
9 : $\sigma := (\mathsf{ct_{we}}, \pi_{\mathsf{nzk}}, k_{\mathsf{msg}}, \bot, \bot)$
10 : return $\sigma$
```

```
Forward(usk, $m, \sigma$)
1 : parse usk = $(\mathsf{mpk}, \mathsf{usk_{gs}}, \mathsf{fk}, \mathsf{cert})$
2 : parse $\sigma = (\mathsf{ct_{we}}, \pi_{\mathsf{nzk}}, k_{\mathsf{msg}}, \mathsf{st}, \pi)$
3 : if $\mathsf{Verify}(\mathsf{mpk}, m, \sigma) = 0$
4 :     return $\bot$
5 : $f := \mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$
6 : $\mathsf{st}' := \mathsf{Uniq.Update}(\epsilon, \mathsf{st}, f)$
7 : $x_{\mathsf{pzk}} := (\mathsf{st}, \mathsf{st}', \epsilon, \mathsf{pk_{sig}}, k_{\mathsf{msg}}, m)$
8 : $\pi_{\mathsf{fwd}} \leftarrow \mathsf{PNIZK.Prove}($
        $\mathsf{crs_{pzk}}, x_{\mathsf{pzk}}, (\mathsf{fk}, \mathsf{cert}))$
9 : $\mathsf{par} := (\mathsf{crs_{pzk}}, \epsilon, \mathsf{pk_{sig}}, k_{\mathsf{msg}}, m)$
10 : $\pi' \leftarrow \mathsf{IVC.Prove}($
        $\mathsf{crs_{ivc}}, \phi_{\mathsf{par}}, \mathsf{st}, \pi, \pi_{\mathsf{fwd}}, \mathsf{st}')$
11 : $\sigma' := (\mathsf{ct_{we}}, \pi_{\mathsf{nzk}}, k_{\mathsf{msg}}, \mathsf{st}', \pi')$
12 : return $\sigma'$
```

```
Verify(mpk, $m, \sigma$)
1 : parse mpk =
        $(\mathsf{gpk_{gs}}, \mathsf{pk_{sig}}, \mathsf{crs_{nzk}}, \mathsf{crs_{pzk}}, \mathsf{crs_{ivc}})$
2 : parse $\sigma = (\mathsf{ct_{we}}, \pi_{\mathsf{nzk}}, k_{\mathsf{msg}}, \mathsf{st}, \pi)$
3 : $x_{\mathsf{nzk}} := (\mathsf{gpk_{gs}}, t(\lambda), \epsilon, \mathsf{crs_{ivc}}, \mathsf{crs_{pzk}},$
        $\mathsf{pk_{sig}}, k_{\mathsf{msg}}, m)$
4 : if $\mathsf{NIZK.Verify}(\mathsf{crs_{nzk}}, x_{\mathsf{nzk}}, \pi_{\mathsf{nzk}}) = 0$
5 :     return $0$
6 : $\mathsf{par} := (\mathsf{crs_{pzk}}, \epsilon, \mathsf{pk_{sig}}, k_{\mathsf{msg}}, m)$
7 : return $\mathsf{IVC.Verify}(\mathsf{crs_{ivc}}, \phi_{\mathsf{par}}, \mathsf{st}, \pi)$
```

```
Open(msk, $m, \sigma$)
1 : parse msk = $(\mathsf{mpk}, \mathsf{osk_{gs}})$
2 : parse $\sigma = (\mathsf{ct_{we}}, \pi_{\mathsf{nzk}}, k_{\mathsf{msg}}, \mathsf{st}, \pi)$
3 : if $\mathsf{Verify}(\mathsf{mpk}, m, \sigma) = 0$ or
        $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) < t$
4 :     return $\bot$
5 : $\sigma_{\mathsf{gs}} := \mathsf{WE.Dec}((\mathsf{st}, \pi), \mathsf{ct_{we}})$
6 : $\mathsf{upk} := \mathsf{GSig.Open}(\mathsf{osk_{gs}}, m, \sigma_{\mathsf{gs}})$
7 : return upk
```

Figure 9: Signing, forwarding, verification, and opening algorithms for the succinct HTS scheme. Setup and registration are described in Figure 8.

scheme is of the form

$$\sigma = \left(\underbrace{\mathsf{ct_{we}}, \quad \pi_{\mathsf{nzk}},}_{\text{Encryption of source identity}} \quad \underbrace{k_{\mathsf{msg}}, \quad \mathsf{st}, \quad \pi}_{\text{Counting unique forwarders}} \right).$$

Here, $\mathsf{ct_{we}}$ is a witness encryption of a group signature σ_{gs} on the message m, π_{nzk} is a proof that $\mathsf{ct_{we}}$ encrypts a valid group signature under the required statement, and k_{msg} is a PRF key—

all generated by the source during signing. To forward the signature, the user updates both the streaming algorithm state st and the IVC proof π, which together help count the number of unique forwarders. Specifically, each forwarder computes its contribution $f := \mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$, inputs f to the streaming algorithm to compute the updated state st′, and proves that it computed st′ honestly, using k_{msg} and a valid (authenticated) forwarding key. Once t_{trace} users have forwarded the signature, the witness encryption $\mathsf{ct}_{\mathsf{we}}$ can be decrypted to recover the group signature σ_{gs} computed by the source. In particular, the E2EE server can then use the group signature opening key osk to deanonymize σ_{gs} and learn the source identity.

We briefly sketch the correctness and security of the scheme to build intuition for the construction; the formal statement and proof are provided in Theorem 11.

Correctness (Lemma 6): Correctness requires that a signature σ_ℓ that has been honestly signed and forwarded ℓ times satisfy several properties. First, σ_ℓ must verify successfully. This follows directly from the completeness and correctness of the underlying primitives. Second, when σ_ℓ can be opened, it only opens to the identity of the source. Since the construction deanonymizes the source by opening the group signature σ_{gs}, correctness of the group signature scheme ensures that σ_ℓ only opens to the source's identity. Finally, the signature σ_ℓ must fail to open when the number of unique forwarders $L < t_{\mathsf{anon}}$, and succeed when $L \geq t_{\mathsf{trace}}$. Since the inputs to the streaming algorithm are pseudorandom—being outputs of a PRP—the resulting estimate of the streaming algorithm is similar to that produced on a uniformly random input stream. From the correctness of the streaming algorithm, this estimate is at most ϵL away from L. In particular, when $L < t_{\mathsf{anon}}$, the estimate is at most $(1+\epsilon) \cdot t_{\mathsf{anon}} = t$ with overwhelming probability, causing $\mathsf{Open}(\mathsf{msk}, m, \sigma_\ell)$ to return $\perp$. Conversely, if $L \geq t_{\mathsf{trace}}$, then the estimate is at least $(1-\epsilon) \cdot t_{\mathsf{trace}} = t$ with overwhelming probability. Combined with the completeness of the IVC and the correctness of the witness encryption and group signature schemes, this implies that σ_ℓ opens to the identity of the source.

Source Anonymity Against Opener (Lemma 7): Source anonymity against the opener requires that, even after corrupting the E2EE server and fewer than t_{anon} users, the adversary cannot identify the source of a signature σ generated by an honest user. At a high level, the only components of the signature that depend on the source identity are π_{nzk} and $\mathsf{ct}_{\mathsf{we}}$. The zero-knowledge property of NIZK ensures that π_{nzk} preserves the anonymity of the source. As for $\mathsf{ct}_{\mathsf{we}}$, the adversary cannot compute a valid witness for decryption. This follows from the unforgeability of the signature scheme Sig, which ensures that the adversary obtains fewer than t_{anon} valid certificates, corresponding only to corrupt parties. Consequently, the knowledge soundness of PNIZK and the IVC imply that, in any signature computed by the adversary via forwards, the streaming algorithm state must consist of fewer than t_{anon} inputs. Since each input is derived via a PRP evaluation at a key sampled uniformly at random by the honest source, the deletion robustness and correctness properties of the streaming algorithm imply that the estimate in such a signature is less than t with overwhelming probability. Therefore, the adversary cannot compute a valid witness to decrypt $\mathsf{ct}_{\mathsf{we}}$, which by the security of the witness encryption scheme, implies that the underlying group signature σ_{gs} remains hidden—and with it, the source's identity.

Note that the adversary can query the HFwd oracle to forward signatures on non-challenge messages on behalf of honest parties. It is crucial that such queries do not allow the adversary to learn additional certificates, as our argument above relies on the adversary obtaining fewer than t_{anon} certificates. The zero-knowledge property of PNIZK ensures that these forwarded signatures do not reveal the certificates and forwarding keys of honest parties. This is precisely why we require *Puncturable* NIZKs: they ensure zero-knowledge for non-challenge messages, while still guaranteeing knowledge soundness on the challenge message. Finally, we note that an artifact of our proof strategy is that the adversary cannot adaptively choose the challenge message or adaptively corrupt honest parties. We discuss this in more detail in Remarks 3 and 4.

Source Anonymity Against Registration Server (Lemma 8): Source anonymity against the registration server requires that an adversary that corrupts the registration server and learns its secret key rsk cannot deanonymize the source of signatures generated by honest users. While such an adversary can register an arbitrary number of parties and always decrypt the witness encryption, the result of decryption is the anonymous group signature σ_{gs}, which it cannot open without access to the group signature opening key $\mathsf{osk}_{\mathsf{gs}}$.

Traceability and Unframeability (Lemma 9): Traceability requires that for any signature σ_0 output by an adversary, forwarding it sequentially through at least t_{trace} honest users must always deanonymize the signature to a corrupt user's identity. Observe that the Open algorithm initially checks if the streaming algorithm's estimate is greater than t, and if so, decrypts the witness encryption to obtain the group signature σ_{gs} and subsequently opens σ_{gs}. Knowledge soundness of the NIZK scheme guarantees that $\mathsf{ct}_{\mathsf{we}}$ decrypts to a valid group signature σ_{gs}. The traceability of the group signature scheme then ensures that σ_{gs} will open to a corrupt user's identity.

