Anonymous Complaint Aggregation for Secure Messaging

Connor Bell
University of North Carolina at Chapel Hill
connorbe@cs.unc.edu

Saba Eskandarian
University of North Carolina at Chapel Hill
saba@cs.unc.edu

ABSTRACT

Private messaging platforms provide strong protection against platform eavesdropping, but malicious users can use privacy as cover for spreading abuse and misinformation. In an attempt to identify the sources of misinformation on private platforms, researchers have proposed mechanisms to trace back the source of a user-reported message (CCS '19,'21). Unfortunately, the threat model considered by initial proposals allowed a single user to compromise the privacy of another user whose legitimate content the reporting user did not like. More recent work has attempted to mitigate this side effect by requiring a threshold number of users to report a message before its origins can be identified (NDSS '22). However, the state of the art scheme requires the introduction of new probabilistic data structures and only achieves a "fuzzy" threshold guarantee. Moreover, false positives, where the source of an unreported message is identified, are possible.

This paper introduces a new *threshold source tracking* technique that allows a private messaging platform, with the cooperation of a third-party moderator, to operate a threshold reporting scheme with exact thresholds and no false positives. Unlike prior work, our techniques require no modification of the message delivery process for a standard source tracking scheme, affecting only the abuse reporting procedure, and do not require tuning of probabilistic data structures.

1 INTRODUCTION

End-to-end encrypted (E2EE) messaging platforms allow users the opportunity to communicate without possible eavesdropping by the messaging platform itself. Widely deployed in Signal, WhatsApp, iMessage, Android Messages, and Messenger Secret Conversations, E2EE messaging has rapidly become the standard for privacy in mobile communication.

Unfortunately, the strong privacy protections of end-to-end encryption can also provide cover for malicious users who wish to propagate hate speech or disinformation without repercussions from platform moderators. In response to the pressing nature of this problem, various countries, including India and Brazil, have sought to introduce policies that compel messaging platforms to reveal the sources of misinformation messages [5, 37, 43, 44, 46, 47]. The policies proposed by these governments have received condemnation from platforms, policymakers, and technologists because they amount to roundabout ways of circumventing end-to-end encryption [3, 31].

While a number of works have studied handling abuse reports in E2EE messaging [8, 16, 20, 22, 28–30, 34, 48] or proactively scanning encrypted messages for inappropriate content [6, 11, 33], few works consider the problem of identifying the originators of user-reported misinformation without violating E2EE guarantees for non-reported messages. This problem has been studied under the name *traceback* by Tyagi et al. [49] and *source tracking* by Peale et al. [42] (we will refer to this functionality as source tracking in this paper). In source tracking, clients can verify that a received message, along with related metadata (e.g. the author), is traceable back to the original sender, or the direct sender if the message was not a forwarded message.

Unfortunately, allowing a single user report to reveal the source of a message can be problematic, as any user who dislikes the contents of a given forwarded message can reduce the privacy that the platform provides to the author of that message. For example, a user who receives widely-forwarded details about the time and place of a planned protest can cause the platform to learn who sent the messages planning the protest. This means that source tracking allows users of a messaging platform to de-anonymize other users to the platform, *even if they have never communicated with each other directly.*

Recently, Liu et al. [35] have introduced FACTS, a scheme for anonymous tallying of misinformation messages. FACTS allows for a message to be reported to a platform for source tracking after it is reported a certain number of times, in hopes of reducing the risk posed by allowing a single user to deanonymize another. This does not prevent a malicious user who receives a message from directly revealing the necessary reporting data to the platform operator out-of-band, but it provides a way for honest users to prevent reporting of content whose objectionable status has not yet been widely confirmed. In the FACTS system, clients update a probabilistic data structure each time they report a message, and messages that have received roughly the correct number of reports are revealed to the platform for source tracking. FACTS is the first system to support this kind of *threshold source tracking*.

This paper introduces a new system for threshold source tracking. Unlike FACTS, our system allows for exact thresholds for reporting messages, never has false positives, and does not require locking a global data structure for each report. Moreover, we make no changes to how message processing or delivery is handled beyond standard source tracking. The modest overhead introduced by our scheme occurs only during the reporting process.

Our key technical contribution is a new two server *anonymous tally scheme*, a primitive of independent interest. In the context of source tracking, we split the work of handling anonymous report tallying between the platform itself and a third-party moderator. Security critically relies on the non-collusion of these two parties. Users only interact with the platform during the reporting process, meaning that the platform necessarily learns which users make reports, but does not learn anything about the messages being reported. The platform occasionally passes on report data to the moderator, allowing the moderator to tally reports for a message.

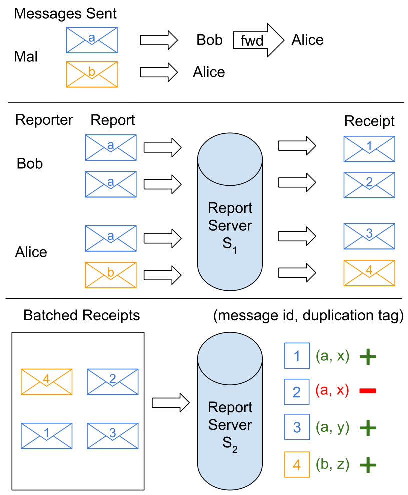

Figure 1: The expected outcomes for sample message reporting behavior. - Bob receives a from Mal and forwards to Alice. Alice receives b from Mal. Bob attempts to report a twice, and Alice honestly reports a and b. S_1 returns an anonymous receipt of the interactions back to Alice and Bob. These receipts are encrypted by Bob and Alice before being batched and shuffled for delivery to S_2 for validation; Bob's second report of a produces an identical duplication tag to the first report, so only one report will count, while both of Alice's honest reports are accepted for counting.

Only after a message receives sufficient reports is it revealed to the platform/moderator. Fig. 1 shows how the expected behavior when the system identifies duplicate reports from the same user while allowing multiple users to report the same message.

To analyze our proposed scheme, we formalize and prove security with respect to security definitions that ensure a given user cannot contribute more than one tally toward revealing a message, that report contents remain hidden from the platform before a message is reported a threshold number of times, and that the moderator learns nothing about who makes each report.

We implement our proposed scheme and find that the additional overhead of our reporting protocol and report verification algorithm each take well under 1ms of client or server computation time to complete. The computational cost of our scheme ranges from comparable to orders of magnitude lower than the FACTS system, depending on the choice of threshold and parameter settings used for FACTS. Our code is open source and publicly available at https://github.com/connorbelll/anonymous-tally.

Threat Model Limitations. We wish to point out three important limitations of our work on threshold source tracking, both to clarify our contributions and to point out promising avenues for future work.

First and foremost, as mentioned above, threshold source tracking (both in this and prior work) does not prevent a malicious user from colluding with a malicious server and immediately revealing the source of a message, thereby circumventing any threshold mechanism. Concretely, this means that if a user produces a piece of widely forwarded content to which a government or other powerful entity objects, the messaging system can be compelled to strip that user's anonymity after seeing only a single report from a government-controlled device that has received the message.

Second, we note that our scheme introduces the need to split trust between multiple servers to achieve security. While this is a widely-used assumption in the anonymous messaging literature, the performance, functionality, and security benefits of our approach need to be weighed against this additional requirement in making deployment decisions for threshold source tracking. We discuss this assumption in more detail in Section 3.

Finally, in order for threshold source tracking to be effective, it must be paired with an effective mechanism for preventing the creation of fraudulent accounts. While the techniques for preventing fake accounts are orthogonal to the mechanisms used for source tracking, we observe that using these two security features together places additional importance on preventing fake accounts. Whereas these mechanisms usually function to prevent malicious users from misbehavior caused by generating harmful content, e.g., bullying or producing misinformation, our scheme also relies on them to protect the anonymity of users whose messages are reported. Such a shift in security properties relying on duplicate account prevention may also raise the motivation of attackers to circumvent measures already in place.

In summary, this paper makes the following contributions:

- Introduces the notion of a two-server anonymous tally scheme and presents the appropriate security definitions.
- Shows how to use anonymous tally schemes to build threshold source tracking.
- Designs an anonymous tally scheme that enables threshold source tracking with exact thresholds.
- Implements and evaluates our anonymous tally scheme, including a comparison to prior work, demonstrating that our scheme achieves low overheads and is suitable for practical applications.

2 BACKGROUND: SOURCE TRACKING AND THRESHOLD SOURCE TRACKING

In a source tracking scheme, the platform augments the message delivery process with additional information that can be used to identify the originator of a reported message. When a user wishes to report a message, it produces a report that consists of the message and additional cryptographic material that aids the moderator in verifying that the report was indeed sent through the platform by the claimed sender. The only property we require of the underlying source tracking scheme in this work is that the process of reporting a message to the moderator does not require multiple rounds of interaction and that the actual data sent to report a message does not depend on the user sending the report. This property holds in the "tree-linkable" variant of the Peale et al. source tracking scheme [42], their more efficient construction, as well as Hecate [30]. Schemes that trace back a message to its source hop by hop [32, 49] do not satisfy this requirement because the lack of a consistent cryptographic identifier for the forwarded message works against the platform's ability to aggregate reports.

A threshold source tracking scheme augments the source tracking process by adding a mechanism where messages are revealed for source tracking after the servers have received a certain number of complaints about a given message. The only known threshold source tracking scheme is FACTS [35], where clients collaboratively update a data structure hosted by the server to keep track of *approximately* how many times a message has been reported. When clients detect that a message has been reported enough times, any reporting client can make a final report to the moderator. The final report reveals the information necessary for source tracking to the moderator. FACTS does not prevent users from submitting multiple reports for the same message and may additionally leak honest users' intended reports to the platform when the platform becomes aware of the report identifier, both of which we aim to address in this work. We also track exact, instead of approximate, report counts. In both FACTS and our work, the focus is on allowing a moderator to be notified when a message has received enough reports, not to prevent malicious clients from sharing reports with malicious moderation servers out of band, as many source tracking schemes provide any recipient of a forwarded message with sufficient information to report the message alone.

Threshold source tracking shares some common goals with electronic voting; in elections, votes should remain anonymous while preventing any single voter from voting on the same issue twice. In this work, we present de-duplication constructions which surface common identifying strings if a malicious user attempts to report the same message twice. Similar definitions were established for unique ring signatures by Franklin and Zhang [25], where malicious duplicate signatures will result in "a large common component" between the signatures, which can be used to link the duplicate signatures together. We include a further discussion in Appendix A to compare anonymous report aggregation to electronic voting more broadly and to illustrate why our solution takes a different approach than common electronic voting tools such as traceable ring signatures [27].

3 ANONYMOUS TALLIES

This section introduces *anonymous tallies* and sketches their properties at a high level. Since anonymous tallies form the core of our threshold source tracking scheme, we begin by introducing them before showing how to integrate them with existing messaging systems to support threshold source tracking.

A two-server anonymous tally scheme allows two servers to blindly keep a count of user-reported messages. The servers can learn the number of distinct user reports of a given message, but they do not learn the messages themselves or the identity of the user who filed each report.

The design of our scheme has the two servers playing distinct roles. Users interact with the first server, S_1, to send a report for tallying. Server S_1 sends batches of anonymized reports to S_2, who computes the anonymous tallies. For each report sent to the tallying scheme, the server S_2 can derive a *duplication tag* dupTag which will be identical if the same user reports the same message more than once. The dupTag can be used to detect and discard duplicate reports. Server S_2 also derives some *hidden data* hd which it can send to S_1 to enable recovery of *report data* rd sent by the client. Server S_2 can also prove to S_1 that the tally for a given report has passed a given threshold. This abstraction allows us to easily integrate our anonymous tally scheme syntax with different message reporting schemes.

We require the following high-level security properties from an anonymous tally scheme.

- **Report confidentiality**: a single server behaving maliciously, potentially colluding with malicious users, cannot learn the contents of honest users' reports.
- **Reporter anonymity**: a single server behaving maliciously, potentially colluding with malicious users, cannot learn which honest user sent which report.
- **Report uniqueness**: if the servers behave honestly, no malicious user should be able to contribute more than one report to the tally for a given report.
- **Threshold unforgeability**: a malicious S_2 cannot misrepresent a given report as having more than a threshold number of reports when it really does not.
- **Deniability**: even if user or server secrets are made public, reports cannot be verifiably tied back to a given user.

Looking ahead, our scheme will (necessarily) allow server S_1 to learn the identities of all the users who send reports, but hide which messages those users report. At the same time, server S_2 will learn the values being counted in the tally, but it will not be able to connect any given report with a particular user. To further mask the identities of the reporters, messaging clients can occasionally send a report with random report data as cover traffic for real reports.

In Section 5, we formalize these properties and discuss various additional security considerations.

Security from splitting trust. Our scheme relies on splitting trust between two non-colluding servers to achieve security. In particular, a deployment must be able to set up two servers, e.g., the message platform itself and a third party moderator-run server, who can be relied upon not to collude to violate the security of the anonymous tally. Failure to satisfy this assumption in practice allows the servers to deanonymize the author of any message after a single report, reverting the scheme to a standard (non-threshold) source tracking scheme.

While a two-server split trust setup may be difficult to achieve in many scenarios, recent large-scale deployments of split-trust systems for private browser telemetry in Mozilla Firefox [4, 21] and measurement of the effectiveness of the Apple/Google Covid-19 exposure notification system [1, 2] provide reason for optimism that this is a workable approach. The stakes in these deployments are, however, considerably lower than those of anonymous messaging, where potential privacy harms are not only the exposure of consumers' browsing or health data, but also persecution (and potentially execution [7]) of dissidents.