However, decryption of $\mathsf{ct}_{\mathsf{we}}$ is conditioned on the streaming algorithm's estimate being greater than t when the signature is forwarded through t_{trace} honest users. This occurs with overwhelming probability due to the zero-knowledge property of the Puncturable NIZK, the security of the PRF and the insertion robustness and correctness of the streaming algorithm. In more detail, the zero-knowledge property of the Puncturable NIZK ensures that the forwarding key fk of honest users is hidden from the adversary. The security of the PRF then implies that $\mathsf{PRF}(\mathsf{fk}, m)$ is pseudorandom to the adversary, and since PRP is a permutation (albeit not pseudorandom), the contribution $f = \mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$ input by honest users to the streaming algorithm is pseudorandom. Consequently, the insertion robustness and correctness properties guarantee that the streaming algorithm's estimate is greater than t with overwhelming probability, regardless of how the adversary computed σ_0 — the input stream used to compute the streaming algorithm state st_0 in σ_0 constitute the insertions in the stream by the adversary, which from the insertion robustness of the streaming algorithm, cannot lower the final estimate.

Unframeability follows by a similar argument: since the adversary now possesses the group signature registration key $\mathsf{rsk}_{\mathsf{gs}}$, the unframeability of the group signature scheme ensures that the signature σ_{gs} cannot be opened to the identity of any honest party.

Forwarder Anonymity (Lemma 10): Forwarder anonymity requires that a forwarded signature does not leak the forwarder's identity. To prove that the scheme is forwarder anonymous, we need to argue that for any message m of the adversary's choice, forwarding a signature on m with one party's secret key is indistinguishable from using another's. Intuitively, this follows from the zero-knowledge property of the Puncturable NIZK and the fact that forwarding a signature only involves evaluating a PRF on m using the forwarding key. Since this PRF evaluation on m is indistinguishable from a uniformly random value, it is independent of the forwarding key and thus does not leak the forwarder's identity.

Succinctness Only the streaming algorithm state and IVC are updated in each forward. Due to the efficiency of the IVC and streaming algorithm, the size of the signature grows at most poly-logarithmically in the length of the forwarding path.

We now proceed to formally prove that the scheme described in Figures 8 and 9 is a succinct HTS scheme.

Lemma 6 (Correctness)**.** *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-correct (Definition 10), where $t_{\mathsf{anon}} = t/(1+\epsilon)$, $t_{\mathsf{trace}} = t/(1-\epsilon)$, and $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. Consider any non-uniform polynomial time adversary $\mathcal{A}$. Let the output of the correctness experiment be 1 if $\mathcal{A}$ wins the correctness experiment, and be defined as 0 otherwise. We use a hybrid argument to show that the output of the correctness experiment when run with $\mathcal{A}$ is 0 with overwhelming probability.

- Hyb_0: This is the output of the correctness experiment when run with $\mathcal{A}$.
- Hyb_1: This is identical to the previous hybrid, except that when $\mathsf{Verify}(\mathsf{mpk}, m, \sigma_\ell) = 0$, the experiment outputs 0.

  The only difference between Hyb_0 and Hyb_1 is that $\mathsf{Hyb}_0 = 1$ if σ_ℓ fails to verify while $\mathsf{Hyb}_1 = 0$ in this case. Observe that verifying σ_ℓ involves verifying the NIZK proof π_{nzk} and the IVC proof π. The correctness of the group signature scheme and the fact that Π_{reg} realizes $\mathcal{F}_{\mathsf{reg}}^{\mathsf{Sig}}$ implies that $(\sigma_{\mathsf{gs}}, r_{\mathsf{we}})$ is a valid witness for the NIZK statement x_{nzk}. The completeness of the NIZK scheme then implies that π_{nzk} verifies successfully with overwhelming probability. For the IVC proof to verify successfully, the computation needs to be compliant, as defined by the IVC compliance predicate, which in turn requires each Puncturable NIZK proof to verify successfully. Since Π_{reg} realizes $\mathcal{F}_{\mathsf{reg}}^{\mathsf{Sig}}$ and the digital signature scheme is correct, each cert verifies successfully under $\mathsf{pk}_{\mathsf{sig}}$ which implies from the completeness of the Puncturable NIZK scheme that each proof π_{fwd} verifies successfully with overwhelming probability. This implies that the computation is compliant and the IVC proof verifies successfully with overwhelming probability from the completeness of the IVC scheme. It follows that σ_ℓ will verify successfully with overwhelming probability and thus, $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$.
- Hyb_2: This is identical to the previous hybrid, except that the experiment outputs 0 when $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq t$ but $\mathsf{Open}(\mathsf{msk}, m, \sigma_\ell)$ does not output the source identity upk_0, where st is the streaming algorithm state in σ_ℓ.

  We argue that $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$ due to the correctness of the witness encryption and the group signature schemes. Observe that if the IVC proof π does not verify successfully then both Hyb_1 and Hyb_2 are 0. Similarly, the hybrids are identical when $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) < t$. Thus, the only case in which Hyb_1 and Hyb_2 can differ is when $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq t$ and π verifies successfully. However, in this case, (st, π) is a valid witness for the witness encryption statement x_{we}, and so $\mathsf{WE.Dec}((\mathsf{st}, \pi), \mathsf{ct}_{\mathsf{we}})$ outputs a valid group signature σ_{gs} with overwhelming probability. The correctness of the group signature scheme then implies that $\mathsf{Open}(\mathsf{msk}, m, \sigma_\ell) = \mathsf{upk}_0$ in this case. Thus, it follows that $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$.

- Hyb_3: This is identical to the previous hybrid, except that the experiment outputs 0 if

$$|\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) - L| \geq \epsilon \cdot L$$

where st is the streaming algorithm state in σ_ℓ.

  We argue that the streaming algorithm's output is more than $\epsilon \cdot L$ away from L with at most negligible probability. Consider the distribution of $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}')$, where st' is a streaming algorithm state that is computed identical to st except that the PRP is replaced with a truly random permutation i.e., each input to $\mathsf{Uniq.Update}$ is an evaluation of the truly random permutation on $\mathsf{PRF}(\mathsf{fk}, m)$. It follows from the pseudorandomness of the PRP that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}')$ and $\mathsf{Uniq.Query}(\epsilon, \mathsf{st})$ are computationally indistinguishable. However, the correctness of the streaming algorithm implies that $|\mathsf{Uniq.Query}(\epsilon, \mathsf{st}') - L| \leq \epsilon \cdot L$ with overwhelming probability. Thus, it follows that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st})$ is at most $\epsilon \cdot L$ away from L with overwhelming probability, which in turn implies that $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$.

- Hyb_4: This hybrid is always 0.

  Observe that Hyb_3 and Hyb_4 are both 0 when $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) > t$ but σ_ℓ does not open to the identity of the source. Similarly, if $|\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) - L| \geq \epsilon \cdot L$, where st is the streaming algorithm state in σ_ℓ, then both Hyb_3 and Hyb_4 are 0. Thus, Hyb_3 and Hyb_4 only differ in the case when $|\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) - L| \leq \epsilon \cdot L$. Here, Hyb_3 is 1 when $L < t_{\mathsf{anon}}$ and $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq t$ or when $L \geq t_{\mathsf{trace}}$ and $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) < t$. However, in this case, when $L < t_{\mathsf{anon}}$ we have $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \leq (1+\epsilon)L < (1+\epsilon)t_{\mathsf{anon}} = t$, and when $L \geq t_{\mathsf{trace}}$, we have $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq (1-\epsilon)L \geq (1-\epsilon)t_{\mathsf{trace}} = t$. This implies that Hyb_3 is never 1 and thus $\mathsf{Hyb}_3 \overset{p}{\equiv} \mathsf{Hyb}_4$.

It follows from our argument above that $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_4$, which in turn implies that the output of the correctness experiment is 0 with overwhelming probability when run with any adversary $\mathcal{A}$. Thus, the given scheme is $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-correct. ■

Lemma 7 (Source Anonymity Against Opener). *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is t_{anon}-source anonymous against the opener (Definition 10), where $t_{\mathsf{anon}} = t/(1+\epsilon)$, and $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. Consider any non-uniform polynomial time adversary $\mathcal{A}$. We use a hybrid argument to show that for any polynomial length message m, $\mathcal{A}$ wins the source anonymity experiment for the opener (Figure 2) with at most negligible probability more than $1/2$. In what follows, we use *pre-challenge phase* to refer to steps 1–3 of the source anonymity experiment, until the challenge signature σ is computed, and use *post-challenge phase* to refer to steps 4–5, where the adversary takes the challenge signature σ as input and attempts to guess the challenge bit b.

- Hyb_0: This is the output of the source anonymity experiment for the opener, when run with $\mathcal{A}$ i.e., this hybrid is equal to $\mathsf{ExpSrcAnon}^{\mathsf{open}}_{\mathsf{HTS},\mathcal{A}}(1^\lambda, m)$.

- Hyb_1: This hybrid is identical to the previous hybrid, except that the oracles Reg and RegH now run $\mathcal{F}^{\mathsf{Sig}}_{\mathsf{reg}}$ in place of Π_{reg}, and Reg uses the protocol's simulator $\mathcal{S}_{\mathsf{reg}}$ to simulate $\mathcal{A}$'s view.

  In both Hyb_0 and Hyb_1, the adversary only learns the public key of the honest user when it invokes RegH since the registration server is honest. In case of Reg, the adversary's view is that of a corrupt user participating in Π_{reg}. Since Π_{reg} realizes $\mathcal{F}^{\mathsf{Sig}}_{\mathsf{reg}}$, the view of the adversary in this hybrid is indistinguishable to that in the previous hybrid, which implies that $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$.

- Hyb_2: This hybrid is identical to the previous hybrid, except that the experiment terminates and outputs 0 when $\mathcal{A}$ queries HFwd on the challenge message m, or if it invokes Reg for the t_{anon}-th time.

  The only difference between Hyb_1 and Hyb_2 is that the experiment terminates early in Hyb_2 when $\mathcal{A}$ queries HFwd on m or if it corrupts t_{anon} users. However, both hybrids are equal to 0 in this case. It follows that $\mathsf{Hyb}_1 \overset{\mathsf{p}}{\equiv} \mathsf{Hyb}_2$.

- Hyb_3: This hybrid is identical to the previous hybrid, except for the following changes.
  - Setup additionally computes $\mathsf{td}_m \leftarrow \mathsf{PNIZK.Puncture}(\mathsf{td}, \nu_m)$ where td is the trapdoor output by $\mathsf{PNIZK.Setup}$. Here, the predicate ν_m is such that $\nu_m(x_{\mathsf{pzk}}) = 0$ if and only if the message m' in x_{pzk} (parsed according to $\mathcal{R}_{\mathsf{pzk}}$) is equal to the challenge message m.
  - The HFwd oracle is modified so that for every query $(\mathsf{upk}, m', \sigma')$, it computes π_{fwd} as

    $$\pi_{\mathsf{fwd}} \leftarrow \mathsf{PNIZK.SimProve}(\mathsf{crs}_{\mathsf{pzk}}, x_{\mathsf{pzk}}, \mathsf{td}_m)$$

    when running $\mathsf{Forward}(\mathsf{usk}, m', \sigma')$.

  The only difference between Hyb_2 and Hyb_3 is that in Hyb_2, π_{fwd} is computed using the user's forwarding key fk and certificate cert when answering queries on HFwd, while in Hyb_3, π_{fwd} is simulated using td_m. However, observe that by definition of the predicate ν_m and the puncturable zero-knowledge property of the Puncturable NIZK, π_{fwd} computed in Hyb_2 is indistinguishable from π_{fwd} computed in Hyb_3, for all messages $m' \neq m$. Moreover, both Hyb_2 and Hyb_3 terminate and output 0 if $\mathcal{A}$ queries HFwd on m and thus Hyb_3 never requires simulating π_{fwd} for statements containing the challenge message m. Thus, $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$ follows from a straightforward reduction to the puncturable zero-knowledge property of PNIZK.

- Hyb_4: This hybrid is identical to the previous hybrid, except for the following modifications.
  - Before the pre-challenge phase is run, the experiment samples $t_{\mathsf{anon}} - 1$ PRF keys $\{\mathsf{fk}_i\}_i$ and for each key, computes a certificate $\mathsf{cert}_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{fk}_i)$. Let $\mathcal{K}_{\mathsf{pre}} = \{(\mathsf{fk}_i, \mathsf{cert}_i)\}_{i=1}^{t_{\mathsf{anon}}-1}$.
  - The oracle RegH no longer computes the certificate cert on the honest user's forwarding key fk within $\mathcal{F}_{\mathsf{reg}}^{\mathsf{Sig}}$.
  - The oracle Reg uses a $(\mathsf{fk}_i, \mathsf{cert}_i) \in \mathcal{K}_{\mathsf{pre}}$ within $\mathcal{F}_{\mathsf{reg}}^{\mathsf{Sig}}$, instead of computing it afresh. The used pair is removed from $\mathcal{K}_{\mathsf{pre}}$.

  The only difference between Hyb_3 and Hyb_4 is that in Hyb_3, the experiment samples a fresh PRF key fk_i and computes a certificate cert_i using $\mathsf{sk}_{\mathsf{sig}}$ for each invocation of Reg and RegH. On the other hand, in Hyb_4, certificates for forwarding keys of honest parties are never computed and the set of forwarding keys and certificates of corrupt parties are pre-computed and stored in $\mathcal{K}_{\mathsf{pre}}$, at the start of the experiment. Note that it suffices to pre-compute $t_{\mathsf{anon}} - 1$ pairs since the experiment aborts in both hybrids if $|\mathcal{I}| = t_{\mathsf{anon}}$. Moreover, since the proof π_{fwd} is simulated in Hyb_3 and Hyb_4, the certificate on the forwarding keys of honest users is no longer required. It thus follows that $\mathsf{Hyb}_3 \overset{\mathsf{p}}{\equiv} \mathsf{Hyb}_4$.

- Hyb_5: This hybrid is identical to the previous hybrid, except that $\mathsf{crs}_{\mathsf{pzk}}$ is computed using $\mathsf{PNIZK}.\mathcal{E}(1^\lambda) \to (\mathsf{crs}_{\mathsf{pzk}}, \mathsf{td}, \mathsf{st}_{\mathsf{pzk}})$, where $\mathsf{PNIZK}.\mathcal{E}$ is the extractor for PNIZK.

  $\mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5$ follows directly from the fact that $(\mathsf{crs}_{\mathsf{pzk}}, \mathsf{td})$ output by $\mathsf{PNIZK}.\mathcal{E}$ is indistinguishable from that generated using $\mathsf{PNIZK.Setup}$.

- Hyb_6: This hybrid is identical to the previous hybrid, except that the NIZK simulator $\mathcal{S}_{\mathsf{nzk}}$ is used to generate $\mathsf{crs}_{\mathsf{nzk}}$ in Setup and for simulating π_{nzk} when computing the challenge signature σ.

  The only difference between Hyb_5 and Hyb_6 is that the NIZK proof in the challenge signature is simulated. $\mathsf{Hyb}_5 \overset{c}{\approx} \mathsf{Hyb}_6$ follows immediately from a straightforward reduction to the zero-knowledge property of NIZK.

- Hyb_7: This hybrid is identical to the previous hybrid, except that $\mathsf{ct}_{\mathsf{we}}$ is computed as an encryption of $0^{|\sigma_{\mathsf{gs}}|}$ instead of σ_{gs}.

  Assume for the sake of contradiction that Hyb_6 is distinguishable from Hyb_7 with non-negligible probability i.e., the difference in the probability with which $\mathcal{A}$ wins the source anonymity game in Hyb_6 compared to Hyb_7 is non-negligible. We will use $\mathcal{A}$ to construct an adversary that breaks the deletion robustness property of the streaming algorithm with non-negligible probability. However, this contradicts the security of the streaming algorithm, which in turn implies that our assumption was wrong and that Hyb_6 and Hyb_7 are indeed indistinguishable.

  First, we prove the following claim, which informally states that if Hyb_6 and Hyb_7 are distinguishable with non-negligible probability, then there exists an adversary $\mathcal{A}_{\mathsf{int}}$ that takes $t_{\mathsf{anon}} - 1$ forwarding keys with their certificates as input, and outputs a streaming algorithm state st such that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}) \geq t$. Moreover, each value in the input stream used to compute st is of the form $\mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}_i, m))$ for some forwarding key fk_i with a valid certificate cert_i.

Claim. *If* Hyb_6 *and* Hyb_7 *are distinguishable with non-negligible probability, then there exists a non-uniform polynomial time adversary* $\mathcal{A}_{\mathsf{int}}$ *and a non-negligible function* $\delta(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and all polynomial length messages* m*,*

$$\Pr\left[\begin{array}{c}\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) \geq t \\ \wedge \\ \mathsf{PNIZK.Verify}\left(\mathsf{crs}_{\mathsf{pzk}}, x_{\mathsf{pzk}}^{(i)}, \pi_{\mathsf{fwd}}^{(i)}\right) = 1,\ \forall i \in [\ell]\end{array}\right] \geq \delta(\lambda)$$

where $\mathsf{st}_0 = \perp$, $x_{\mathsf{pzk}}^{(i)} := (\mathsf{st}_{i-1}, \mathsf{st}_i, \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m)$ *and the probability is over the following experiment.*

- *Generate* $(\mathsf{crs}_{\mathsf{pzk}}, \mathsf{td}, \mathsf{st}_{\mathsf{pzk}}) \leftarrow \mathsf{PNIZK}.\mathcal{E}(1^\lambda)$, $\mathsf{td}_m \leftarrow \mathsf{PNIZK.Puncture}(\mathsf{td}, \nu_m)$ *and* $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$.
- *Sample* $k_{\mathsf{msg}} \leftarrow \{0,1\}^\lambda$ *uniformly at random. Sample* $\mathsf{fk}_i \leftarrow \{0,1\}^\lambda$ *uniformly at random and compute* $\mathsf{cert}_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{fk}_i)$ *for each* $i \in [t_{\mathsf{anon}} - 1]$.
- *Run* $\left\{\mathsf{st}_i, \pi_{\mathsf{fwd}}^{(i)}\right\}_{i=1}^{\ell} \leftarrow \mathcal{A}_{\mathsf{int}}\left(\mathsf{crs}_{\mathsf{pzk}}, \mathsf{td}_m, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, \{\mathsf{fk}_i, \mathsf{cert}\}_i\right)$.

Proof. If Hyb_6 is distinguishable from Hyb_7 then the difference in the probabilities with which $\mathcal{A}$ wins in Hyb_6 and Hyb_7 is non-negligible. We will exploit this to construct the adversary $\mathcal{A}_{\mathsf{int}}$.

Observe that running the experiment in Hyb_6 and Hyb_7 with $\mathcal{A}$ requires the following, which we denote by $\mathsf{st}_{\mathsf{exp}}$: the public key $\mathsf{mpk} = (\mathsf{gpk}_{\mathsf{gs}}, \mathsf{pk}_{\mathsf{sig}}, \mathsf{crs}_{\mathsf{nzk}}, \mathsf{crs}_{\mathsf{pzk}}, \mathsf{crs}_{\mathsf{ivc}})$, the group signature scheme's registration key $\mathsf{rsk}_{\mathsf{gs}}$ and opening key $\mathsf{osk}_{\mathsf{gs}}$, the pre-computed set of forwarding keys and certificates $\mathcal{K}_{\mathsf{pre}}$, the NIZK simulation trapdoor $\mathsf{td}_{\mathsf{nzk}}$, and the punctured trapdoor td_m of the Puncturable NIZK. It will also be helpful to include the PRP key k_{msg} in $\mathsf{st}_{\mathsf{exp}}$, that is used in the challenge signature σ. Importantly, running the experiment does not require the registration server's signing key $\mathsf{sk}_{\mathsf{sig}}$ nor the certificates on the forwarding key of any of the honest users.