Anonymous tally scheme syntax. More formally, a two-server anonymous tally scheme consists of seven algorithms $\mathsf{SKGen1}$, $\mathsf{SKGen2}$, UKGen, Verify, Reveal, $\mathsf{S2Prove}$, and $\mathsf{S1Verify}$, and an interactive protocol Report.

- $\mathsf{SKGen1}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{sk}_s)$: The server controlled by the messaging platform, S_1, runs this algorithm at system initialization. It takes in a security parameter 1^λ and public parameters pp, and it generates 3 keys: public and secret

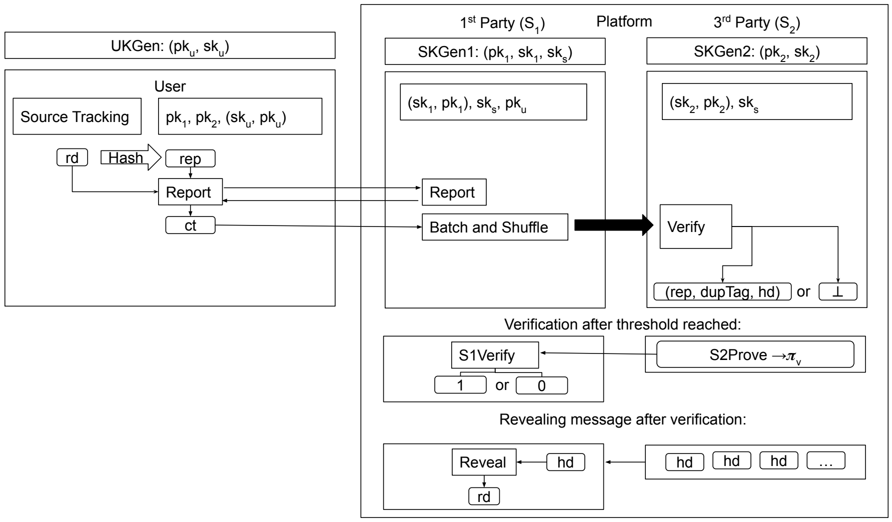

Figure 2: System diagram of threshold source tracking showing the ownership of keys and the flow of data. A user begins with the source tracking data *rd* and computes a report identifier rep **to feed the** Report **and** Verify **algorithms, resulting in either a)** S_2 **learning** rep**, a duplication tag** dupTag**, and hidden metadata** hd **or b)** S_2 **rejecting the report. When a platform-specified reporting threshold is reached,** S_2 **proves to** S_1 **that the threshold for the report was reached honestly and delivers the associated** *hd* **values for** S_1 **to** Reveal **the source tracking information.**

keys for interactions with the reporter, as well as a shared secret key sk_s.

- $\mathsf{SKGen2}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{pk}_2, \mathsf{sk}_2)$: The server controlled by the 3rd party, S_2 runs this algorithm at system initialization. It takes in a security parameter 1^λ and public parameters pp, and it generates public and secret keys for receiving encrypted messages.
- $\mathsf{UKGen}(1^\lambda, \mathsf{pp}) \rightarrow (\mathsf{pk}_u, \mathsf{sk}_u)$: A user runs this algorithm to participate in the system. It takes in a security parameter 1^λ and public parameters pp, and it generates a user key pair $\mathsf{pk}_u, \mathsf{sk}_u$.
- Report: $\langle U(\mathsf{rep}, \mathsf{rd}, \mathsf{pk}_u, \mathsf{sk}_u, \mathsf{pk}_1, \mathsf{pk}_2), S_1(\mathsf{pk}_1, \mathsf{sk}_s, \mathsf{sk}_1, \mathsf{pk}_u)\rangle \rightarrow \mathsf{ct}/\perp$: this is a protocol run between a user U and the first server S_1. Each party has access to relevant public and private keys, and the user U additionally holds values rep and rd. The value rd can be, e.g., the contents of a report in a source tracking scheme. rep is not the contents of a report in the underlying source tracking scheme, but rather a value that uniquely identifies the report, e.g., its hash. To allow for flexibility in use cases, there is no enforced relationship between rep and rd in the anonymous tally scheme itself.
- $\mathsf{Verify}(\mathsf{sk}_s, \mathsf{sk}_2, \mathsf{ct}) \rightarrow (\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})/\perp$: this algorithm is run by server S_2 to validate the contents of a report. The algorithm takes as inputs the keys sk_s and sk_2, as well as a ciphertext ct. If the ciphertext passes the server's verification process, the algorithm returns the values rep, dupTag, and hd. dupTag is used for detecting duplicate reports. If the same user sends the same rep twice, the second report will result in the same dupTag as the first report.
- $\mathsf{S2Prove}(\mathsf{rep}, \mathbf{pData}, \mathsf{thresh}) \rightarrow \pi_v$: this algorithm allows S_2 to produce a proof π_v that it has a report rep which has been reported at least thresh number of times. The **pData** input includes scheme-specific data needed by S_2 to produce this proof.
- $\mathsf{S1Verify}(\mathsf{rep}, \mathbf{vData}, \mathbf{dupTags}, \pi_v) \rightarrow 0/1$: this allows S_1 to verify the output of an execution of S2Prove. The **vData** input includes scheme-specific data needed by S_1 to verify this proof, and **dupTags** includes values of dupTag held by S_2 for the reports. This allows S_1 to ensure that S_2 only reveals messages that have exceeded the threshold.
- $\mathsf{Reveal}(\mathsf{sk}_1, \mathsf{hd}) \rightarrow \mathsf{rd}$: this algorithm is run by S_1 to recover report data rd from hidden data hd provided by S_2.

We define correctness for a two-server anonymous tally scheme as follows.

Definition 3.1 (Correctness (informal)). A two-server anonymous tally scheme is *correct* if when the servers and all users follow the scheme honestly, all algorithms and protocols fail (output $\perp$) with at most negligible probability, server 2 returns a duplicate dupTag from Verify upon receiving a duplicate report for the same message from the same honest user with probability one, and server 2 returns distinct dupTags for distinct user, message pairs with all but negligible probability. Moreover, proofs produced by S2Prove when run with a rep value that has thresh or more distinct reports are accepted by S1Verify. Finally, if an hd value output by Verify is given to $\mathsf{Reveal}(\mathsf{sk}_1, \cdot)$, the result will be the corresponding rd value provided by the reporting client.

4 THRESHOLD SOURCE TRACKING VIA ANONYMOUS TALLIES

A two-server anonymous tally scheme can be integrated into a messaging system that supports source tracking to build a threshold source tracking scheme with no changes to the underlying messaging system and minimal changes to the message reporting flow. This process is depicted in Fig. 2.

4.1 From Tallies to Threshold Source Tracking

In our scheme, the messaging platform is composed of the first party entity running the messaging service, who runs server S_1, and a third party entity who aids in the moderation process only, who runs S_2. Users only ever interact directly with the first party S_1. At system initialization, the servers will run SKGen1 and SKGen2 to set up their respective keys and user devices will run UKGen in the process of registering to use the messaging platform (on top of any other registration processes). Then users can send messages using the underlying messaging scheme with no modifications, until they want to report messages.

When a user wishes to report a message m, it computes the report data rd for m via the source tracking scheme and hashes it with a hash function H to get a hashed report rep $\leftarrow H(\mathsf{rd})$. Throughout this paper, we will refer to the hashed report rep as the "report" for the purposes of the tallying scheme. This hashed report rep, in addition to the source tracking metadata rd itself, serves as user U's input rep to the anonymous tally scheme's Report protocol. The user U sends the resulting ciphertext ct to S_1 at the end of the Report protocol.

Periodically (on a system-specified schedule), the server S_1 sends a shuffled batch of ciphertexts to S_2. Server S_2 runs $\mathsf{Verify}(sk_s, \mathsf{sk}_2, \mathsf{ct})$ to recover the report rep, a deduplication tag dupTag, and hidden metadata hd for each ciphertext. Server S_2 keeps a table of dupTags and reports, and if a dupTag repeats, the report is dropped. Otherwise, it increments the count for the report rep.

Once the count for a given report rep passes a system-specified threshold thresh, S_2 will send the hidden metadata hd for the reports to S_1. S_2 also runs S2Prove to provide proof that the message in question received sufficient reports, while maintaining the privacy of the reporters by hiding the real set of reports among a list of masking reports; these proofs are verified by S_1 in S1Verify. S_1 will then run Reveal and verify that a given revealed metadata entry rd hashes to rep before proceeding further with processing the source tracking information. Alternatively, the hidden data hd in the anonymous tally scheme can be set to $\perp$, and the server S_1 can solicit users to come forward with the corresponding rd to a given rep once that message reaches the threshold. This latter approach is roughly the one taken in FACTS, so our scheme strictly increases flexibility in reporting options.

4.2 Choosing a Source Tracking Scheme

An anonymous tally scheme only affects the abuse reporting process in an E2EE messaging system, so it is compatible with any source tracking scheme where message reports consist of a single, reporter-independent, message sent from a user to the moderator. Thus our scheme is compatible with source tracking schemes that report message plaintext, message sender identity, and other platform-specified metadata, but can also be used with schemes that only report some subset of this data according to the platform's desired moderation policy.

Since we can generically add the anonymous tally step to the reporting process, we need not concern ourselves with the security details of the underlying source tracking scheme, which are not affected by the introduction of an anonymous tally to reporting. Thus, any threshold source tracking scheme built by adding an anonymous tally to an existing source tracking scheme inherits the security properties provided by the underlying scheme for unreported messages.

Finally, a threshold source tracking scheme built on top of a standard source tracking scheme inherits the limitations of the underlying system as well. In particular, source tracking schemes typically rely on users honestly following the protocol to ensure that messages can be linked back to the original sender, i.e., indicating that a message is being forwarded rather than copy/pasting the same text to forward a message.

4.3 Security for Threshold Source Tracking

Intuitively, splitting trust between S_1 and S_2 ensures that no malicious actor with control of the platform's (first-party) infrastructure can learn the contents of reports before they reach the specified threshold, while the deduplication tags revealed to S_2 allow it to learn nothing beyond a histogram of reported message frequencies, without knowing the report contents or the identities of the reporters. S_1 is the sole holder of the Reveal key for the hidden metadata, which is first delivered to S_2, so both parties must agree that the threshold is met for the metadata to be revealed.

We must allow, however, for the possibility of malicious users and servers colluding to artificially raise a message above the reporting threshold. We now briefly consider the possible combinations of malicious users and servers, discussing the possible consequences for each case.

Non-security of known messages. The security of threshold source tracking aims to keep a reported message hidden from the platform until that message receives sufficiently many reports. However, as discussed briefly in Section 1, it is possible for a malicious server S_1 to learn a particular message and its corresponding report data rd out-of-band and then abuse the source tracking system to identify the author of that message. This means that a threshold source tracking system does not strengthen the anonymity of message senders compared to a non-threshold source tracking system. Instead, it mitigates the risk of accidental or spurious abuse of the source tracking mechanism by individual users.

In Appendix B, we discuss the security ramifications for both our scheme and prior work in the situation where a malicious server S_1 does know the value rd of a reported message and wants to learn which other users are reporting the same message. While both our scheme and FACTS lose some degree of report confidentiality in this setting, we show that our scheme does provide a degree of protection not present in prior work.

In the remainder of this section, we consider the setting where threshold source tracking does provide additional security over conventional source tracking: where users are reporting messages not yet known to the servers.

Malicious users only. The anonymous tally's report uniqueness property ensures that, for a threshold of t reports to reveal a message, a group of fewer than t malicious users do not cause a message to be revealed. However, if an adversary has control of t or more malicious users (or can create t fake users), a message sent to this malicious group of users can always be revealed to the platform by having each malicious user report the message.

Our scheme does not handle issues of user authentication and validation, e.g., protecting against sybil attacks. An adversary who controls many users can report a message once per user it controls. We assume an external mechanism for authenticating users and ensure that, within the protocol, a single user cannot repeatedly report the same message to artificially increase its tally.

Malicious S_2 (and malicious users). The report anonymity property of the anonymous tally scheme, combined with the fact that S_1 shuffles and batches messages, ensures that S_2 cannot learn the identity of the sender of any given report. In a sense, S_1 acts similarly to a server in a mixnet [14], breaking the link between the report sender and the next server to receive the report.

However, a malicious S_2, potentially colluding with a malicious user who has a message it wants reported, could attempt to bypass the threshold mechanism by simply lying to S_1 about when a report has reached the threshold, bypassing the report uniqueness protections of the tally scheme. This potential attack is blocked by the verification protocol S_1 runs to protect against a malicious S_2. Observe that this means that while S_1 can arbitrarily unmask the identities of senders of known messages, S_2 cannot. This is another point where protections against fraudulent accounts are critical, lest a malicious S_2 create new fake accounts to artificially inflate the count of reports for a given message.

Malicious S_1 (and malicious users). The report confidentiality property of the anonymous tally ensures that a malicious S_1, potentially colluding with some malicious users, cannot learn the contents of the report of an honest user, so long as the server does not already know the contents of the report. See above for the case where the report contents are already known by the server.

Note that the confidentiality property of the underlying source tracking scheme implies that the contents of reports have high entropy, or else an adversary against the underlying source tracking scheme could simply guess-and-check reports for ciphertexts it wants to decrypt, breaking any confidentiality in the messaging system. This means that even guessing message contents given a ciphertext does not suffice for S_1 to predict the contents of rd.

5 SECURITY FOR ANONYMOUS TALLIES

We now discuss the formal security definitions for a two-server anonymous tally scheme. This section fully describes and formally defines our required security properties.