We now proceed to show that $\mathcal{A}_{\mathsf{int}}$ can indeed be constructed from $\mathcal{A}$. We begin by showing that $\mathcal{A}$ can be used to construct an adversary $\mathcal{A}_{\mathsf{we}}$ against the extractable security of WE. $\mathcal{A}_{\mathsf{we}}$ receives $\mathsf{st}_{\mathsf{exp}}$ as auxiliary input and runs the experiment in Hyb_7 with $\mathcal{A}$. When $\mathcal{A}$ outputs $(\mathsf{upk}_0, \mathsf{upk}_1)$, $\mathcal{A}_{\mathsf{we}}$ computes the challenge signature σ as in Hyb_7 except for the witness encryption component. Instead, it outputs x_{we} and a pair of messages $(\sigma_{\mathsf{gs}}, 0^{|\sigma_{\mathsf{gs}}|})$. Upon receiving the challenge witness encryption ciphertext $\mathsf{ct}_{\mathsf{we}}$, it appends $\mathsf{ct}_{\mathsf{we}}$ to σ and continues to run the experiment. $\mathcal{A}_{\mathsf{we}}$ outputs 1 if $\mathcal{A}$ wins the internal source anonymity experiment and outputs 0 otherwise. If $\mathsf{ct}_{\mathsf{we}}$ corresponds to an encryption of σ_{gs} then the view of $\mathcal{A}$ corresponds to that in Hyb_6, else if it's an encryption of $0^{|\sigma_{\mathsf{gs}}|}$, $\mathcal{A}$'s view corresponds to that in Hyb_7. Thus, $\mathcal{A}_{\mathsf{we}}$ distinguishes between witness encryptions of σ_{gs} and $0^{|\sigma_{\mathsf{gs}}|}$ with non-negligible probability. It follows from the extractable security of WE that there exists an extractor $\mathcal{E}_{\mathsf{we}}$ that outputs a statement x_{we} and a witness (st, π) for the relation $\mathcal{R}_{\mathsf{we}}$, with non-negligible probability.

Next, we use $\mathcal{E}_{\mathsf{we}}$ to construct an adversary $\mathcal{A}_{\mathsf{ivc}}$ against the knowledge soundness of IVC. $\mathcal{A}_{\mathsf{ivc}}$ takes $\mathsf{crs}_{\mathsf{ivc}}$ as input and $\mathsf{st}_{\mathsf{exp}}$ as auxiliary input. It updates the Puncturable NIZK CRS in $\mathsf{st}_{\mathsf{exp}}$ with $\mathsf{crs}_{\mathsf{ivc}}$ to obtain $\mathsf{st}'_{\mathsf{exp}}$ and runs $\mathcal{E}_{\mathsf{we}}$ with $\mathsf{st}'_{\mathsf{exp}}$ as auxiliary input. It obtains x_{we}, a streaming algorithm state st and an IVC proof π and outputs $(\phi_{\mathsf{par}}, \mathsf{st}, \pi)$, where par are computed from x_{we}. Since π verifies under $\mathsf{crs}_{\mathsf{ivc}}$ with non-negligible probability, it follows from the knowledge soundness of the IVC that there exists an extractor $\mathcal{E}_{\mathsf{ivc}}$ that outputs a valid computation trace $T = \left(\mathsf{st}_i, \pi^{(i)}_{\mathsf{fwd}}\right)_{i=1}^{\ell}$ under ϕ_{par}.

Finally, we use $\mathcal{E}_{\mathsf{ivc}}$ to construct $\mathcal{A}_{\mathsf{int}}$ as follows. $\mathcal{A}_{\mathsf{int}}$ takes $\mathsf{crs}_{\mathsf{pzk}}$, td_m, $\mathsf{pk}_{\mathsf{sig}}$, k_{msg}, and $\mathcal{K}_{\mathsf{pre}}$ as input and computes $(\mathsf{gpk}_{\mathsf{gs}}, \mathsf{rsk}_{\mathsf{gs}}, \mathsf{osk}_{\mathsf{gs}})$, $(\mathsf{crs}_{\mathsf{nzk}}, \mathsf{td}_{\mathsf{nzk}})$, and $\mathsf{crs}_{\mathsf{ivc}}$ to build $\mathsf{st}_{\mathsf{exp}}$. It then runs $\mathcal{E}_{\mathsf{ivc}}$ with $\mathsf{crs}_{\mathsf{ivc}}$ as input and $\mathsf{st}_{\mathsf{exp}}$ as auxiliary input to obtain and output $\left(\mathsf{st}_i, \pi^{(i)}_{\mathsf{fwd}}\right)_{i=1}^{\ell}$. We have $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) \geq t$ since $\mathcal{E}_{\mathsf{we}}$ outputs a witness (st_ℓ, π) under $\mathcal{R}_{\mathsf{we}}$. Moreover, since $\mathcal{E}_{\mathsf{ivc}}$ outputs a valid computation trace under ϕ_{par} with non-negligible probability, it follows that each $\pi^{(i)}_{\mathsf{fwd}}$ verifies successfully under the statement $x^{(i)}_{\mathsf{pzk}}$ with non-negligible probability. □

Next, we show that $\mathcal{A}_{\mathsf{int}}$ can be used to construct an adversary $\mathcal{A}_{\mathsf{uq}}$ against the deletion robustness property of the streaming algorithm, albeit on a pseudorandom input stream.

Claim. *If* Hyb_6 *and* Hyb_7 *are distinguishable with non-negligible probability, then there exists a non-uniform polynomial time stateful adversary* $\mathcal{A}_{\mathsf{uq}}$ *and a non-negligible function* $\delta(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c} \{x_i\}_{i=1}^{t_{\mathsf{anon}}-1} \supseteq \{y_i\}_{i=1}^{\ell} \\ \wedge \\ \mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) \geq t \end{array} : \begin{array}{r} x_1, \ldots, x_{t_{\mathsf{anon}}-1} \leftarrow \mathcal{A}_{\mathsf{uq}}(1^\lambda) \\ k_{\mathsf{msg}} \leftarrow \{0,1\}^\lambda \\ y_1, \ldots, y_\ell \leftarrow \mathcal{A}_{\mathsf{uq}}(k_{\mathsf{msg}}) \\ \mathsf{st}_0 := \perp \\ \mathsf{st}_i := \mathsf{Uniq.Update}(\epsilon, \mathsf{st}_{i-1}, \mathsf{PRP}(k_{\mathsf{msg}}, y_i)),\ \forall i \in [\ell] \end{array}\right] \geq \delta(\lambda).$$

Proof. We construct $\mathcal{A}_{\mathsf{uq}}$ using the adversary $\mathcal{A}_{\mathsf{int}}$ from the previous claim. For the sake of brevity, we say $\mathcal{A}_{\mathsf{int}}$ wins its experiment to denote the event when its output indeed provides a valid sequence of streaming algorithm states and Puncturable NIZK proofs such that the estimate on the last state is greater than t.

$\mathcal{A}_{\mathsf{uq}}$ proceeds as follows. It first computes $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}})$ using $\mathsf{Sig.Gen}$, samples $\mathsf{fk}_i \leftarrow \{0,1\}^\lambda$ uniformly at random and computes $\mathsf{cert}_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{\mathsf{sig}}, \mathsf{fk}_i)$ for each $i \in [t_{\mathsf{anon}} - 1]$. It then chooses an arbitrary message m and outputs $(x_1, \ldots, x_{t_{\mathsf{anon}}-1})$, where each $x_i = \mathsf{PRF}(\mathsf{fk}_i, m)$. Upon receiving k_{msg} as input, it runs the experiment described in the previous claim for $\mathcal{A}_{\mathsf{int}}$. If $\mathcal{A}_{\mathsf{int}}$ does not win its experiment, $\mathcal{A}_{\mathsf{uq}}$ outputs $(x_1, \ldots, x_{t_{\mathsf{anon}}-1})$. Else, it obtains $\left\{\mathsf{st}_i, \pi_{\mathsf{fwd}}^{(i)}\right\}_{i=1}^{\ell}$ from $\mathcal{A}_{\mathsf{int}}$ and outputs $(y_1, \ldots, y_\ell)$, where each $y_i = \mathsf{PRF}(\mathsf{fk}'_i, 0)$ and $(\mathsf{fk}'_i, \mathsf{cert}'_i)$ is the witness extracted by $\mathsf{PNIZK}.\mathcal{E}$ for the proof $\pi_{\mathsf{fwd}}^{(i)}$.

We will first use a hybrid argument to prove that $\{y_i\}_{i=1}^{\ell}$ output by $\mathcal{A}_{\mathsf{uq}}$ is indeed a subset of $\{x_i\}_{i=1}^{t_{\mathsf{anon}}-1}$ with all but negligible probability.

- $\mathsf{Hyb}_{7.0}$: In this hybrid, we run $\mathcal{A}_{\mathsf{uq}}$ as described in the claim. The hybrid is 0 if $\{y_i\}_{i=1}^{\ell} \not\subseteq \{x_i\}_{i=1}^{t_{\mathsf{anon}}-1}$, and is equal to 1 otherwise.
- $\mathsf{Hyb}_{7.1}$: This hybrid is identical to the previous hybrid, except that it is 1 whenever $\mathcal{A}_{\mathsf{int}}$ run by $\mathcal{A}_{\mathsf{uq}}$ does not win its experiment.

  In this case, $\mathcal{A}_{\mathsf{uq}}$ outputs $\{y_i\}_i = \{x_i\}_i$. It follows that $\mathsf{Hyb}_{7.0} \overset{\mathsf{p}}{=} \mathsf{Hyb}_{7.1}$.
- $\mathsf{Hyb}_{7.2}$: This hybrid is identical to the previous hybrid, except that it is 1 if $\mathsf{PNIZK}.\mathcal{E}$ does not output a valid witness for $\pi_{\mathsf{fwd}}^{(i)}$ under the statement $x_{\mathsf{pzk}}^{(i)} = (\mathsf{st}_{i-1}, \mathsf{st}_i, \epsilon, \mathsf{pk}_{\mathsf{sig}}, k_{\mathsf{msg}}, m)$ and relation $\mathcal{R}_{\mathsf{pzk}}$.