Recall that at a high level, our definitions will allow server S_1 to learn who submits reports and server S_2 to learn derivatives of the reports themselves, but neither server will learn which user made which report. At the same time, the servers need assurance that malicious users cannot take advantage of their strong privacy protections to fraudulently report a single message multiple times.

5.1 Notation

Before we continue, we formalize our notation. The following notation is used to describe various operations in the definitions and schemes presented in the rest of this paper.

Let $x \leftarrow F(y)$ denote assignment of the output of $F(y)$ to x, and let $x \xleftarrow{R} S$ denote assignment to x of an element sampled uniformly random from a set S. A bolded variable $\mathbf{x}$ denotes a vector, with entries in the vector represented as (non-bolded) $x_1, ...x_n$. We use $\mathcal{A}^O$ to denote that $\mathcal{A}$ has oracle access to some function(s) or can participate in a given set of interactive protocols, and the adversary $\mathcal{A}$ in our security experiments is allowed to be stateful. A function $\mathsf{negl}(x)$ is *negligible* if for all $c > 0$, there is a x_0 such that for all $x > x_0$, $\mathsf{negl}(x) < \frac{1}{x^c}$. We omit x if the parameter is implicit. Finally, we use $\perp$ to indicate an empty message or special character indicating failure.

We define an interaction between two parties using the notation

$$\langle P_1(\text{params}), P_2(\text{params})\rangle \rightarrow \text{out}_1.$$

The first party in the protocol acts according to the protocol defined by P_1 and the second party acts according to P_2, and out_1 represents the output of the protocol. Only the first party has any output from interactive protocols in this paper.

Our security definitions use tables to keep track of important information about adversary queries. Tables are denoted with a capital T and a subscript name, and store key/value pairs. To add a key/value pair to a table, we use the notation $T[key] \leftarrow value$. We use standard set notation to check if a key is included in a table ($key \in T$). Sets use the same notation as tables, but only store a set of values. We use $\mathsf{set}(\mathbf{x})$ to convert a vector to a set of its unique constituent elements. Tables and sets defined in a security experiment are considered globally accessible by the experiment in the oracles and protocols allowed to the adversary in that experiment.

5.2 Report Confidentiality

Our first security property, report confidentiality requires that a malicious server S_1 does not learn anything about the reports sent through the system by honest users. This definition allows an adversary to control S_1 and an arbitrary number of malicious users while also being allowed to register honest users and compel them to report messages. At the core of this game is the adversary's power to run the Report protocol with a provided user, identified by a user id uid, and one of two potential messages. The experiment has an input b that determines which report is actually sent.

At any point in the report confidentiality experiment, the adversary may call a Process oracle, which plays the role of S_2 on a set S of ciphertexts and a reporting threshold provided by the adversary S_1. The set S consists of a subset of the ciphertexts returned by honest users in the Report protocol, as well as any additional ciphertexts the adversary chooses to send. The Process oracle verifies each ciphertext, discards duplicates, and keeps tallies for each report rep. The oracle returns a table R of reported messages and report frequencies, as well as the S_2 verification proof π_v if the provided threshold is exceeded for any message. In order to prevent trivial

```
RCONF[𝒜, Π, λ, b] :
(pk_2, sk_2) ← SKGen2(1^λ, pp)
(pk_1, sk_1, sk_s) ← 𝒜(1^λ, pk_2)
U ← {}; T ← {}; S ← {}
b′ ← 𝒜^O(1^λ)
output b′

Report(uid, rep_0, rd_0, rep_1, rd_1) :
if uid ∉ U : output ⊥
(sk_u, pk_u) ← U[uid]
ct ← ⟨U(rep_b, rd_b, pk_u, sk_u, pk_1, pk_2), 𝒜⟩
if ct = ⊥ : output ⊥
T[ct] ← {uid, rep_0, rep_1}
return ct

AddHonUser(uid) :
if uid ∈ U : output ⊥
sk_u, pk_u ← UKGen(1^λ, pp)
U[uid] ← (sk_u, pk_u)
output pk_u

Submit(ct) :
S ← S ∪ {ct}

Process(thresh) :
R ← {}; R_0 ← {}; R_1 ← {}; P ← {};
D ← {}; D_0 ← {}; D_1 ← {};
for ct ∈ S :
  (rep, dupTag, hd) ← Verify(sk_s, sk_2, ct)
  if (rep, dupTag, hd) = ⊥ : continue
  if (rep, dupTag) ∉ D :
    D ← D ∪ (rep, dupTag)
    R[rep] ← R[rep] + 1
  if ct ∈ T :
    (uid, rep_0, rep_1) ← T[ct]
    if (uid, rep_0) ∉ D_0 :
      D_0 ← D_0 ∪ {(uid, rep_0)}
      R_0[rep_0] ← R_0[rep_0] + 1
    if (uid, rep_1) ∉ D_1 :
      D_1 ← D_1 ∪ {(uid, rep_1)}
      R_1[rep_1] ← R_1[rep_1] + 1
if R_0 ≠ R_1 : Abort experiment, return 0
for rep ∈ R where R[rep] > thresh :
  P ← P ∪ S2Prove(rep, pData, thresh)
output R, P, dupTags //pData will include dupTags
```

Figure 3: Report confidentiality experiment RCONF **(Definition 5.1).**

```
RANON[𝒜, Π, λ, b] :
(pk_1, sk_1, ⊥) ← SKGen1(1^λ, pp)
(pk_2, sk_2, sk_s) ← 𝒜(1^λ, pk_1)
U ← {}; R_0 ← {}; R_1 ← {}
b′ ← 𝒜^O(1^λ)

AddHonUser(uid) :
if uid ∈ U : output ⊥
pk_u, sk_u ← UKGen(1^λ, pp)
U[uid] ← (sk_u, pk_u)
output pk_u

MalReport(pk_u) :
if (·, pk_u) ∈ U : output ⊥
⟨𝒜, S_1(pk_1, sk_s, sk_1, pk_u)⟩

HonReport(uid_0, uid_1, rep, rd) :
if uid_0 ∉ U or uid_1 ∉ U : output ⊥
(sk_u, pk_u) ← U[uid_b]
ct ← ⟨U(rep, rd, pk_u, sk_u, pk_1, pk_2), S_1(pk_1, sk_s, sk_1, pk_u)⟩
if ct = ⊥ : output ⊥
if (uid_0, rep) ∈ R_0 or (uid_1, rep) ∈ R_1 : output ⊥
R_0 ← R_0 ∪ {(uid_0, rep)}; R_1 ← R_1 ∪ {(uid_1, rep)}
output ct
```

Figure 4: Reporter anonymity experiment RANON **(Definition 5.2).**

wins, the experiment will abort and return 0 if the adversary calls Process while the tally is in a state where there would be different numbers of reports from honest users if $b = 0$ vs $b = 1$ in an honest execution of the protocol.

Note that the adversary in this game is stronger than is needed in the threshold source tracking setting, where a malicious S_1 (potentially colluding with some users) does not know, and cannot guess, the contents of honest users' reports. The check that the game makes to ensure that an honestly-generated R would have the same state regardless of whether $b = 0$ or $b = 1$ is there to rule out attacks that would not be possible in threshold source tracking due to the adversary not actually knowing rd and rep.

Definition 5.1 (Report Confidentiality)**.** We define the report confidentiality experiment $\mathsf{RCONF}[\mathcal{A}, \Pi, \lambda, Q_O, b]$ with respect to a stateful adversary $\mathcal{A}$, a list of numbers Q_O setting upper limits on the number of queries $\mathcal{A}$ makes to each of its oracles, a two-server anonymous tally scheme Π, a security parameter λ, and a bit b. The experiment is described in Figure 3. While not explicitly included in the description, we assume that the experiment retains the relevant transcript data from S_2 in the Report protocol in order to produce **pData** for S2Prove.

We define the *confidentiality advantage* of $\mathcal{A}$ as

$$\begin{aligned}&\mathsf{CONFAdv}(\mathcal{A}, \Pi, \lambda, Q_O)\\ &\qquad = \Big|\Pr[\mathsf{RCONF}[\mathcal{A}, \Pi, \lambda, Q_O, 0] = 1]\\ &\qquad\qquad - \Pr[\mathsf{RCONF}[\mathcal{A}, \Pi, \lambda, Q_O, 1] = 1]\Big|.\end{aligned}$$

We say that Π satisfies *report confidentiality* if for all PPT adversaries $\mathcal{A}$ and security parameters $\lambda \in \mathbb{N}$, it holds that

$$\mathsf{CONFAdv}(\mathcal{A}, \Pi, \lambda, Q_O) \leq \mathsf{negl}(\lambda).$$

5.3 Reporter Anonymity

Whereas report confidentiality protects against a malicious S_1 learning which messages are reported, reporter anonymity protects against a malicious S_2 learning the identities of users reporting messages. This definition allows an adversary to control S_2 and an arbitrary number of malicious users, who can interact with an honest S_1, while also being allowed to register honest users and compel them to report messages. At the core of this game is the adversary's power to have one of two honest users of its choosing interact with the honest S_1 to submit a report of its choosing. The $\mathsf{HonReport}(\mathsf{uid}_0, \mathsf{uid}_1, \mathsf{rep}, \mathsf{rd})$ oracle takes in the identifiers for two honest users and has one of them, determined by an input bit b, send a report rep with report data rd to S_1 via the Report protocol. The resulting ciphertext ct output by the protocol is returned to the adversary, as this is what S_2 receives from S_1 in our application. After sending a number of reports of its choosing, the adversary outputs a distinguishing bit b'.

To prevent trivial wins, the HonReport oracle outputs $\perp$ if the adversary attempts to have an honest user submit a duplicate report. Allowing duplicate reports trivially allow an adversary to distinguish $b = 0$ from $b = 1$. For example, an adversary who submits $\mathsf{HonReport}(\mathsf{uid}_0, \mathsf{uid}_1, \mathsf{rep}, \mathsf{rd})$ and $\mathsf{HonReport}(\mathsf{uid}_0, \mathsf{uid}_2, \mathsf{rep}, \mathsf{rd})$, will identify a duplicate report if $b = 0$ but not if $b = 1$. This is an acceptable restriction because an honest user does not have any reason to submit an identical report twice.

Definition 5.2 (Reporter Anonymity)**.** We define the reporter anonymity experiment $\mathsf{RANON}[\mathcal{A}, \Pi, \lambda, b]$ with respect to a stateful adversary $\mathcal{A}$, two-server anonymous tally scheme Π, security parameter λ, and a bit b. The experiment is described in Figure 4.

We define the *anonymity advantage* of $\mathcal{A}$ as

$$\begin{aligned}\mathsf{ANONAdv}&(\mathcal{A}, \Pi, \lambda)\\ &= \big|\Pr[\mathsf{RANON}[\mathcal{A}, \Pi, \lambda, 0] = 1]\\ &\qquad - \Pr[\mathsf{RANON}[\mathcal{A}, \Pi, \lambda, 1] = 1]\big|.\end{aligned}$$

We say that a scheme Π satisfies *reporter anonymity* if for all PPT adversaries $\mathcal{A}$ and security parameters $\lambda \in \mathbb{N}$, it holds that

$$\mathsf{ANONAdv}(\mathcal{A}, \Pi, \lambda) \leq \mathsf{negl}(\lambda).$$

5.4 Report Uniqueness

The report uniqueness property ensures that honest servers S_1 and S_2 can keep accurate tallies, even in the presence of potentially malicious users. In this experiment the adversary controls malicious users who can interact with S_1 via a MalReport oracle and compel other honest users to make reports of its choosing via an HonReport oracle. The adversary sees the ciphertexts that result from any of these interactions and can choose the set S of ciphertexts that are eventually sent to S_2. This set could include some subset of the ciphertexts outputs by oracle queries or new ciphertexts of the adversary's choosing. This experiment conservatively models a group of malicious users with strong control over the network between S_1 and S_2.

The adversary wins the report uniqueness experiment if, after reports by honest users are subtracted from the total report tally, 1) there are more total tallies left than the adversary made calls to MalReport or 2) there is any rep that has more tallies than there are distinct malicious users, as counted by the number of distinct public keys used with the MalReport oracle. The former situation implies that the adversary was able to produce new report tallies without interacting with S_1, and the latter situation implies that the adversary was able to thwart the scheme's duplicate tally prevention mechanism.

Our report uniqueness definition implies stronger protection for message senders than is available in FACTS [35]. FACTS does not strictly prevent malicious users from submitting multiple reports for the same message, relying instead on out-of-protocol throttling on the number of reports a user can make to ensure that no malicious users can affect a message's tally by too much. Report uniqueness requires that no malicious user can contribute more than one report to the tally for a given report.

Definition 5.3 (Report Uniqueness)**.** We define the report uniqueness experiment $\mathsf{RUNIQ}[\mathcal{A}, \Pi, \lambda, Q_O]$ with respect to a stateful adversary $\mathcal{A}$, a list of numbers Q_O setting upper limits on the number of queries $\mathcal{A}$ makes to each of its oracles, a two-server anonymous tally scheme Π, and a security parameter λ. The experiment is described in Figure 5.

We define the *report uniqueness advantage* of $\mathcal{A}$ as

$$\mathsf{RUNIQAdv}(\mathcal{A}, \Pi, \lambda, Q_O) = \Pr\big[\mathsf{RUNIQ}[\mathcal{A}, \Pi, \lambda, Q_O] = 1\big]$$

and we say that the scheme Π satisfies *report uniqueness* if for all efficient adversaries $\mathcal{A}$ and security parameters $\lambda \in \mathbb{N}$, it holds that

$$\mathsf{RUNIQAdv}(\mathcal{A}, \Pi, \lambda, Q_O) \leq \mathsf{negl}(\lambda).$$

5.5 Threshold Unforgeability

Threshold unforgeability prevents a malicious S_2 from fraudulently convincing S_1 that a threshold number of reports have been received. The adversary in this experiment controls a malicious S_2 who can create honest users and compel them to make reports of messages of its choosing via an HonReport oracle. The adversary receives all the resulting ciphertexts and can attempt to fool S_1 into accepting an incorrect π_v proof via a Verify oracle. The adversary wins the experiment if it can cause S_1 to accept a proof π_v for a report rep where the threshold thresh is larger than the number of times rep has been reported. The experiment does not allow the adversary to control malicious users for bookkeeping reasons: allowing adversary-controlled users to make reports hides the rep being sent to S_1 and makes it impossible to do the necessary record keeping to determine whether the adversary has won the experiment.

Definition 5.4 (Threshold Unforgeability)**.** We define the threshold unforgeability experiment $\mathsf{THFORG}[\mathcal{A}, \Pi, \lambda, Q_O]$ with respect to a stateful adversary $\mathcal{A}$, a list of numbers Q_O setting upper limits on the number of queries $\mathcal{A}$ makes to each of its oracles, a two-server anonymous tally scheme Π, and a security parameter λ. The experiment is described in Figure 6. While not explicitly included in

$\underline{\mathsf{RUNIQ}[\mathcal{A},\Pi,\lambda]:}$
$(\mathsf{pk}_1,\mathsf{sk}_1,\mathsf{sk}_s) \leftarrow \mathsf{SKGen1}(1^\lambda,\mathsf{pp})$
$(\mathsf{pk}_2,\mathsf{sk}_2) \leftarrow \mathsf{SKGen2}(1^\lambda,\mathsf{pp})$
$U \leftarrow \{\}; T \leftarrow \{\}; S \leftarrow \{\}; M \leftarrow \{\}$
$\mathsf{win} \leftarrow 0; \mathsf{count} \leftarrow 0$
$\mathcal{A}^{O}(1^\lambda)$
output win

$\underline{\mathsf{MalReport}(\mathsf{pk}_u):}$
if $(\cdot,\mathsf{pk}_u) \in U:$ output $\bot$
if $\mathsf{pk}_u \notin M:\ M \leftarrow M \cup \{\mathsf{pk}_u\}$
$\langle \mathcal{A}, S_1(\mathsf{pk}_1,\mathsf{sk}_s,\mathsf{sk}_1,\mathsf{pk}_u)\rangle$
$\mathsf{count} \leftarrow \mathsf{count}+1$

$\underline{\mathsf{Submit}(\mathsf{ct}):}$
$S \leftarrow S \cup \{\mathsf{ct}\}$

$\underline{\mathsf{AddHonUser}(\mathsf{uid}):}$
if $\mathsf{uid} \in U:$ output $\bot$
$\mathsf{pk}_u,\mathsf{sk}_u \leftarrow \mathsf{UKGen}(1^\lambda,\mathsf{pp})$
$U[\mathsf{uid}] \leftarrow (\mathsf{sk}_u,\mathsf{pk}_u)$
output pk_u

$\underline{\mathsf{HonReport}(\mathsf{uid},\mathsf{rep},\mathsf{rd}):}$
if $\mathsf{uid} \notin U:$ output $\bot$
$(\mathsf{sk}_u,\mathsf{pk}_u) \leftarrow U[\mathsf{uid}]$
$\mathsf{ct} \leftarrow \langle U(\mathsf{rep},\mathsf{rd},\mathsf{pk}_u,\mathsf{sk}_u,\mathsf{pk}_1,\mathsf{pk}_2),$
$\quad S_1(\mathsf{pk}_1,\mathsf{sk}_s,\mathsf{sk}_1,\mathsf{pk}_u)\rangle$
$T \leftarrow T \cup \{\mathsf{ct}\}$
output ct

$\underline{\mathsf{Process}():}$
$R \leftarrow \{\}; \mathsf{HonR} \leftarrow \{\}; D \leftarrow \{\}$
for $\mathsf{ct} \in S:$
$\quad (\mathsf{rep},\mathsf{dupTag},\mathsf{hd}) \leftarrow \mathsf{Verify}(\mathsf{sk}_s,\mathsf{sk}_2,\mathsf{ct})$
$\quad$ if $(\mathsf{rep},\mathsf{dupTag},\mathsf{hd}) = \bot:$ continue
$\quad$ if $(\mathsf{rep},\mathsf{dupTag}) \notin D:$
$\quad\quad D \leftarrow D \cup \{(\mathsf{rep},\mathsf{dupTag})\}$
$\quad\quad R[\mathsf{rep}] \leftarrow R[\mathsf{rep}]+1$
$\quad\quad$ if $\mathsf{ct} \in T:$
$\quad\quad\quad \mathsf{HonR}[\mathsf{rep}] \leftarrow \mathsf{HonR}[\mathsf{rep}]+1$
$\mathsf{count}' \leftarrow 0$
for $\mathsf{rep} \in R:$
$\quad \mathsf{diff} \leftarrow R[\mathsf{rep}] - \mathsf{HonR}[\mathsf{rep}]$
$\quad$ if $\mathsf{diff} > |M|:\ \mathsf{win} \leftarrow 1$
$\quad \mathsf{count}' \leftarrow \mathsf{count}' + \mathsf{diff}$
if $\mathsf{count}' > \mathsf{count}:\ \mathsf{win} \leftarrow 1$
output R

Figure 5: Report uniqueness experiment RUNIQ **(Definition 5.3).**

$\underline{\mathsf{THFORG}[\mathcal{A},\Pi,\lambda]:}$
$(\mathsf{pk}_1,\mathsf{sk}_1,\bot) \leftarrow \mathsf{SKGen1}(1^\lambda,\mathsf{pp})$
$(\mathsf{pk}_2,\mathsf{sk}_2,\mathsf{sk}_s) \leftarrow \mathcal{A}(1^\lambda,\mathsf{pk}_1)$
$U \leftarrow \{\}; R \leftarrow \{\}$
$\mathsf{win} \leftarrow 0$
$\mathcal{A}^{O}(1^\lambda)$
output win

$\underline{\mathsf{AddHonUser}(\mathsf{uid}):}$
if $\mathsf{uid} \in U:$ output $\bot$
$\mathsf{pk}_u,\mathsf{sk}_u \leftarrow \mathsf{UKGen}(1^\lambda,\mathsf{pp})$
$U[\mathsf{uid}] \leftarrow (\mathsf{sk}_u,\mathsf{pk}_u)$
output pk_u

$\underline{\mathsf{HonReport}(\mathsf{uid},\mathsf{rep},\mathsf{rd}):}$
if $\mathsf{uid} \notin U:$ output $\bot$
$(\mathsf{sk}_u,\mathsf{pk}_u) \leftarrow U[\mathsf{uid}]$
$\mathsf{ct} \leftarrow \langle U(\mathsf{rep},\mathsf{rd},\mathsf{pk}_u,\mathsf{sk}_u,\mathsf{pk}_1,\mathsf{pk}_2),$
$\quad S_1(\mathsf{pk}_1,\mathsf{sk}_s,\mathsf{sk}_1,\mathsf{pk}_u)\rangle$
$R[\mathsf{rep}] \leftarrow R[\mathsf{rep}]+1$
output ct

$\underline{\mathsf{Verify}(\mathsf{rep},\mathbf{dupTags},\pi_v):}$
//get number of clauses in proof
$\mathsf{thresh} \leftarrow |\pi_v|$
$\mathsf{ver} \leftarrow \mathsf{S1Verify}(\mathsf{rep},\mathbf{vData},\mathbf{dupTags},\pi_v)$
if $\mathsf{thresh} > \mathsf{R}[\mathsf{rep}] \ \wedge\ \mathsf{ver} = 1:$
$\quad \mathsf{win} \leftarrow 1$

Figure 6: Threshold unforgeability experiment THFORG **(Definition 5.4).**

the description, we assume that the experiment retains the relevant transcript data from S_1 in the Report protocol in order to produce **vData** for S1Verify.

We define the *threshold unforgeability advantage* of $\mathcal{A}$ as

$$\mathsf{THFORGAdv}(\mathcal{A},\Pi,\lambda,Q_O) = \Pr\big[\mathsf{THFORG}[\mathcal{A},\Pi,\lambda,Q_O] = 1\big],$$

and we say that the scheme Π satisfies *threshold unforgeability* if for all efficient adversaries $\mathcal{A}$ and security parameters $\lambda \in \mathbb{N}$, it holds that

$$\mathsf{THFORGAdv}(\mathcal{A},\Pi,\lambda,Q_O) \le \mathsf{negl}(\lambda).$$

5.6 Deniability

The majority of the deniability needs for threshold source tracking are handled by the deniability of the underlying source tracking scheme. That said, deniability can be a valuable property for anonymous tallies as well. Deniability in an anonymous tally used for threshold source tracking means that the individual reports made toward reaching the source tracking threshold can be denied.

Deniability requires that even if user or server secrets are made public, reports cannot be verifiably tied back to a given user. Specifically, we will consider two kinds of deniability.

(1) *Server compromise deniability*: even if all the server secrets $\mathsf{pk}_1,\mathsf{pk}_2,\mathsf{sk}_1,\mathsf{sk}_2,\mathsf{sk}_s$ are made public, there should exist a Forge_{SC} algorithm that, given a user uid's public key pk_u and the leaked secrets, generates a report that is indistinguishable from a real report made by user uid.
(2) *User compromise deniability*: even if a user's secret sk_u is made public, there should exist a Forge_{UC} algorithm that, given a user uid's public key pk_u and leaked secret key sk_u, generates a report ct and decrypted (rep, dupTag) that are indistinguishable from a real report made by user uid.

We do not formalize these definitions, but we will require them from our scheme and will discuss how we achieve them.

6 TWO-SERVER ANONYMOUS TALLY

This section describes our main construction of a two-server anonymous tally scheme.

6.1 Building Up the Construction

A simple scheme. We begin with a scheme that satisfies our correctness requirements but fails to achieve our security goals and ignores the report data rd. As explained previously, the Report procedure begins with rep, which can be a hash of the original report contents from the source tracking scheme. In the anonymous tally scheme, the user samples randomness $r \xleftarrow{R} \mathbb{Z}_q$ and sends $w \leftarrow \mathsf{rep} + r \in \mathbb{Z}_q$ and uid to the server S_1. S_1 computes and returns a MAC $\sigma \leftarrow \mathsf{MAC.Sign}(\mathsf{sk}_s, (w, \mathsf{uid}))$. The user encrypts $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, (\mathsf{rep}, \sigma, r, \mathsf{uid}))$ as the output of the Report protocol. In Verify, S_2 decrypts this message, verifies the MAC tag σ, and sets $\mathsf{dupTag} \leftarrow (\mathsf{rep}, \mathsf{uid})$.

This scheme satisfies correctness because each user's report results in a distinct dupTag. Unfortunately, while the Report protocol does not reveal anything about rep to S_1, it fails to satisfy other security goals. In particular,

(1) A single user can lie about its value of uid, allowing it to submit the same rep multiple times, breaking report uniqueness.
(2) It fully reveals the identity of the user uid to S_2, failing to achieve reporter anonymity.

The solutions to these two problems seem to pull in different directions, forcing users to always use the same uid to protect report uniqueness while trying to hide uid for reporter anonymity. We show to achieve both properties together.

Adding report uniqueness. In order to add report uniqueness, we need users to always send the same uid and make sure that no malicious user can use another user's uid to submit a report. We will accomplish this by having each user select a secret key $\mathsf{sk}_u \xleftarrow{R} \mathbb{Z}_q$ and setting $\mathsf{pk}_u \leftarrow g^{\mathsf{sk}_u}$. We will have pk_u be tied to the user id uid, where g is a generator of a prime order group G, $|G| = q$. Users now compute w as $w \leftarrow H(\mathsf{rep})^r$ (so r still masks $H(\mathsf{rep})$), and instead of sending (w, uid) to S_1, they send (w, v) where $v \leftarrow w^{\mathsf{sk}_u}$. Users also sends a proof of knowledge of sk_u to demonstrate that they know the secret key being used. Verifying this proof gives S_1 confidence that a user is not assuming another user's identity to submit duplicate reports.

We can build the proof system needed to prove knowledge of sk_u using a Chaum-Pedersen proof [15] made non-interactive in the random oracle model [9, 24]. This proof allows the user to prove that it knows the secret sk_u such that $w = H(\mathsf{rep})^r, \mathsf{pk}_u = g^{\mathsf{sk}_u}, v = w^{\mathsf{sk}_u}$ form a DH tuple [19]. We denote proofs using the notation of Camenisch and Stadler [13], where $\mathsf{PoK}\{(\mathsf{sk}_u), \mathsf{pk}_u = g_u^{\mathsf{sk}}, v = w^{\mathsf{sk}_u}\}$ represents the Chaum-Pedersen proof, and require the standard zero knowledge and knowledge extraction properties [12].

The work of S_2 changes very little in this version of the protocol. The ciphertext output by the user consists of the same plaintext contents $(\mathsf{rep}, \sigma, r, \mathsf{uid})$, and S_2 only needs to change how it calculates w to match the updated scheme.

The addition of a user secret and proof requirement means that a malicious user cannot lie about its identity to S_2 and will therefore always have the same dupTag for the same message, ensuring report uniqueness.

Adding reporter anonymity. Next, we add reporter anonymity. The challenge of reporter anonymity is to replace the tag uid with a tag t unique to each user for each message. This tag must be user-dependent and deterministic, but must be unlinkable to uid. To prevent S_2 from identifying which set of reports have come from the same user, the tag t must depend on both the identity of the user and the content of rep.