  Observe that $\mathsf{Hyb}_{7.1}$ and $\mathsf{Hyb}_{7.2}$ only differ in the case when $\mathcal{A}_{\mathsf{int}}$ wins its experiment. However, in this case, each $\pi_{\mathsf{fwd}}^{(i)}$ is a valid proof for the statement $x_{\mathsf{pzk}}^{(i)}$. Moreover, since $\mathsf{td}_m(x_{\mathsf{pzk}}^{(i)}) = 0$ for each $x_{\mathsf{pzk}}^{(i)}$, it follows from a standard hybrid argument, reducing to the puncturable knowledge soundness of PNIZK, that $\mathsf{Hyb}_{7.1} \overset{c}{\approx} \mathsf{Hyb}_{7.2}$.
- $\mathsf{Hyb}_{7.3}$: This hybrid is always equal to 1.

  Observe that the only case in which $\mathsf{Hyb}_{7.2}$ might not be equal to 1 is when $\{\mathsf{fk}'_i\}_{i=1}^{\ell} \not\subseteq \{\mathsf{fk}_i\}_{i=1}^{t_{\mathsf{anon}}-1}$. Let fk'_j be the element not in $\{\mathsf{fk}_i\}_{i=1}^{t_{\mathsf{anon}}-1}$. If this is the case, then $(\mathsf{fk}'_j, \mathsf{cert}'_j)$ extracted from $\pi_{\mathsf{fwd}}^{(j)}$ is such that cert'_j is a valid signature on fk'_j but a signature on fk'_j was never provided to $\mathcal{A}_{\mathsf{int}}$. Thus, this breaks the unforgeability of Sig. Since running $\mathcal{A}_{\mathsf{int}}$ does not require the signing key $\mathsf{sk}_{\mathsf{sig}}$, it follows from a straightforward reduction to unforgeability of Sig that $\mathsf{Hyb}_{7.3} \overset{c}{\approx} \mathsf{Hyb}_{7.2}$.

It follows from our argument that $\mathsf{Hyb}_{7.0} \overset{c}{\approx} \mathsf{Hyb}_{7.3}$, which implies that $\{y_i\}_{i=1}^{\ell}$ is indeed a subset of $\{x_i\}_{i=1}^{t_{\mathsf{anon}}-1}$ with overwhelming probability. Finally, observe that when $\mathcal{A}_{\mathsf{int}}$ wins its experiment, the output of $\mathcal{A}_{\mathsf{uq}}$ is such that the resulting streaming algorithm state st_ℓ provides an estimate greater than or equal to t, with overwhelming probability. This is because, the $\mathcal{A}_{\mathsf{int}}$ outputs st_ℓ such that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) \geq t$ and st_ℓ computed using $\mathcal{A}_{\mathsf{uq}}$'s output is identical to $\mathcal{A}_{\mathsf{int}}$'s output. The claim then follows immediately from the fact that when Hyb_6 and Hyb_7 are distinguishable with non-negligible probability, $\mathcal{A}_{\mathsf{int}}$ wins its experiment with non-negligible probability. □

We conclude by showing that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) < t$ with overwhelming probability due to the streaming algorithm's deletion robustness and correctness. Consequently, this implies that Hyb_6 is indistinguishable from Hyb_7.

Claim. $\mathsf{Hyb}_6 \overset{c}{\approx} \mathsf{Hyb}_7$.

Proof. Let $(x_1, \ldots, x_{t_{\mathsf{anon}}-1}) \leftarrow \mathcal{A}_{\mathsf{uq}}(1^\lambda)$ and let $(y_1, \ldots, y_\ell) \leftarrow \mathcal{A}_{\mathsf{uq}}(k_{\mathsf{msg}})$, where $\mathcal{A}_{\mathsf{uq}}$ is the adversary from the previous claim and $k_{\mathsf{msg}} \leftarrow \{0,1\}^\lambda$ is sampled uniformly at random. Let

$\mathsf{st}_0^x = \mathsf{st}_0^y = \perp$ and let

$$\mathsf{st}_i^x := \mathsf{Uniq.Update}(\epsilon, \mathsf{st}_{i-1}^x, \mathsf{PRP}(k_{\mathsf{msg}}, x_i)), \quad \forall i \in [t_{\mathsf{anon}} - 1]$$
$$\mathsf{st}_i^y := \mathsf{Uniq.Update}(\epsilon, \mathsf{st}_{i-1}^y, \mathsf{PRP}(k_{\mathsf{msg}}, y_i)), \quad \forall i \in [\ell].$$

That is, $\mathsf{st}_{t_{\mathsf{anon}}-1}^x$ and st_ℓ^y are the streaming algorithm states obtained on inputs $\{\mathsf{PRP}(k_{\mathsf{msg}}, x_i)\}_i$ and $\{\mathsf{PRP}(k_{\mathsf{msg}}, y_i)\}_i$ respectively. From the previous claim, we have $\{y_i\}_{i=1}^{\ell} \subseteq \{x_i\}_{i=1}^{t_{\mathsf{anon}}-1}$, which along with the deletion robustness property of the streaming algorithm, implies that

$$\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell^y) \leq \mathsf{Uniq.Query}(\epsilon, \mathsf{st}_{t_{\mathsf{anon}}-1}^x). \tag{1}$$

Next, observe that the security of PRP implies that the distribution of $\mathsf{st}_{t_{\mathsf{anon}}-1}^x$ is indistinguishable from the streaming algorithm state computed using $t_{\mathsf{anon}} - 1$ uniformly random inputs. In particular, since $\mathcal{A}_{\mathsf{uq}}$ outputs $\{x_i\}_i$ before receiving k_{msg} as input, $\mathsf{st}_{t_{\mathsf{anon}}-1}^x$ can be computed with only oracle access to the PRP. Thus, from the correctness of the streaming algorithm, the estimate on $\mathsf{st}_{t_{\mathsf{anon}}-1}^x$ is at most $\epsilon(t_{\mathsf{anon}} - 1)$ away from $t_{\mathsf{anon}} - 1$. Combined with Equation (1), this implies that

$$\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell^y) \leq \mathsf{Uniq.Query}(\epsilon, \mathsf{st}_{t_{\mathsf{anon}}-1}^x) \leq (1+\epsilon)(t_{\mathsf{anon}} - 1) < (1+\epsilon)t_{\mathsf{anon}} = t$$

with overwhelming probability. However, by the contrapositive of the previous claim and the fact that $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell^y) < t$ with overwhelming probability, it follows that $\mathsf{Hyb}_6 \overset{c}{\approx} \mathsf{Hyb}_7$. $\square$

Observe that the challenge signature σ is independent of upk_b in Hyb_7. Thus, any adversary $\mathcal{A}$ wins the experiment in Hyb_7 with probability at most $1/2$. Since $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_7$, it follows that $\mathcal{A}$ wins the HTS source anonymity experiment with at most negligible probability more than $1/2$. Thus, the given scheme is t_{anon}-source anonymous against the opener. ■

Remark 3 (Selective Message Security). As discussed earlier, the construction achieves selective message security for source anonymity against the opener. In particular, the challenge message is fixed in advance and not chosen by the adversary during step 2 of the experiment (see Figure 2). This restriction arises from the structure of the proof: leveraging the knowledge soundness of the IVC requires constructing an adversary $\mathcal{A}_{\mathsf{ivc}}$ that internally simulates the entire experiment for the witness encryption adversary. To do so, $\mathcal{A}_{\mathsf{ivc}}$ must receive the *punctured* trapdoor as input before the pre-challenge phase, since the reduction ultimately relies on the puncturable knowledge soundness of the PNIZK. As a result, computing the punctured trapdoor necessitates fixing the challenge message at the outset of the experiment.

Remark 4 (Barriers to Adaptive Corruptions). In addition to selective message security (see Remark 3), another limitation arising from the proof is that the adversary cannot adaptively corrupt honest parties in the source anonymity experiment. Allowing such adaptive corruptions would require revealing the honest party's forwarding key fk and corresponding certificate cert to the adversary upon corruption. However, the reduction to the unforgeability of the signature scheme Sig relies on the adversary $\mathcal{A}_{\mathsf{int}}$ using fewer than t_{anon} certificates. Accommodating adaptive corruptions while ensuring that the forwarding keys of corrupted parties are consistent with the inputs to the streaming algorithm—as computed by HFwd prior to corruption—appears to require more than t_{anon} certificates, which would exceed the bound required for the reduction to go through.

Remark 5 (Inputs to the Streaming Algorithm). Because the challenge message is *fixed non-adaptively* in the source anonymity against opener experiment, the proof continues to hold even if inputs to the streaming algorithm are computed as $\mathsf{PRF}(\mathsf{fk}, m)$, rather than $\mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$. In this case, since the message is fixed before registration and all forwarding keys are sampled uniformly at random during Π_{reg}, the resulting inputs to the streaming algorithm are pseudorandom. Correctness and deletion robustness then suffice to argue that the estimate remains below t with overwhelming probability.

However, as noted in Remark 3, selective message security is an artifact of the proof strategy. We therefore adopt a more general design that decouples handling of the streaming algorithm from the selective-message restriction. In particular, when the message is chosen *adaptively* after some corrupt parties are registered, the input $\mathsf{PRF}(\mathsf{fk}, m)$ is no longer pseudorandom, and the estimate of the streaming algorithm can become arbitrarily inaccurate. Computing the inputs as $\mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}, m))$ ensures security even in the adaptive setting: pseudorandomness for corrupt parties registered before message selection follows from the security of $\mathsf{PRP}(k_{\mathsf{msg}}, \cdot)$, while pseudorandomness for parties registered afterward follows from the uniform sampling of fk in Π_{reg}.

Lemma 8 (Source Anonymity Against Registration Server)**.** *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is source anonymous against the registration server (Definition 10), where $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. The registration server, possessing the signing key $\mathsf{sk}_{\mathsf{sig}}$, can locally generate the keys of t_{trace} users. This allows it to forward the HTS signature through these users, decrypt the witness encryption and learn the group signature computed by the source. However, the source's anonymity remains intact due to the anonymity properties of the group signature scheme, since the registration server lacks access to the group signature opening key $\mathsf{osk}_{\mathsf{gs}}$. Observe that a user's signing key $\mathsf{usk}_{\mathsf{gs}}$, under the group signature scheme, is only used to compute the signature σ_{gs}. Thus, source anonymity against the registration server follows from a straightforward reduction to the anonymity of the group signature scheme. ■

Lemma 9 (Traceability and Unframeability)**.** *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is t_{trace}-traceable and t_{trace}-unframeable (Definition 10), where $t_{\mathsf{trace}} = t/(1-\epsilon)$, and $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. We focus on arguing traceability; the proof for unframeability follows a similar approach and is discussed later. We use a hybrid argument to show that for any non-uniform polynomial time adversary $\mathcal{A}$, the output of the traceability experiment is 1 with at most negligible probability.

- Hyb_0: This is the output of the traceability experiment when run with $\mathcal{A}$ i.e., this hybrid is equal to $\mathsf{ExpTrace}_{\mathsf{HTS},\mathcal{A}}(1^\lambda)$.
- Hyb_1: This hybrid is identical to the previous hybrid, except that the experiment outputs 0 when σ_ℓ fails to verify in step 3 of $\mathsf{Open}(\mathsf{msk}, m, \sigma_\ell)$.

  If σ_0 output by $\mathcal{A}$ fails to verify, then both Hyb_0 and Hyb_1 are 0. On the other hand, when σ_0 does verify, the correctness of the digital signature scheme and the completeness of the Puncturable NIZK and IVC guarantee that the IVC proof in σ_ℓ verifies successfully, since σ_ℓ is computed by forwarding σ_0 honestly. Thus, $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$.
- Hyb_2: This hybrid is identical to the previous hybrid, except that $\mathsf{PNIZK.SimProve}$ and td_* are used to simulate π_{fwd} when computing $\sigma_1, \ldots, \sigma_\ell$ and within HFwd. Here, td_* corresponds to

the output of $\mathsf{PNIZK.Puncture}(\mathsf{td}, \nu_*)$ where td is the trapdoor output by $\mathsf{PNIZK.Setup}$, and the predicate ν_* evaluates to 1 on all inputs.

$\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$ from a straightforward reduction to the puncturable zero-knowledge property of PNIZK.

- Hyb_3: This hybrid is identical to the previous hybrid, except that it is 0 if $L \geq t_{\mathsf{trace}}$ and $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) < t$, where st_ℓ is the streaming algorithm state in σ_ℓ and L is the number of unique honest users used to forward σ_0 and compute σ_ℓ.

  Observe that the adversary's view is identical in Hyb_2 and Hyb_3. The only difference between the two hybrids is that Hyb_2 is 1 when $L \geq t_{\mathsf{trace}}$ and $\mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell) < t$, while Hyb_3 is 0 in this case. We will show that the streaming algorithm's estimate on st_ℓ is at least t with overwhelming probability when $L \geq t_{\mathsf{trace}}$.

  Let k_{msg} be the PRF key in σ_0, and for each upk_i output by $\mathcal{A}$ let fk_i be the party's corresponding forwarding key. Let $\mathsf{st}'_0 := \perp$ and let

$$\mathsf{st}'_i := \mathsf{Uniq.Update}(\epsilon, \mathsf{st}'_{i-1}, \mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}_i, m))), \ \forall i \in [\ell].$$

  That is, st_ℓ in σ_ℓ denotes the streaming algorithm state obtained by updating the initial state st_0 in σ_0 with the inputs of the honest parties, whereas st'_ℓ refers to the state computed using only the honest parties' inputs, ignoring any inputs already incorporated into st_0. It then immediately follows from the insertion robustness property that

$$\mathsf{Uniq.Query}(\epsilon, \mathsf{st}'_\ell) \leq \mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell).$$

  Next, observe that the distribution of the state st'_ℓ is indistinguishable from that computed using L uniformly random inputs. This is because each $\mathsf{PRP}(k_{\mathsf{msg}}, \mathsf{PRF}(\mathsf{fk}_i, m))$ is pseudorandom: $\mathsf{PRF}(\mathsf{fk}_i, m)$ is pseudorandom since the adversary's view is independent of fk_i, and applying a permutation PRP to pseudorandom inputs yields pseudorandom outputs. In particular, note that it suffices to have oracle access to $\mathsf{PRF}(\mathsf{fk}_i, .)$ in this hybrid since π_{fwd} is simulated. It then follows from the correctness of the streaming streaming algorithm that the estimate on st'_ℓ is at most ϵL away from L. Combined with the previous observation, this implies that

$$t = (1-\epsilon)t_{\mathsf{trace}} \leq \mathsf{Uniq.Query}(\epsilon, \mathsf{st}'_\ell) \leq \mathsf{Uniq.Query}(\epsilon, \mathsf{st}_\ell)$$

  with overwhelming probability. Thus, $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$.

- Hyb_4: This hybrid is identical to the previous hybrid, except that the experiment outputs 0 if decryption of $\mathsf{ct}_{\mathsf{we}}$ within Open yields an invalid group signature.

  When the NIZK proof π_{nzk} fails to verify, both Hyb_3 and Hyb_4 are 0. On the other hand, if π_{nzk} verifies successfully, the soundness of the NIZK guarantees that $\mathsf{ct}_{\mathsf{we}}$ is an encryption of a valid group signature. It then follows from the soundness of the NIZK and the perfect correctness of the witness encryption scheme that $\mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4$.

- Hyb_5: This hybrid is always 0.

  The only difference between Hyb_4 and Hyb_5 is that Hyb_4 is 1 if the group signature σ_{gs}, obtained by decrypting $\mathsf{ct}_{\mathsf{we}}$, does not open to a corrupt user's identity, while Hyb_5 is 0 in this case. However, observe that since the adversary never queries HSign on the message m, it follows from a straightforward reduction to the traceability of the group signature scheme that $\mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5$.

It follows from our argument that $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_5$. Since Hyb_5 is always 0, it follows any non-uniform polynomial time adversary $\mathcal{A}$ wins the traceability experiment with at most negligible probability.

Lastly, note that the scheme's t_{trace}-unframeability property can be proved using a similar argument as the one used to prove t_{trace}-traceability. The key distinction is that with the adversary's view now consisting of $\mathsf{rsk}_{\mathsf{gs}}$, the indistinguishability between Hyb_4 and Hyb_5 reduces to the unframeability property of the group signature scheme. ■

Lemma 10 (Forwarder Anonymity). *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is forwarder anonymous (Definition 10), where $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. Consider any non-uniform polynomial time adversary $\mathcal{A}$. We define the output of the forwarder anonymity experiment to be 1 if $\mathsf{upk}_0, \mathsf{upk}_1$ are not corrupt at the end of the experiment, if the adversary did not query HFwd on m in the pre-challenge phase and if $b = b'$. The output of the experiment is defined as 0 in all other cases. We show that $\mathcal{A}$ wins the forwarder anonymity experiment with at most negligible probability.

- Hyb_0: This is the output of the forwarder anonymity experiment when run with $\mathcal{A}$.
- Hyb_1: This hybrid is identical to the previous hybrid, except that $\mathsf{PNIZK.SimProve}$ and td_* are used to simulate π_{fwd} when computing forwards on behalf of honest users within the HFwd oracle. Here, $\mathsf{td}_* \leftarrow \mathsf{PNIZK.Puncture}(\mathsf{td}, \nu_*)$ where td is the trapdoor output by $\mathsf{PNIZK.Setup}$, and the predicate ν_* evaluates to 1 on all inputs.

  $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$ from a straightforward reduction to the puncturable zero-knowledge property of PNIZK.
- Hyb_2: This hybrid is identical to the previous hybrid, except for the following modification to HFwd_0 and HFwd_1. When forwarding any signature on the challenge message m output by $\mathcal{A}$, they compute the input to the streaming algorithm as $f_0 = \mathsf{PRP}(k_{\mathsf{msg}}, r_0)$ for upk_0 and $f_1 = \mathsf{PRP}(k_{\mathsf{msg}}, r_1)$ for upk_1, where r_0 and r_1 are uniformly random and sampled at the onset of the experiment.

  Let fk_0 and fk_1 denote the forwarding keys corresponding to upk_0 and upk_1 respectively. The only difference between the two hybrids is in the input to the PRP when forwarding signatures on the message m on behalf of upk_0 and upk_1. In Hyb_1, these are computed by evaluating the PRF at m, under fk_0 and fk_1, while in Hyb_2, these are uniformly random. However, observe that $\mathcal{A}$'s view is independent of fk_0 and fk_1 since the Puncturable NIZK proofs are simulated and the adversary never corrupts upk_0 and upk_1. Moreover, $\mathcal{A}$ does not query HFwd on (upk_0, m) nor (upk_1, m) in the pre-challenge phase, which implies that its view is independent of $\mathsf{PRF}(\mathsf{fk}_0, m)$ and $\mathsf{PRF}(\mathsf{fk}_1, m)$. It thus follows from the security of the PRF that $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$.

Observe that in Hyb_2, HFwd_0 and HFwd_1 are identical. Thus, in this hybrid, $b = b'$ with probability at most $1/2$ for any adversary $\mathcal{A}$. Since $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_2$, it follows that any non-uniform polynomial time adversary wins the forwarder anonymity game with at most negligible probability. ■

Theorem 11. *For all polynomials t and real numbers $\epsilon \leq 1/2$, the scheme described in Figure 8 is a $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-secure succinct HTS scheme (Definitions 10 and 11), where $t_{\mathsf{anon}} = t/(1+\epsilon)$, $t_{\mathsf{trace}} = t/(1-\epsilon)$, and $1/\epsilon$ is polynomial in the security parameter λ of the scheme.*

Proof. It follows from Lemmas 6 to 10 that the scheme is a $(t_{\mathsf{anon}}, t_{\mathsf{trace}})$-secure HTS scheme. We are left to argue that the scheme is a succinct HTS scheme. Observe that the only components of the HTS signature that are updated on each forward are the streaming algorithm and IVC proof. However, the succinctness of the streaming algorithm and compactness of IVC immediately imply that the size of the HTS grows sublinearly in the length of the forwarding path. It follows that the scheme is a succinct HTS scheme. ■

7 Extensions to HTS

In this section, we discuss how our construction from Section 6 can be extended to support alternative definitions of viral traceback.