Our solution is to have the server compute t as a PRF evaluation of the user's identity and the report rep. The challenge is to do this without revealing rep to S_1. Our final scheme has S_1 compute t by evaluating an oblivious PRF (OPRF) [26, 40] evaluation on v using the secret key sk_1, resulting in a tag $t = v^{\mathsf{sk}_1} = w^{\mathsf{sk}_u \mathsf{sk}_1}$. As before, the server S_1 learns nothing about rep because $H(\mathsf{rep})$ is masked by r. Instead of computing $\sigma \leftarrow \mathsf{MAC.Sign}(\mathsf{sk}_1, (w, \mathsf{uid}))$, S_1 sets $\sigma \leftarrow \mathsf{MAC.Sign}(\mathsf{sk}_1, (w, t))$. The tag t now depends on all three of rep, sk_u, and sk_1. To ensure that the server S_1 does not misbehave, it also sends a Chaum-Pedersen proof that it has honestly computed t.

At the end of the Report protocol, the user sets its output to $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, (\mathsf{rep}, t, \sigma, r))$. When running Verify, S_2 now computes dupTag as $\mathsf{dupTag} \leftarrow t^{1/r}$, resulting in t being a deterministic function of rep, sk_u, and sk_1:

$$t^{1/r} = H(\mathsf{rep})^{r \mathsf{sk}_u \mathsf{sk}_1 / r} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_1}.$$

As intended, the dupTag now depends on the user and the message. Assuming that the DDH problem is hard in G, $H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_1}$ will appear uniformly randomly distributed in G, meaning that the dupTag reveals nothing about uid to the server S_2. Including the server key sk_1 in the exponent in t, while not strictly necessary for the anonymity property, serves to ensure deniability, as we will discuss below.

Adding verification of S_2. As specified in Section 4, the platform itself will host S_1, allowing for internal audits and monitoring, while S_2 is hosted by a third party. We now briefly describe a protocol that allows the platform to verify claims from S_2 that a certain rep has exceeded a given threshold thresh, without revealing which users' reports contributed to the threshold.

A naïve and insecure way for S_2 to prove to S_1 that users have in fact sent thresh distinct instances of a particular report rep is for S_2 to reveal the $(\mathsf{rep}, r, \mathsf{dupTag})$ tuples for each report. Using this information, S_1 (who must keep the values w and t that it receives in the Report protocol) can check if it previously saw values of $w = H(\mathsf{rep})^r$ and $t = \mathsf{dupTag}^r$. Due to the collision-resistance of H and the hardness of discrete log in G, S_2 will be unable to forge such reports, and the distinct dupTag values mean that S_2 is sending reports from distinct users. Unfortunately, directly revealing these values to S_1 allows linking which user made which report, which would break report confidentiality.

In order to go from the naïve solution to one that preserves report confidentiality, we modify the protocol so that S_2 proves to S_1 in zero knowledge that it knows reports that satisfy the relationships above, without revealing which reports they are. Instead of directly revealing rep, r, and dupTag for each report, S_2 reveals only rep and dupTag, neither of which will have previously been seen by S_1. Then, it proves in zero knowledge that it knows the value r such that $H(\mathsf{rep})^r = w$ and $\mathsf{dupTag}^r = t$ for *some* (w, t) held by S_1. This proof is a standard OR-composition of Chaum-Pedersen proofs. This OR proof is repeated for each of the thresh values of dupTag. Thus S_2 can convince S_1 that the reports it has sent includes thresh distinct reports of rep without revealing which clients' interactions with S_1 produced those reports.

More precisely, for a report rep, threshold thresh and a batch of reports of size s, S_2 holds a vector $(r_1, \ldots, r_{\mathsf{thresh}})$ and length-s vectors $\mathbf{w}$, $\mathbf{t}$, and **dupTag**. We prove the statement

$$\phi = \phi_1 \wedge \ldots \wedge \phi_{\mathsf{thresh}},$$

where ϕ_i is defined as

$$\begin{aligned}\phi_i = \{&\big(H(\mathsf{rep})^{r_i} = w_1 \;\wedge\; \mathsf{dupTag}_1^{r_i} = t_1\big)\\ &\vee \ldots.\\ &\vee \big(H(\mathsf{rep})^{r_i} = w_s \;\wedge\; \mathsf{dupTag}_s^{r_i} = t_s\big)\}.\end{aligned}$$

Our verification proof requires time and space $O(s \cdot \mathsf{thresh})$. This scheme allows for a privacy/performance tradeoff where the batch size s is reduced to only subset of reports, thereby reducing the anonymity set of each user whose report is included, but speeding up and shrinking the communication required of the verification process.

Supporting report data. Finally, we complete the scheme by adding support for including report data rd in a report. This is achieved by simply having the user making a report encrypt rd under a public key $\mathsf{pk}_{1\mathsf{rd}}$ held by S_1 and include the corresponding ciphertext hd as part of the plaintext encrypted to produce ct. Thus S_2 does not learn anything from hd when it decrypts ct, but when S_1 runs Reveal, it decrypts hd to recover rd. In our full scheme, the keys sk_1 and pk_1 are split into two parts: $\mathsf{sk}_{1\mathsf{rep}}, \mathsf{pk}_{1\mathsf{rep}}$ which are used for reporting as described thus far, and $\mathsf{sk}_{1\mathsf{rd}}, \mathsf{pk}_{1\mathsf{rd}}$ which are used for encrypting and decrypting report data. Since each report comes with its own copy of hd, S_1 should check that any expected relationship between the decrypted message and the report rep are satisfied, e.g., it should check that $\mathsf{rd} = H(\mathsf{rep})$.

6.2 Full Construction

We now formalize the construction described informally above.

Construction 6.1 (Two-server anonymous tally scheme)**.** Our two-server anonymous tally scheme Π, shown in Figure 7, is defined with respect to a cyclic group G of prime order q with generator $g \in G$ where DDH is hard. The scheme uses the following tools:

- A CCA-secure public key encryption scheme PKE = (KGen, Enc, Dec)
- An existentially unforgeable MAC scheme MAC = (Sign, Verify)
- A hash function $H : \mathcal{R} \to G$ modeled as a random oracle
- A non-interactive zero-knowledge proof of knowledge (NIZKPoK) scheme for Diffie-Hellman triples

6.3 Security Analysis

We now briefly discuss each security property and state the theorems that we prove in Appendix C.

The correctness of the scheme follows largely from the correctness of the underlying cryptographic tools. There is a possibility of distinct honest users having duplicate dupTags if either two reps happen to collide in H or if two users happen have the same sk_u. These events occur with negligible probability in the size of G.

Intuitively, report confidentiality follows from the fact that the value of rep is masked by r when sent to S_1 and encrypted when the adversary sees it and decides whether or not to give it to S_2. However, we also need to make sure that S_1 cannot use the output of the Process oracle to distinguish which messages are being reported. The report confidentiality experiment prevents S_1 from using the output of Process to achieve trivial wins, but we also need to show that S_1 cannot cleverly circumvent these measures.

The proof proceeds by a series of hybrids that first extract the secret $\mathsf{sk}_{1\mathsf{rep}}$ used by the adversary before carefully converting everything in the experiment that depends on the choice of b into a random value, simulated proof, or encryption of 0. A probability argument can then show that an adversary cannot succeed in using Process in a way that circumvents protections against trivial wins.

Theorem 6.2 (Report confidentiality)**.** *Assuming that the encryption scheme* (Enc, Dec) *is CCA-secure, that the proof system* PoK *is a zero knowledge proof of knowledge, that the DDH problem is hard in the group G, and that the hash function H is modeled as a random oracle, then our two-server anonymous tally scheme (Construction 6.1) satisfies report confidentiality (Definition 5.1).*

Specifically, for every report uniqueness adversary $\mathcal{A}$ that attacks our scheme Π and list Q_O specifying the number of queries $\mathcal{A}$ makes to each of its oracles, there exist adversaries against the tools used to build the scheme such that for every λ (omitting adversary names and security parameters),

$$\begin{aligned}&\mathsf{RCONFAdv}(\mathcal{A}, \Pi, \lambda, Q_O)\\ &\qquad\leq 2Q_{\mathsf{Report}}(\mathsf{PoKAdv}(\mathsf{PoK}) + Q_{\mathsf{Process}}\mathsf{ZKAdv}(\mathsf{PoK}))\\ &\qquad\quad + 2\mathsf{CCAAdv}(\mathsf{PKE}) + 6\mathsf{DDHAdv}(G) + \mathsf{negl}.\end{aligned}$$

Reporter anonymity follows almost immediately from the hardness of DDH in G. Since the reporter anonymity adversary controls S_2, the only element of the adversary's view that depends on a reporting user's identity is the value $t = H(\mathsf{rep})^{r \mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$, from which S_2 derives $\mathsf{dupTag} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$. Intuitively, the adversary should not be able to distinguish between $(H(\mathsf{rep}), \mathsf{pk}_u, \mathsf{dupTag})$ and $(H(\mathsf{rep}), \mathsf{pk}_u, R)$ for $R \xleftarrow{R} G$. The proof formalizes this via a reduction to DDH. Additionally, the fact that the report data rd is encrypted under the public key of S_1 means that the adversary cannot learn anything from hd.

Theorem 6.3 (Reporter anonymity)**.** *Assuming that* PoK *has perfect completeness, that the DDH problem is hard in the group G, that the encryption scheme* (Enc, Dec) *is CPA-secure, and that the hash function H is modeled as a random oracle, then our two-server anonymous tally scheme (Construction 6.1) satisfies reporter anonymity (Definition 5.2).*

Specifically, for every reporter anonymity adversary $\mathcal{A}$ that attacks our scheme Π, there exist DDH and CPA adversaries $\mathcal{B}$ and C such that for every λ,

$$\begin{aligned}&\mathsf{ANONAdv}(\mathcal{A}, \Pi, \lambda)\\ &\qquad\leq 2 \cdot \mathsf{DDHAdv}(\mathcal{B}, G, \lambda) + 2 \cdot \mathsf{CPAAdv}(C, \mathsf{PKE}).\end{aligned}$$

For report uniqueness, we show that an adversary who cannot break our scheme's underlying primitives needs to roughly "follow the rules" in the report uniqueness game, meaning the adversary has no opportunities to deviate from the protocol and cause incorrect outcomes. The only degrees of freedom afforded to an adversary are its choices of reports and randomness r for each report. We show, via the hardness of discrete logarithm in G, that the adversary

$\underline{\mathsf{SKGen1}(1^\lambda, \mathsf{pp}):}$
$\mathsf{sk}_{1\mathsf{rep}} \xleftarrow{R} \mathbb{Z}_q$
$\mathsf{pk}_{1\mathsf{rep}} \leftarrow g^{\mathsf{sk}_{1\mathsf{rep}}}$
$(\mathsf{sk}_{1\mathsf{rd}}, \mathsf{pk}_{1\mathsf{rd}}) \leftarrow \mathsf{PKE.KGen}(1^\lambda)$
$\mathsf{sk}_s \xleftarrow{R} \{0,1\}^\lambda$
$\mathsf{sk}_1 \leftarrow (\mathsf{sk}_{1\mathsf{rep}}, \mathsf{sk}_{1\mathsf{rd}})$
$\mathsf{pk}_1 \leftarrow (\mathsf{pk}_{1\mathsf{rep}}, \mathsf{pk}_{1\mathsf{rd}})$
output $(\mathsf{pk}_1, \mathsf{sk}_1, \mathsf{sk}_s)$

$\underline{\mathsf{Verify}(\mathsf{sk}_s, \mathsf{sk}_2, \mathsf{ct}):}$
$(\mathsf{rep}, t, \sigma, r, \mathsf{hd}) \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_2, \mathsf{ct})$
$w' \leftarrow H(\mathsf{rep})^r$
if $\mathsf{MAC.Ver}(\mathsf{sk}_s, (w', t), \sigma) = 0$:
  output $\perp$
$\mathsf{dupTag} \leftarrow t^{1/r}$
output $(\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})$

$\underline{\mathsf{Reveal}(\mathsf{sk}_1, \mathsf{hd}):}$
$(\mathsf{sk}_{1\mathsf{rep}}, \mathsf{sk}_{1\mathsf{rd}}) \leftarrow \mathsf{sk}_1$
$\mathsf{rd} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_{1\mathsf{rd}}, \mathsf{hd})$
output rd

$\underline{\mathsf{SKGen2}(1^\lambda, \mathsf{pp}):}$
$(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{PKE.KGen}(1^\lambda)$
output $(\mathsf{pk}_2, \mathsf{sk}_2)$

$\underline{\mathsf{UKGen}(1^\lambda, \mathsf{pp}):}$
$\mathsf{sk}_u \xleftarrow{R} \mathbb{Z}_q$
$\mathsf{pk}_u \leftarrow g^{\mathsf{sk}_u}$
output $(\mathsf{pk}_u, \mathsf{sk}_u)$

$\underline{\mathsf{S2Prove}(\mathsf{rep}, \mathbf{pData}, \mathsf{thresh}):}$
$(\mathbf{r}, \mathbf{dupTag}, \mathbf{w}, \mathbf{t}) \leftarrow \mathbf{pData}$
$\pi_v \leftarrow \mathsf{PoK}\{(\mathbf{r}), \phi\}$ //ϕ defined in text
output π_v

$\underline{\mathsf{S1Verify}(\mathsf{rep}, \mathbf{vData}, \mathbf{dupTags}, \pi_v):}$
$(\mathbf{w}, \mathbf{t}) \leftarrow \mathbf{vData}$
if $\mathsf{PoK.Ver}(\pi_v, (\mathbf{dupTags}, \mathbf{w}, \mathbf{t})) = 0$:
  output 0
if $|\mathsf{set}(\mathbf{dupTags})| < |\mathbf{dupTags}|$:
  output 0 //duplicate tags
else output 1

Report :

$\underline{U(\mathsf{rep}, \mathsf{rd}, \mathsf{pk}_u, \mathsf{sk}_u, \mathsf{pk}_1, \mathsf{pk}_2)}$
$(\mathsf{pk}_{1\mathsf{rep}}, \mathsf{pk}_{1\mathsf{rd}}) \leftarrow \mathsf{pk}_1$
$r \xleftarrow{R} \mathbb{Z}_q$
$w \leftarrow H(\mathsf{rep})^r$
$v \leftarrow w^{\mathsf{sk}_u}$
$\pi_u \leftarrow \mathsf{PoK}\{(\mathsf{sk}_u), \mathsf{pk}_u = g^{\mathsf{sk}_u}, v = w^{\mathsf{sk}_u}\}$

$\xrightarrow{w, v, \pi_u}$

$\underline{S_1(\mathsf{pk}_1, \mathsf{sk}_s, \mathsf{sk}_1, \mathsf{pk}_u)}$
$(\mathsf{sk}_{1\mathsf{rep}}, \mathsf{sk}_{1\mathsf{rd}}) \leftarrow \mathsf{sk}_1$
$(\mathsf{pk}_{1\mathsf{rep}}, \mathsf{pk}_{1\mathsf{rd}}) \leftarrow \mathsf{pk}_1$
if $\mathsf{PoK.Ver}(\pi_u, (\mathsf{pk}_u, w, v)) = 0$:
  output $\perp$
$t \leftarrow v^{\mathsf{sk}_{1\mathsf{rep}}}$
$\pi_s \leftarrow \mathsf{PoK}\{(\mathsf{sk}_{1\mathsf{rep}}), \mathsf{pk}_{1\mathsf{rep}} = g^{\mathsf{sk}_{1\mathsf{rep}}}, t = v^{\mathsf{sk}_{1\mathsf{rep}}}\}$
$\sigma \leftarrow \mathsf{MAC.Sign}(\mathsf{sk}_s, (w, t))$

$\xleftarrow{t, \sigma, \pi_s}$

if $\mathsf{PoK.Ver}(\pi_s, (\mathsf{pk}_{1\mathsf{rep}}, v, t)) = 0$:
  output $\perp$
$\mathsf{hd} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{1\mathsf{rd}}, \mathsf{rd})$
$\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_2, (\mathsf{rep}, t, \sigma, r, \mathsf{hd}))$
output ct

Figure 7: Our two-server anonymous tally scheme (Construction 6.1).

cannot pick reports and corresponding randomnesses that lead to colliding values of dupTag for different users.

Theorem 6.4 (Report uniqueness). *Assuming that* MAC *is an existentially unforgeable MAC scheme, that the non-interactive proof system* PoK *satisfies soundness and zero knowledge, that the encryption scheme* (Enc, Dec) *is CCA-secure, that the discrete logarithm problem is hard in the group G, and that the hash function H is modeled as a random oracle, then our two-server anonymous tally scheme (Construction 6.1) satisfies report uniqueness (Definition 5.3).*

Specifically, for every report uniqueness adversary $\mathcal{A}$ that attacks our scheme Π and list Q_O specifying the number of queries $\mathcal{A}$ makes to each of its oracles, there exist adversaries against the tools used to build the scheme such that for every λ (omitting adversary names and security parameters),

$$\begin{aligned}\mathsf{RUNIQAdv}(\mathcal{A}, \Pi, \lambda, Q_O) \leq\ & Q_{\mathsf{MalReport}} \cdot \mathsf{PoKAdv}(\mathsf{PoK}) \\ & + \mathsf{CCAAdv}(\mathsf{PKE}) + \mathsf{MACAdv}(\mathsf{MAC}) \\ & + Q_H \cdot \mathsf{DLAdv}(G) + \mathsf{negl}.\end{aligned}$$

Threshold unforgeability follows directly from the extractability of the zero knowledge proof and the hardness of discrete logarithm in G. If S_2 can produce a false proof that there are more reports of some rep than have actually been made, it must break a discrete logarithm to pretend some report was for a different message than it really was.

Theorem 6.5 (Threshold unforgeability). *Assuming that* PoK *is a proof of knowledge, that the discrete logarithm problem is hard in the group G, and that the hash function H is modeled as a random oracle, then our two-server anonymous tally scheme (Construction 6.1) satisfies threshold unforgeability (Definition 5.4).*

Specifically, for every threshold unforgeability adversary $\mathcal{A}$ that attacks our scheme Π and list Q_O specifying the number of queries $\mathcal{A}$ makes to each of its oracles, there exist adversaries against the tools used to build the scheme such that for every λ (omitting adversary names and security parameters),

$$\begin{aligned}&\mathsf{THFORGAdv}(\mathcal{A}, \Pi, \lambda, Q_O) \\ &\qquad \leq Q_{\mathsf{Verify}} \cdot \mathsf{PoKAdv}(\mathsf{PoK}) + 2Q_H \cdot \mathsf{DLAdv}(G) + \mathsf{negl}.\end{aligned}$$

Finally, we turn our attention to deniability. Recall that we want two kinds of deniability: user compromise deniability and server compromise deniability.

In server compromise deniability, all the server secret keys $\mathsf{sk}_1 = (\mathsf{sk}_{1\mathsf{rep}}, \mathsf{sk}_{1\mathsf{rd}}), \mathsf{sk}_2, \mathsf{sk}_s$ are made public, and we wish to ensure that no $(\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})$ verifiably ties a report to a particular user uid. This is accomplished by showing that there exists an algorithm Forge_{SC} whose outputs are distributed indistinguishably from a real $(\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})$ for a report from the user uid. Since hd is an encryption of rd under $\mathsf{sk}_{1\mathsf{rd}}$, we need for the contents of rd to be deniable via a forgery algorithm $\mathsf{Forge}_{\mathsf{rd}}$ that outputs a forged string rd^*. Such an algorithm exists for source tracking schemes discussed in this paper, such as that of Peale et al. [42]. The Forge_{SC} algorithm outputs $(\mathsf{rep}, R, \mathsf{PKE.Enc}(\mathsf{pk}_{1\mathsf{rd}}, \mathsf{rd}^*))$ for $R \xleftarrow{R} G$. As we did in the proof of reporter anonymity, we can show, via a reduction to the DDH problem in G, that the distribution of $(H(\mathsf{rep}), \mathsf{pk}_u, \mathsf{dupTag})$ is indistinguishable from that of $(H(\mathsf{rep}), \mathsf{pk}_u, R)$ as long as sk_u remains secret (which is enforced by the zero-knowledge property of the proof π_u).

In user compromise deniability, the keys $(\mathsf{sk}_u, \mathsf{pk}_u)$ of a user uid are made public, and we wish to ensure that no $(\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})$ verifiably ties a report to that user. This is accomplished by showing that there exists an algorithm Forge_{UC} whose outputs are distributed indistinguishably from a real $(\mathsf{rep}, \mathsf{dupTag}, \mathsf{hd})$ for a report from user uid. Similarly to the case of server compromise deniability, this is easily achieved by an algorithm that outputs $(\mathsf{rep}, R, \mathsf{PKE.Enc}(\mathsf{pk}_{1\mathsf{rd}}, 0))$ for $R \xleftarrow{R} G$. Even if a user's sk_u is made public, $\mathsf{dupTag} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$ appears random to S_2. This is because including $\mathsf{sk}_{1\mathsf{rep}}$ in the exponent means we can show that dupTag is indistinguishable from random via DDH not only for a secret sk_u but also a secret $\mathsf{sk}_{1\mathsf{rep}}$. This is proved via a reduction to DDH in G, where we show that the distributions of $(H(\mathsf{rep}), \mathsf{pk}_{1\mathsf{rep}}, \mathsf{dupTag})$ and $(H(\mathsf{rep}), \mathsf{pk}_{1\mathsf{rep}}, R)$ are computationally indistinguishable, as long as $\mathsf{sk}_{1\mathsf{rep}}$ remains secret (which is enforced by the zero-knowledge property of the proof π_s). Likewise, since $\mathsf{sk}_{1\mathsf{rd}}$ remains secret, the encryption of 0 is indistinguishable from the encryption of a real rd. This means that as long as both the user and S_1 are not compromised simultaneously, user reports are deniable.

Cover traffic. To achieve larger anonymity sets, it may be desirable to have clients periodically submit valid random reports in the absence of a user's request to submit a real report. With support for a large message hash reporting space, the reports submitted as cover traffic will look legitimate to S_1, but will not increment the tally for a legitimate message except with negligible probability. Client cover traffic would ensure that S_1 could not guess with any confidence which users reported which message, while also ensuring that S_2 has a sufficiently large anonymity set of messages to cover the tracks of real reporters during the verification protocol.

7 EVALUATION

We implemented our anonymous tally scheme in Rust. Group operations are performed using curve25519 via the curve25519 − dalek library [36]. The Chaum-Pedersen proofs in the protocol were made non-interactive via Fiat-Shamir [24]. We instantiated our MAC scheme with HMAC-SHA256 and our encryption scheme with 2048-bit RSA-OAEP using the rust-openssl implementations [23, 45]. Finally, we instantiate our hash function H with SHA512. Since we only hash fixed-length messages, SHA512 will be indifferentiable from a random oracle in this restricted setting [17, 38].

We evaluated the performance of the implementation by running the protocols with random keys and inputs in at least 1,000 trials, with the Rust Criterion benchmarking library configured to a 95% confidence interval, on an 11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz processor running Ubuntu Linux 20.04.5 LTS. The results in Table 3 were obtained with Criterion configured to a 90% confidence interval with at least 20 runs due to extended runtime. Comparisons to performance of other schemes are made by re-running their performance benchmarks, where the source is available, or comparing to published performance data when not.

Evaluation results. Table 1 shows average runtimes for reporting messages and verifying reports in our scheme. Reports and report verification each take well under 1ms to complete. This remains true even when counting the time to add our scheme on top of a conventional source tracking scheme. Combining our anonymous tally scheme with the tree-linkable source tracking of Peale et al. [41, 42], their faster and more practical scheme, only requires an additional $43\mu s$ of computation to produce and hash a report for a 1KB message to derive the rep value used as an input to our scheme.

The communication overhead to report a message, beyond the size of the message itself, is summarized in Table 2. Reporting a message via our Report protocol requires less than 1KB of communication overhead between the user and the servers. The persistent storage required to hold values of dupTag, w, or t is 32 Bytes each, and the hd scales based on the length of the original message. Users may wish to pad the length of reported messages to some constant size to avoid leaking length information.

To show feasibility for a range of anonymity group sizes, we present our benchmark of the protocol to verify S_2 in Table 3. Each row represents the time to prove and verify that S_2 holds knowledge of a report with a unique dupTag amongst a batch of $s - 1$ other reports; repeating this process thresh times will convince S_1 that the threshold has been met. Our implementation is single-threaded, but proof and verification can be parallelized using a map-reduce structure, yielding times much faster than our implementation.

Our results suggest that the scheme has sufficiently low overheads for deployment. The constant time Report and Verify algorithms and constant 944B of network communication to report a message appear reasonable, particularly when weighing increased user privacy, server enforced report uniqueness, and the fact that overhead for non-reported messages is unaffected in our scheme.

Comparison to FACTS [35]. We compare the performance of our scheme to FACTS, as it is, to our knowledge, the only previously known threshold source tracking scheme, although FACTS only supports approximate threshold source tracking, not exact tallies. FACTS is not an open source project, so we base our comparisons on data available in the FACTS paper.

The runtime of FACTS for their interactive Complain algorithm, used to report messages, is a function of the approximate threshold when messages are to be revealed. Our anonymous tally scheme takes constant time, regardless of the reporting threshold. FACTS operates by having the server and users cooperate to maintain a *cooperative counting Bloom filter* (CCBF), a data structure that requires parameter tuning, predefined epoch intervals, and fixed server storage per epoch to avoid probabilistic contention between users trying to report the same message. None of these are necessary for our scheme. As a result, storage can be dynamically allocated based on demand and report frequency, not on security or correctness considerations. The FACTS construction also requires locking the global storage state while waiting for the client to determine how to update the CCBF. Reports in our scheme can be processed without any locks on global state, so it is possible to replicate both servers in the scheme to scale to a large user base.

	Computation Time
Report (User)	360 μs
Report (Server)	327 μs
Verify	760 μs

Table 1: Time to run the Report **protocol and the** Verify **algorithm in our scheme.**

	Communication
Report (User)	176B
Report (Server)	160B
Encrypted Report	608B
Total to Report a Message	944B

Table 2: Communication costs between user and S_1 during the Report **protocol.**

s	S2Prove	S1Verify
100	14.2 ms	15.7 ms
1,000	142 ms	157 ms
10,000	1.42 s	1.57 s
100,000	14.2 s	15.7 s

Table 3: Time to run the S2Prove **and** S1Verify **algorithms. Need to run** thresh **times to prove the threshold is met.**

In the absence of source code and metrics for the runtime of FACTS, it is difficult to make a direct performance comparison, but the authors' analysis shows that a complaint threshold of 1,000 reports leads to an average runtime dominated by the network latency of the 3 messages passed between the user and server in the Complain algorithm. Thus it is reasonable to assume that FACTS performs with very little computational overhead on both the server and client for large thresholds. The performance cost for lower thresholds, however, is higher. Running FACTS with a threshold of 200 results in an average Complain time of over 400ms – 160ms above the expected network latency. While not quite an apples to apples comparison due to differences in evaluation setups, the computation time for running Complain with this threshold, not including network latency, is over 160× higher than our scheme. Since our anonymous tally scheme includes an additional server-only verification protocol, messages which reach the threshold will pay additional server computation and communication costs to verify the counts before revealing the message, but our benchmarks show that these costs are manageable for large anonymity set sizes, and since the costs are deferred until message reveal time, they are only paid by messages which reach the threshold.