7.1 Unique Forwarders in a Graph

As discussed in Section 2, virality is modeled using a virality predicate over the forwarding graph—a directed multi-graph that captures the flow of the message through the network, where nodes represent users and edges represent forwarding events (see Section 2.1). Our focus until now has been on the unique-forwarders-on-a-path predicate, which requires that the number of distinct users along any forwarding path from the source to a recipient is at least t. We now consider a more general predicate, which we refer to as *unique-forwarders-in-a-graph*. This predicate requires that the number of distinct users in a subgraph between the source and a sink node in the forwarding graph is at least t—that is, virality is determined based on the number of distinct users in a forwarding subgraph, rather than a single path. In the context of HTS schemes, each signature must track the number of distinct users in such a subgraph and, in effect, serves as a proof that the subgraph contains at least t distinct users. The source node corresponds to the originator of the message, and the sink node corresponds to the user performing the final forward that generated the signature. The signature can then be used to de-anonymize the source if the number of distinct users in the underlying subgraph reaches the threshold t. We next discuss how our construction from Section 6 can be modified to capture this virality predicate.

To enable tracking the number of distinct users in a subgraph, we augment an HTS scheme with the ability to *merge* signatures originating from the same source. Specifically, if a user receives the same message—originating from the same source—from two different forwarders, they can merge the corresponding signatures to compute the number of distinct users across both forwarding paths. More generally, this allows the recipient to count the number of distinct users in the union of the subgraphs associated with each signature. To support merging of signatures, we observe that we need two additional properties from our building blocks.

- **Streaming Algorithms with Mergeable State:** Our construction relies on streaming algorithms to count the number of distinct forwarders. Thus, to merge signatures, we need the ability to merge two streaming algorithm states—each corresponding to a different stream of inputs—such that the merged state counts the number of distinct inputs across both substreams. Specifically, consider two streaming algorithm states: one updated with inputs $(x_1, \ldots, x_n)$ and the other updated with the inputs $(y_1, \ldots, y_m)$. The merged state should count the number of distinct values in $(x_i)_{i=1}^n \parallel (y_i)_{i=1}^m$. In our HTS construction, the IVC ensures consistency across overlapping inputs: if the same party appears in both subgraphs being merged, then its contributions x_i and y_j to the respective states are equal, i.e., $x_i = y_j$.

Many streaming algorithms are already equipped with this property. For example, the streaming algorithm that we discuss in Figure 6 maintains the γ most minimum values in the stream, where γ depends only on the error parameter ϵ. It is then easy to see that one can merge two states of this streaming algorithm by retaining only the γ most minimum values from both the input states.

- **Proof Carrying Data:** The validity of the streaming algorithm state is ensured by the IVC. However, since we now hope to merge states of the streaming algorithm that belong to two different signatures (originating from the same source), we also need a mechanism to output a proof of correctness for the merged state. In more detail, the IVC in Figure 8 takes a streaming algorithm state and its proof of correctness, a witness, and the next state and produces a proof of correctness for the next state. To support merging of streaming algorithm states, we would require an IVC that as input two streaming algorithm states and their corresponding proofs of correctness, a witness, and the merged state and outputs a proof of correctness for the merged state. In other words, we require the IVC to *merge proofs* of correctness on each of its inputs and output a proof of correctness for its output. This property is provided by a generalization of IVC called Proof Carrying Data (PCD) [CT10] which is a primitive used to prove that each step of a distributed computation was carried out correctly, where in a single step a node that gets a set of inputs and proof of correctness for each input uses its local data to compute the input for the next step.

It is then easy to see that our construction can be modified to support the unique-forwarders-in-a-graph predicate by adding a merge algorithm to merge the states of the streaming algorithm and then computing a proof of correctness for the merged state. Moreover, the witness encryption now requires a PCD as the proof of correctness for the streaming algorithm state.

Note that this now allows honest parties to merge signatures and count distinct forwarders in the subgraph rather than counting forwarders only on a single forwarding path. While corrupt parties may choose not to merge signatures, they cannot arbitrarily inflate the count. As a result, source anonymity continues to hold, following a similar argument as in the original construction.

7.2 HTS with Forward Secrecy

Our definition of HTS schemes in Section 4 requires source anonymity only against a *non-colluding* registration server and E2EE server. Indeed, our construction in Section 6 is insecure if an adversary corrupts both servers: it can use the registration server's secret key to generate the key material of more than t_{trace} users, which in turn allows it to render any message viral and subsequently use the E2EE server's opening key to deanonymize the source. As discussed in Section 2.1.1, ensuring a meaningful notion of virality requires assuming that the registration server does not collude with the E2EE server.

Nevertheless, it is natural to ask if we can mitigate the impact of a colluding registration server. In this section, we discuss how our HTS construction can be extended to provide a notion of *forward secrecy*: users who register *after* a message is sent cannot contribute to the count of distinct forwarders for that message. Informally, this means that once a signature is created, only users who were registered at the time of creation can influence the hop count. This ensures that messages sent *before* the registration server is corrupted continue to preserve source anonymity, even if the adversary subsequently compromises the server and registers additional users in an attempt to inflate forwarder counts.

We propose the following modifications to our construction from Section 6 to ensure forward secrecy.

- The setup algorithm generates a keypair $(\mathsf{pk}, \mathsf{sk})$ for a public-key encryption scheme and includes the public key pk in the CRS. The corresponding secret key sk is not provided to any user or server.
- During registration, the user obtains its secret key as before, along with an additional random bitstring $r \in \{0,1\}^\lambda$. The registration server also receives a ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, \mathsf{fk}; r)$ i.e., it obtains an encryption (which we will later use as a commitment) of the user's forwarding key fk under the public key pk and randomness r.
- The registration server computes a Merkle hash h over all ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_N$ generated during user registrations and publishes h as part of the public parameters. This hash serves as a commitment to the set of users registered up to that point. Additionally, the server sends each user a Merkle opening (or hint) ρ_i of their ciphertext ct_i, which encrypts their forwarding key.
- While the input to the streaming algorithm is computed as before when forwarding signatures, the IVC statement requires each forwarder to prove the following: (1) there exists a Merkle opening ρ_i that opens h to ct_i (2) there exists randomness r_i and a forwarding key fk_i such that $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}, \mathsf{fk}_i; r_i)$, and (3) fk_i was used to compute the input to the streaming algorithm. Note that the IVC statement now includes the Merkle hash h. As a result, when a user generates a new signature, h is embedded in the statement of the witness encryption, which, in turn, verifies the IVC proof.

At a high level, forward secrecy is achieved by including the Merkle hash h in both the witness encryption statement and the IVC. This ensures that deanonymizing the source is only possible if the streaming algorithm is updated using inputs from users whose encrypted forwarding keys were included in the computation of h. As a result, users who register after the message was sent do not possess a valid opening under the Merkle hash embedded in the witness encryption, and therefore cannot contribute to updating the streaming algorithm state when forwarding the signature.

One limitation of the scheme described above is that, whenever a new user joins the system, the Merkle hash h must be recomputed, and updated openings (or hints) must be provided to all previously registered users. However, there are generic techniques to mitigate this overhead, allowing the number of updated hints required per user to scale only with $\mathcal{O}(\log N)$, where N is the total number of registered users [GHMR18, GHM+19, HLWW23, GKMR23].

Remark 6. A key advantage of the forward secure HTS variant described in this section is that the registration server no longer requires the signing key $\mathsf{sk}_{\mathsf{sig}}$ used to issue certificates in Figure 8. This is because the Merkle hash published by the server authenticates the forwarding keys held by users. Eliminating the need for this signing key is crucial to achieving forward secrecy: if an adversary were to obtain $\mathsf{sk}_{\mathsf{sig}}$, they could generate certificates for arbitrary forwarding keys and thereby deanonymize signatures retroactively—including those generated before the key was compromised. In the forward-secure variant, the only secret held by the registration server is the group signature registration key rsk, which does not impact source anonymity. Instead, it is used to ensure that users encrypt a valid identity within the witness encryption. Moreover, the unframeability property of the group signature scheme guarantees that even if an adversary learns rsk, it cannot be used to frame honest users.

Acknowledgments

We thank Kuldeep Meel for pointing us toward streaming algorithms for constructing succinct HTS schemes. Pedro Branco is supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation). Matthew Green would like to acknowledge support from the NSF under awards CNS-1955172, CNS-1653110, CNS-1854000 and CNS-1801479,NS-1329737 and CNS-195517 as well as from the Office of Naval Research under contract N00014-19-1-2292, a Security and Privacy research award from Google, and DARPA under Contract No. HR001120C0084. Aditya Hegde and Abhishek Jain were supported by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. Abhishek Jain was additionally supported by JP Morgan Faculty Award, and research gifts from Ethereum, Stellar, and Cisco. Part of this work was done while Gabriel Kaptchuk was at Boston University and supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project and DARPA under Agreement No. HR00112020021. Any opinions, views, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA or other sponsors.

References

[ABD$^+$21] Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 94–125, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[ABED$^+$0] Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. Adversarial laws of large numbers and optimal regret in online classification. *SIAM Journal on Computing*, 0(0):STOC21–154–STOC21–210, 0.

[ABJ$^+$22] Miklós Ajtai, Vladimir Braverman, T.S. Jayram, Sandeep Silwal, Alec Sun, David P. Woodruff, and Samson Zhou. The white-box adversarial data stream model. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '22, page 15–27, New York, NY, USA, 2022. Association for Computing Machinery.