8 CONCLUSION

We have presented a new two-server anonymous tally scheme that can be used to build a threshold source tracking system. The resulting system requires no changes to the message processing or delivery, and only affects the overhead of reporting abusive messages. Compared to prior work, our scheme removes the possibility of false positive message reports and allows for exact report thresholds for revealing messages, not approximate ones.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2234408, as well as gifts from Google and Cisco. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] 2021. Analytics in Exposure Notifications Express: FAQ. https://github.com/google/exposure-notifications-android/blob/master/doc/enexpress-analytics-faq.md. Accessed 5/1/2023.

[2] 2021. Exposure Notification Privacy-preserving Analytics (ENPA) White Paper. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf. Accessed 5/1/2023.

[3] 2021. What is traceability and why does WhatsApp oppose it? https://faq.whatsapp.com/general/security-and-privacy/what-is-traceability-and-why-does-whatsapp-oppose-it/.

[4] Josh Aas and Time Geoghegan. 2020. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/. https://www.abetterinternet.org/post/introducing-prio-services/

[5] Veridiana Alimonti. 2021. Brazil's Fake News Bill: Congress Must Stand Firm on Repealing Dangerous and Disproportionate Surveillance Measures. https://www.eff.org/deeplinks/2021/11/brazils-fake-news-bill-congress-must-stand-firm-repealing-dangerous-and.

[6] Apple. 2021. CSAM Detection: Technical Summary. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf.

[7] Aya Batrawy. 2023. Saudi man sentenced to death for tweets in harshest verdict yet for online critics (NPR). https://www.npr.org/2023/08/31/1196776390/saudi-arabia-man-death-sentence-tweets. Accessed 3/13/2024.

[8] Mihir Bellare and Viet Tung Hoang. 2022. Efficient Schemes for Committing Authenticated Encryption. *IACR Cryptol. ePrint Arch.* (2022).

[9] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS*.

[10] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. 2015. SoK: A comprehensive analysis of game-based ballot privacy definitions. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 499–516.

[11] Abhishek Bhowmick, Dan Boneh, Steve Myers, Kumal Talwar, and Karl Tarbe. 2021. The Apple PSI System. (2021).

[12] Dan Boneh and Victor Shoup. 2020. *A Graduate Course in Applied Cryptography (version 0.5)*. https://cryptobook.us.

[13] Jan Camenisch and Markus Stadler. 1997. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. 410–424.

[14] David Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* 24, 2 (1981), 84–88.

[15] David Chaum and Torben P. Pedersen. 1992. Wallet Databases with Observers. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. 89–105.

[16] Long Chen and Qiang Tang. 2018. People Who Live in Glass Houses Should not Throw Stones: Targeted Opening Message Franking Schemes. *IACR Cryptol. ePrint Arch.* 2018 (2018), 994.

[17] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. 2005. Merkle-Damgård Revisited: How to Construct a Hash Function. In *CRYPTO*.

[18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 164–180.

[19] Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Information Theory* 22, 6 (1976), 644–654.

[20] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. 2018. Fast Message Franking: From Invisible Salamanders to Encryptment. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 155–186.

[21] Steve Englehardt. 2019. Next steps in privacy-preserving Telemetry with Prio. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/

[22] Inc. Facebook. 2017. Messenger Secret Conversations Technical Whitepaper. https://messengernews.fb.com/wp-content/uploads/2018/09/messenger-secret-conversations-technical-whitepaper.pdf.

[23] Steven Fackler. 2022. rust-openssl. https://github.com/sfackler/rust-openssl.

[24] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings (Lecture Notes in Computer Science, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[25] Matthew Franklin and Haibin Zhang. 2013. Unique ring signatures: A practical construction. In *Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17*. Springer, 162–170.

[26] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 303–324.

[27] Eiichiro Fujisaki and Koutarou Suzuki. 2007. Traceable ring signature. In *International Workshop on Public Key Cryptography*. Springer, 181–200.

[28] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. 2017. Message Franking via Committing Authenticated Encryption. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 10403)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, 66–97.

[29] Loïs Huguenin-Dumittan and Iraklis Leontiadis. 2018. A Message Franking Channel. *IACR Cryptol. ePrint Arch.* 2018 (2018), 920.

[30] Rawane Issa, Nicolas Alhaddad, and Mayank Varia. 2022. Hecate: Abuse Reporting in Secure Messengers with Sealed Sender. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 2335–2352.

[31] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. 2021. Outside looking in: Approaches to content moderation in end-to-end encrypted systems. https://cdt.org/insights/outside-looking-in-approaches-to-content-moderation-in-end-to-end-encrypted-systems/

[32] Erin Kenney, Qiang Tang, and Chase Wu. 2022. Anonymous Traceback for End-to-End Encryption. In *European Symposium on Research in Computer Security*. Springer, 42–62.

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

[34] Iraklis Leontiadis and Serge Vaudenay. 2018. Private Message Franking with After Opening Privacy. *IACR Cryptol. ePrint Arch.* 2018 (2018), 938.

[35] Linsheng Liu, Daniel S. Roche, Austin Theriault, and Arkady Yerukhimovich. 2021. Fighting Fake News in Encrypted Messaging with the Fuzzy Anonymous Complaint Tally System (FACTS). *IACR Cryptol. ePrint Arch.* (2021).

[36] IA Lovecruft and Henry de Valence. 2021. curve25519-dalek: A pure-rust implementation of group operations on ristretto and curve25519. https://github.com/dalek-cryptography/curve25519-dalek.

[37] Namrata Maheshwari. 2020. Traceability Under Brazil's Proposed Fake News Law Would Undermine Users' Privacy and Freedom of Expression. https://cdt.org/insights/traceability-under-brazils-proposed-fake-news-law-would-undermine-users-privacy-and-freedom-of-expression/

[38] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. 2004. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC*.

[39] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 120–130.

[40] Moni Naor and Omer Reingold. 2004. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM* 51, 2 (2004), 231–262.

[41] Charlotte Peale. 2021. srctracking. https://github.com/cpeale/srctracking.

[42] Charlotte Peale, Saba Eskandarian, and Dan Boneh. 2021. Secure Source-Tracking for Encrypted Messaging. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS*. ACM.

[43] Katitza Rodriguez. 2021. Why Indian Courts Should Reject Traceability Obligations. https://www.eff.org/deeplinks/2021/06/why-indian-courts-should-reject-traceability-obligations.

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

[45] RustCrypto. 2022. RustCrypto/MACs. https://github.com/RustCrypto/MACs.

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

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

[48] Nirvan Tyagi, Paul Grubbs, Julia Len, Ian Miers, and Thomas Ristenpart. 2019. Asymmetric Message Franking: Content Moderation for Metadata-Private End-to-End Encryption. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 222–250.

[49] Nirvan Tyagi, Ian Miers, and Thomas Ristenpart. 2019. Traceback for End-to-End Encrypted Messaging. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 413–430.

A ADDITIONAL RELATED WORK

Electronic Voting: Electronic voting as a problem space has many parallels to message report aggregation; a set of users wish to contribute towards a common tally without revealing their identity while preventing repeated voting or "ballot-box stuffing". Tools such as *traceable ring signatures* work well in the electronic voting setting, providing anonymity for a voter within a pool as long as they do not attempt to vote twice for the same "issue ID" [27]. There is also significant overlap in the basic goals for the systems, including similar notions of honest voter privacy, such that "an attacker should not notice if the votes of two voters are swapped", as well as *tally uniqueness*, which "ensures that two different tallies for the same [election] cannot be accepted by the verification algorithm, even if all the [voters] in the system are malicious" [10]. It is worth highlighting a few assumptions that can be accommodated in electronic voting that prevent these tools from being applied to solve anonymous report aggregation.

- Traceable ring signatures often require a consistent group of public keys to ensure that votes were authenticated by one of the corresponding private keys and that no signature was used to vote twice on the same issue. In elections, voters can be registered before the election, allowing for consistent signing groups of voters for a given election; the votes are also submitted during a fixed period of time. In an encrypted messaging platform, users can sign up and report forwarded messages at any time, making it difficult to ensure a consistent user group across the aggregated reports of a message.
- Traceable ring signatures provide a Trace procedure that returns whether two signatures are duplicates. To ensure a set of n signatures contains no duplicates, this requires $O(n^2)$ invocations of Trace. The dupTags in our scheme allow for $O(n)$ de-duplication within S_2, while S_2 can prove correctness of the tally to S_1 in $O(k*n)$, with an anonymity group of size k for any given report. In practice, this gives the system more flexibility; it does not limit the anonymity group of users who registered at a similar time, but instead, to any report across the lifetime of the system, while also scaling the anonymity group independently from the reporting threshold.

While this is by no means an exhaustive review of electronic voting literature, we believe it illustrates some key differences in assumptions that can be made when compared to designing a system for report aggregation in encrypted messaging.

Privacy Pass: Our usage of an oblivious PRF, along with a MAC of the output at a specific point, can be viewed as a *verifiable random function*, where a single evaluation can be verified without taking away the randomness of other evaluations [39]. These are used in many other settings, including Privacy Pass, which uses oblivious PRFs to batch generate anonymous tokens for honest users to bypass CAPTCHAs when using Tor or other anonymous traffic systems [18]. Similar to our goals of avoiding the end server learning which user reported the message, their system serves to prevent the end server from learning which user accessed the resource while providing some assurance that the user is not a bot.

B REPORT CONFIDENTIALITY FOR KNOWN REPORT DATA

This appendix considers the impact on the confidentiality of reports in the case where a malicious server already knows an honest user's report data rd, a setting not fully covered by our formal security definitions. We briefly consider the consequences of this for FACTS and for our scheme. Note that the focus here is on the confidentiality of the users reporting the message, not on the sender of the message. We discuss consequences for the sender of the message in Section 1 and Section 4.3.

Consequences for FACTS. The FACTS scheme relies on a "Collaborative Counting Bloom Filter" data structure of multiple overlapping Bloom filters that clients update in the clear. Since each client is only allowed to flip one bit at a time, several clients must report a message before a given message is included in the filter, at which point any client who wishes to report the message knows to tell the server the report data rd when making its report.

A malicious server who learns rd can simply set all the bits for that message, causing any user who wishes to report rd to immediately reveal themself to the server. This is a full break of confidentiality by the server.

Consequences for our scheme. A malicious server S_1 who knows rd in our scheme can still attack report confidentiality, albeit less directly. The combination of masking and zero-knowledge proofs used in the Report protocol ensures that this protocol reveals nothing to a malicious S_1, regardless of whether or not S_1 knows rd. However, the scheme has no check that a given value of t in the encrypted output of Report corresponds to a real user. Thus a malicious server who knows rd can produce many fake reports for the same rd without needing to control multiple malicious users. This can be used to produce a targeted attack on report confidentiality.

Suppose a malicious server receives a report from an honest user U and wishes to check if the report data was some particular rd that the server knows. The server creates $t-1$ additional fake reports for rd and includes all t reports (the honest report from U and $t-1$ fake reports) in a batch of report ciphertexts sent to S_2. If S_2 reveals rd to S_1 as having t reports, S_1 knows that it correctly guessed the report data for user U. Otherwise it can try again with a different candidate report until it guesses correctly. The report confidentiality rules out this attack by checking that the adversary has not sent reports that cause the output of S_2 to differ if $b=0$ or $b=1$, as this kind of attack is only possible when the adversary does know rd in advance.

Our scheme requires S_2 to learn the report data rd for each report, but it has no way of tying this information to a given user. Nonetheless, it's important for S_1 to provide some kind of account verification process to prevent S_2 from producing many fake users and using them to get rd over the threshold. This is yet another way that the security of the scheme is broken if both servers are compromised.

Observe that there is a difference in the kind of compromise that occurs if a server learns rd in our scheme versus in FACTS. In FACTS, this event leads to a complete break in which the adversary can always bypass the threshold and learn every user's report immediately. In our scheme, S_1 can launch a targeted attack on a particular user to guess and check the contents of their reports. However, as long as S_2 doesn't allow for "re-reporting" of messages past the threshold, this attack cannot be scaled to attack every user at once. Thus switching to a two server model in our scheme provides for a more gradual degradation of security properties when the adversary knows rd in advance of receiving reports.

C DEFERRED PROOFS

Proof of Theorem 6.2 (report confidentiality).

Proof. The proof proceeds by a series of indistinguishable hybrids.