[AJJM22] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Pre-Constrained Encryption. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[AWZ23] Damiano Abram, Brent Waters, and Mark Zhandry. Security-preserving distributed samplers: How to generate any crs in one round without random oracles. Cryptology ePrint Archive, Paper 2023/860, 2023. https://eprint.iacr.org/2023/860.

[Ban18] Samarth Bansal. 'China model of censorship': Proposal to trace online content sparks concern. Hindustan Times. Available at https://www.hindustantimes.com/india-news/china-model-of-censorship-proposal-to-trace-online-content-sparks-concern/story-ACbKKedJWFZKCvR55RVu5K.html, December 2018.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[BCD$^+$25] Pedro Branco, Arka Rai Choudhuri, Nico Döttling, Abhishek Jain, Giulio Malavolta, and Akshayaram Srinivasan. Black-box non-interactive zero knowledge from vector trapdoor hash. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 64–92, Cham, 2025. Springer Nature Switzerland.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

[BDH$^+$19] Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Ring signatures: Logarithmic-size, no setup - from standard assumptions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 281–311, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[BEJWY22] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *Journal of the ACM*, 69(2):1–33, January 2022.

[BEY20] Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, page 49–62, New York, NY, USA, 2020. Association for Computing Machinery.

[BF14] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 520–537, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[BGJP23] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. End-to-end secure messaging with traceability only for illegal content. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 35–66, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[BHM$^+$21] Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. Adversarial robustness of streaming algorithms through importance sampling, 2021.

[BJPY18] Elette Boyle, Abhishek Jain, Manoj Prabhakaran, and Ching-Hua Yu. The bottleneck complexity of secure multiparty computation. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018: 45th International Colloquium on Automata, Languages and Programming*, volume 107 of *LIPIcs*, pages 24:1–24:16, Prague, Czech Republic, July 9–13, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 738–767, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[Bla25] Dan Black. Google Threat Intelligence Group: Signals of Trouble: Multiple Russia-Aligned Threat Actors Actively Targeting Signal Messenger, 2025.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153, San Francisco, CA, USA, February 14–18, 2005. Springer, Heidelberg, Germany.

[BYJK+02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting Distinct Elements in a Data Stream. In G. Goos, J. Hartmanis, J. van Leeuwen, José D. P. Rolim, and Salil Vadhan, editors, *Randomization and Approximation Techniques in Computer Science*, volume 2483. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. Series Title: Lecture Notes in Computer Science.

[CC20] Helena Costa and Mônica Chaves. Infected texts: disinformation meets polarization in Brazil. Heinrich Böll Stiftung. Available at https://eu.boell.org/en/2020/06/22/infected-texts-disinformation-meets-polarization-brazil, June 2020.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1082–1090, Phoenix, AZ, USA, June 23–26, 2019. ACM Press.

[Cer] Certificate transparency : Certificate transparency. https://certificate.transparency.dev/.

[Cho18] Rohit Chopra. Disinformation Spreads on WhatsApp Ahead of Brazilian Election. The New York Times. Available at https://www.nytimes.com/2018/10/19/technology/whatsapp-brazil-presidential-election.html, October 2018.

[Cho19] Rohit Chopra. In India, WhatsApp is a weapon of antisocial hatred. The Conversation. Available at https://theconversation.com/in-india-whatsapp-is-a-weapon-of-antisocial-hatred-115673, April 2019.

[CK16] Amit Chakrabarti and Sagar Kale. Strong fooling sets for multi-player communication with applications to deterministic estimation of stream statistics. In Irit Dinur, editor, *57th Annual Symposium on Foundations of Computer Science*, pages 41–50, New Brunswick, NJ, USA, October 9–11, 2016. IEEE Computer Society Press.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 310–331, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

[Cv91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

[CVM22] Sourav Chakraborty, N. V. Vinodchandran, and Kuldeep S. Meel. Distinct elements in streams: An algorithm for the (text) book, 2022.

[DJJ+25] Pratish Datta, Abhishek Jain, Zhengzhong Jin, Alexis Korb, Surya Mathialagan, and Amit Sahai. Incrementally Verifiable Computation for NP from Standard Assumptions. In Seny Kamara and Yael Tauman Kalai, editors, *Advanced in Cryptology — CRYPTO 2025*. Springer, Heidelberg, Germany, 2025.

[DP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science*, pages 427–436, Pittsburgh, PA, USA, October 24–27, 1992. IEEE Computer Society Press.

[FEFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science*, DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07), jan 2007.

[FJW24] Ying Feng, Aayush Jain, and David P. Woodruff. Fast white-box adversarial streaming without a random oracle, 2024.

[FK15] Alan Frieze and Michał Karoński. *Introduction to Random Graphs*. Cambridge University Press, 2015.

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, sep 1985.

[FZ13] Matthew K. Franklin and Haibin Zhang. Unique ring signatures: A practical construction. In Ahmad-Reza Sadeghi, editor, *FC 2013: 17th International Conference on Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 162–170, Okinawa, Japan, April 1–5, 2013. Springer, Heidelberg, Germany.

[GGHW14] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 518–535, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GHM$^+$19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 63–93, Beijing, China, April 14–17, 2019. Springer, Heidelberg, Germany.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018: 16th Theory of Cryptography Conference, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 689–718, Panaji, India, November 11–14, 2018. Springer, Heidelberg, Germany.

[GKL21] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 553–583, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[GKM$^+$22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 252–282, Virtual Event, March 8–11, 2022. Springer, Heidelberg, Germany.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1065–1079, New York, NY, USA, 2023. Association for Computing Machinery.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 339–358, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[HAN23] Mathias Hall-Andersen and Jesper Buus Nielsen. On valiant's conjecture: Impossibility of incrementally verifiable computation from random oracles. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 438–469, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[HIJ$^+$16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 157–168, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 511–542, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[HW13] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 121–130, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[IAV22] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. Hecate: Abuse reporting in secure messengers with sealed sender. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 2335–2352, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 3–32, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 60–73, Virtual Event, Italy, June 21–25, 2021. ACM Press.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part I*, volume 13275 of *Lecture Notes in Computer Science*, pages 670–699, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[KLN23] Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 594–625, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[KNW10] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS ’10, page 41–52, New York, NY, USA, 2010. Association for Computing Machinery.

[KS92] Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

[LL23] Sean Lawlor and Kevin Lewi. Deploying key transparency at whatsapp. https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/, apr 2023.

[LLK13] Ben Laurie, Adam Langley, and Emilia Kasper. Rfc 6962 - certificate transparency. https://datatracker.ietf.org/doc/html/rfc6962, Jun 2013.

[LR89] Michael Luby and Charles Rackoff. A study of password security. *Journal of Cryptology*, 1(3):151–158, October 1989.

[LRTY22] Linsheng Liu, Daniel S. Roche, Austin Theriault, and Arkady Yerukhimovich. Fighting fake news in encrypted messaging with the fuzzy anonymous complaint tally system (FACTS). In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

[MBB$^+$15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015: 24th USENIX Security Symposium*, pages 383–398, Washington, DC, USA, August 12–14, 2015. USENIX Association.

[MFS23] Sam A. Markelon, Mia Filić, and Thomas Shrimpton. Compact frequency estimators in adversarial environments. Cryptology ePrint Archive, Paper 2023/1366, 2023. https://eprint.iacr.org/2023/1366.

[New21] Newley Purnell and Jeff Horwitz. WhatsApp Says It Filed Suit in India to Prevent Tracing of Encrypted Messages. Available from https://www.wsj.com/world/india/whatsapp-says-it-filed-suit-in-india-to-prevent-tracing-of-encrypted-messages-11622000307, May 2021.

[oET21] Ministry of Electronics and Information Technology. Intermediary guidelines and digital media ethics code rules. Gazette Of India. Available at https://web.archive.org/web/20230124115918/https://www.meity.gov.in/writereaddata/files/Intermediary_Guidelines_and_Digital_Media_Ethics_Code_Rules-2021.pdf, 2021.

[PEB21] Charlotte Peale, Saba Eskandarian, and Dan Boneh. Secure complaint-enabled source-tracking for encrypted messaging. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 1484–1506, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[PFG$^+$23] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar Jois, and Yinzhi Cao. Squint hard enough: Attacking perceptual hashing with adversarial machine learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 211–228, Anaheim, CA, August 2023. USENIX Association.

[PR21] Kenneth G. Paterson and Mathilde Raynal. HyperLogLog: Exponentially bad in adversarial settings. Cryptology ePrint Archive, Report 2021/1139, 2021. https://eprint.iacr.org/2021/1139.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[RS20] Katitza Rodriguez and Seth Schoen. FAQ: Why Brazil's Plan to Mandate Traceability in Private Messaging Apps Will Break User's Expectation of Privacy and Security. Electronic Frontier Foundation. Available at https://www.eff.org/deeplinks/2020/08/faq-why-brazils-plan-mandate-traceability-private-messaging-apps-will-break-users, August 2020.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[SHNK22] Lukas Struppek, Dominik Hintersdorf, Daniel Neider, and Kristian Kersting. Learning to break deep perceptual hashing: The use case neuralhash. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 58–69, 2022.

[SM23] Sarah Scheffler and Jonathan R. Mayer. Sok: Content moderation for end-to-end encryption. *Proc. Priv. Enhancing Technol.*, 2023(2):403–429, 2023.

[TMR19] 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: 26th Conference on Computer and Communications Security*, pages 413–430, London, UK, November 11–15, 2019. ACM Press.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 535–559, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[U.S22] U.S. Department of Justice. Former Twitter Employee Found Guilty of Acting as an Agent of a Foreign Government and Unlawfully Sharing Twitter User Information. Available at https://www.justice.gov/opa/pr/former-twitter-employee-found-guilty-acting-agent-foreign-government-and-unlawfully-sharing, August 2022.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-IO from evasive LWE. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 195–221, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.

[Wha23] WhatsApp. WhatsApp Encryption Overview. Available at https://www.noahpinion.blog/p/dont-be-a-decel, September 2023.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on*

Theory of Computing, STOC ’79, page 209–213, New York, NY, USA, 1979. Association for Computing Machinery.