- Hyb_0: This hybrid is the security experiment $\mathsf{RCONF}[\mathcal{A}, \Pi, \lambda, Q_O, 0]$.
- Hyb_1: In this hybrid, the experiment runs the extractor guaranteed to exist by the proof of knowledge property of PoK to recover the value $\mathsf{sk}_{1\mathsf{rep}}$ for each proof π_s presented in Report. The experiment outputs $\perp$ should any extractor fail.
  This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the proof system PoK. In particular, the experiment aborts with probability $\mathsf{PoKAdv}(\mathsf{PoK})$, the probability of the extractor failing, for each invocation of the Report oracle. Thus the overall additional failure probability introduced by this change is $Q_{\mathsf{Report}} \cdot \mathsf{PoKAdv}(\mathsf{PoK})$, which remains negligible so long as $\mathsf{PoKAdv}(\mathsf{PoK})$ is negligible.
  Note that the extracted value $\mathsf{sk}_{1\mathsf{rep}}$ will always be the same output because there is a unique $\mathsf{sk}_{1\mathsf{rep}}$ satisfying the statement being proved with respect to $\mathsf{pk}_{1\mathsf{rep}}$.
- Hyb_2: In this hybrid, the experiment replaces the proofs π_v, generated by S2Prove and sent to the adversary during interactions with the Process oracle, with simulated proofs.
  This hybrid is indistinguishable from the preceding one by the zero knowledge property of the proof system PoK. The hybrid consists of at most $Q_{\mathsf{Process}} \cdot Q_{\mathsf{Report}}$ subhybrids (Q_{Report} is an upper bound on the number of proofs produced in each call to the Process oracle), where the ith hybrid replaces the ith proof π_v with a simulated proof. The adversary's advantage in distinguishing between adjacent hybrids is at most $\mathsf{ZKAdv}(\mathsf{PoK})$, so the adversary's advantage in distinguishing between all real versus all simulated proofs is at most $Q_{\mathsf{Process}} \cdot Q_{\mathsf{Report}} \cdot \mathsf{ZKAdv}(\mathsf{PoK})$, which remains negligible so long as $\mathsf{ZKAdv}(\mathsf{PoK})$ is negligible. We omit the standard reduction that formalizes this indistinguishability argument.
- Hyb_3: In this hybrid, the experiment replaces the proofs π_u sent to the adversary during interactions with the Report oracle with simulated proofs.
  This hybrid is indistinguishable from the preceding one by the zero knowledge property of the proof system PoK. The hybrid consists of Q_{Report} subhybrids, where the ith hybrid replaces the ith proof π_u with a simulated proof. The adversary's advantage in distinguishing between adjacent hybrids is at most $\mathsf{ZKAdv}(\mathsf{PoK})$, so the adversary's advantage in distinguishing between all real versus all simulated proofs is at most $Q_{\mathsf{Report}} \cdot \mathsf{ZKAdv}(\mathsf{PoK})$, which remains negligible so long as $\mathsf{ZKAdv}(\mathsf{PoK})$ is negligible.

We omit the standard reduction that formalizes this indistinguishability argument.

- Hyb_4: This hybrid is identical to the preceding one, except the experiment keeps track of queries made to the random oracle H and aborts if there are ever queries rep, rep′ made to the oracle such that $\mathsf{rep} \neq \mathsf{rep}'$ but $H(\mathsf{rep}) = H(\mathsf{rep}')$.
  This event occurs with negligible probability because the probability of two queries to the random oracle having the same output is negligible in the length of the output.
- Hyb_5: This hybrid is identical to the preceding one except we replace calls to $\mathsf{PKE.Enc}(\mathsf{pk}, \cdot)$ in Report with calls to encrypt a string of zeros of the same length. The experiment keeps a table T_{Enc} indexed by ciphertexts that keeps the intended plaintext contents of those ciphertexts. This table is used to look up plaintexts when calls are made to PKE.Dec for ciphertexts $\mathsf{ct} \in T_{\mathsf{Enc}}$ in Verify.
  In lemma C.1, we prove that this hybrid is indistinguishable from the preceding one by the CCA security of the encryption scheme.
- Hyb_6: This hybrid is identical to the preceding experiment, except we add an additional abort condition. The experiment will abort and output 0 if there are two different $\mathsf{ct}, \mathsf{ct}' \in T$ where, after looking up their user identifiers uid, uid′ in T and running Verify, it holds that $\mathsf{uid} \neq \mathsf{uid}'$ but $\mathsf{dupTag} = \mathsf{dupTag}'$.
  This hybrid is statistically indistinguishable from the preceding one. Observe that $\mathsf{dupTag} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$. Since the experiment already aborts if there are rep, rep′ that have the same hash, the only remaining way for dupTags to collide is if (1) $\mathsf{sk}_u = \mathsf{sk}'_u$ for two users $u, u' \in U$ or (2) $H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}} = H(\mathsf{rep}')^{\mathsf{sk}'_u \mathsf{sk}_{1\mathsf{rep}}}$ where $\mathsf{rep} \neq \mathsf{rep}'$. But event (1) happens with probability $1/q$ for each pair of users, or $Q^2_{\mathsf{AddHonUser}}/q$ overall. Since outputs of H and $\mathsf{sk}_u, \mathsf{sk}'_u$ are chosen uniformly at random, there is a $1/q$ probability that any two dupTags collide, or a Q^2_{Report}/q probability of event (2) overall. Since both these probabilities are negligible, the hybrid is indistinguishable from the preceding one.
- Hyb_7 : In this hybrid, instead of calculating $v \leftarrow w^{\mathsf{sk}_u}$ in the Report protocol, the experiment samples $v \xleftarrow{\mathrm{R}} G$. The experiment keeps a table T_t of the t values returned by $\mathcal{A}$, along with the values of rep, r, and sk_u which would have been used to calculate v, i.e., $T_t[t] \leftarrow (\mathsf{rep}, r, \mathsf{sk}_u)$. The experiment uses these values, along with the extracted $\mathsf{sk}_{1\mathsf{rep}}$, to calculate values of dupTag for $\mathsf{ct} \in T$ as if it had not changed the calculation of v. That is, it computes

$$(\mathsf{rep}, r, \mathsf{sk}_u) \leftarrow T_t[t]$$
$$\mathsf{dupTag} \leftarrow H(\mathsf{rep})^{r \mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}/r} = w^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}/r}.$$

  In Lemma C.2, we prove that this hybrid is indistinguishable from the preceding one by the hardness of DDH in G.
- Hyb_8 : In this hybrid, instead of calculating $w \leftarrow H(\mathsf{rep})^r$ in the Report protocol, the experiment samples $w \xleftarrow{\mathrm{R}} G$. The experiment continues to use the values in the table T_t to calculate values of dupTag for $\mathsf{ct} \in T$ as if it had not changed the calculation of w. In Lemma C.3, we prove that this hybrid is indistinguishable from the preceding one by the hardness of DDH in G.
- Hyb_9 : In this hybrid, the experiment samples $\mathsf{dupTag} \xleftarrow{\mathrm{R}} G$ when running Verify for $\mathsf{ct} \in T$.
  In Lemma C.4, we prove that this hybrid is indistinguishable from the preceding one by the hardness of DDH in G.
- Hyb_{10} : This hybrid is identical to the preceding one, except we add an additional abort condition. The experiment will abort and output 0 if the Verify function, when run on a ciphertext $\mathsf{ct} \notin T$, returns a (rep, dupTag) tuple where the dupTag matches the dupTag that would have been produced by an honest user $u \in U$ for the same rep, but where (u, rep) does not appear in D_0 or D_1.
  This hybrid is statistically indistinguishable from the preceding one because the abort condition can only be met with negligible probability. This is the case because the values of dupTag for $\mathsf{ct} \in T$ are selected uniformly at random in G, and if (u, rep) does not appear in D_0 or D_1, they are never shown to the adversary. Thus the probability that an adversary produces a matching dupTag is the probability that one of the $\mathsf{ct} \notin T$ that the adversary sends to the Submit oracle matches with a random $\mathsf{ct} \in T$. Since $|T|$ is upper bounded by the number of calls to Report, this probability is at most $Q_{\mathsf{Report}} \cdot Q_{\mathsf{Submit}}/q$, which is negligible.
- Hyb_{11}: This hybrid is identical to the preceding one, except we switch the experiment's input b from $b = 0$ to $b = 1$.
  The view of the adversary in this hybrid is identical to its view in the preceding one because nothing in the adversary's view depends on b. Observe that all the values sent by the experiment to the challenger in the Report protocol are either random group elements (w, v), simulated proofs (π_u), or encryptions of zeroes (ct). Moreover, the output of Process is identical when $b = 0$ or $b = 1$ because the experiment aborts in any situation where a difference would arise due to the choice of b.
  In particular, whenever the experiment does not abort, each (uid, rep) pair input to Report results in a distinct dupTag for dupTag values corresponding to honest users. This means that no $\mathsf{ct} \notin T$ will result in a dupTag that collides with one in a ciphertext $\mathsf{ct} \in T$ for a different user. Let R_{Hon} be the value of R restricted to its contents due to calling Verify on ciphertexts $\mathsf{ct} \in T$. Then we have that $R_{\mathsf{Hon}} = R_0$ when $b = 0$ and $R_{\mathsf{Hon}} = R_1$ when $b = 1$ (since the abort criteria ensure that no $\mathsf{ct} \notin T$ can affect $R_{\mathsf{Hon}}, R_0, R_1$). In both cases, the experiment outputs 0 if $R_0 \neq R_1$, so the value of R_{Hon} is the same regardless of b. This means that R is also the same regardless of b because ciphertexts $\mathsf{ct} \notin T$ do not depend on b.
- Hyb_{12} : This hybrid is identical to the preceding one, except we remove the abort criterion introduced in Hyb_{10}. This hybrid is indistinguishable from the preceding hybrid via the same statistical argument made in Hyb_{10}.
- Hyb_{13} : This hybrid is identical to the preceding one, except we return to always calculating dupTag as specified in Hyb_7. This undoes the change made in Hyb_9 and is indistinguishable from the preceding hybrid via the same argument, relying on the hardness of DDH in G.
- Hyb_{14} : This hybrid is identical to the preceding one, except we return to always calculating w as specified in the protocol. This undoes the change made in Hyb_8 and is indistinguishable from the preceding hybrid via the same argument, relying on the hardness of DDH in G.
- Hyb_{15} : This hybrid is identical to the preceding one, except we return to always calculating v and dupTag as specified in the

protocol. This undoes the change made in Hyb_7 and is indistinguishable from the preceding hybrid via the same argument, relying on the hardness of DDH in G.

- Hyb_{16} : This hybrid is identical to the preceding one, except we drop the additional abort criteria specified in Hyb_6. This undoes the change made in that hybrid, and is indistinguishable from the preceding hybrid via the same argument.
- Hyb_{17}: This hybrid is identical to the preceding one, except encryption is done as specified in the protocol, rather than always encrypting zeros and looking up plaintexts in T_{Enc} to decrypt. This undoes the change made in Hyb_5.
  This hybrid is indistinguishable from the preceding one by the CCA security of the encryption scheme PKE, by an argument analogous to the one made there.
- Hyb_{18}: This hybrid is identical to the preceding one, except the experiment no longer aborts in the case of two queries rep, rep′ made to the random oracle H such that $\mathsf{rep} \neq \mathsf{rep}'$ but $H(\mathsf{rep}) = H(\mathsf{rep}')$. This undoes the change made in Hyb_4, and is indistinguishable from the preceding hybrid via the same argument.
- Hyb_{19} : This hybrid is identical to the preceding one, except the experiment no longer simulates the proofs π_u and uses real proofs instead. This undoes the changes made in Hyb_3, and is indistinguishable from the preceding hybrid via the same argument.
- Hyb_{20} : This hybrid is identical to the preceding one, except the experiment no longer simulates the proofs π_v and uses real proofs instead. This undoes the changes made in Hyb_2, and is indistinguishable from the preceding hybrid via the same argument.
- Hyb_{21}: This hybrid is identical to the preceding one, except the experiment no longer runs the extractors to recover $\mathsf{sk}_{\mathsf{1rep}}$ from each proof π_s provided by the adversary in Report. This undoes the change made in Hyb_1, and is indistinguishable from the preceding hybrid via the same argument used there.
  Note that this hybrid is identical to $\mathsf{RCONF}[\mathcal{A}, \Pi, \lambda, Q_O, 1]$.

The proof of the theorem follows from the indistinguishability of adjacent pairs of hybrids and the triangle inequality.

Lemma C.1. *Suppose that for any adversary $\mathcal{B}$ attacking the CCA security of* PKE*, the advantage of $\mathcal{B}$ in winning the CCA security experiment is at most* $\mathsf{CCAAdv}(\mathcal{B}, \mathsf{PKE})$. *Then, we have that*

$$\left|\Pr[\mathsf{Hyb}_4() = 1] - \Pr[\mathsf{Hyb}_5() = 1]\right| \leq \mathsf{CCAAdv}(\mathcal{B}, \mathsf{PKE}).$$

Proof. We show that if there exists an adversary $\mathcal{A}$ who distinguishes between the two hybrids, then we can build an adversary $\mathcal{B}$ who breaks the CPA security of PKE.Enc. $\mathcal{B}$ plays the role of the challenger to $\mathcal{A}$ and the adversary in the CPA security game. It simulates Hyb_5 exactly, except for two changes. It sets pk_2 to be the public key provided by the CPA security challenger, and whenever Report makes a call to PKE.Enc, it submits two plaintexts to the CPA security challenger: the plaintexts that are encrypted in Hyb_4 and Hyb_5. Since $\mathcal{B}$ keeps a table T_{Enc} as described in Hyb_5, correctness decryption and the outcomes of Process are identical in both cases. Thus if the CPA challenger has input $b = 0$, the adversary $\mathcal{B}$ perfectly simulates Hyb_4 to $\mathcal{A}$, and if the CPA challenger has $b = 1$, $\mathcal{B}$ perfectly simulates Hyb_5. Thus $\mathcal{B}$ distinguishes $b = 0$ vs $b = 1$ in the CPA security game with the same advantage that $\mathcal{A}$ distinguishes between Hyb_4 and Hyb_5. ◻

Lemma C.2. *Suppose that for any adversary $\mathcal{B}$ attacking DDH in G, the advantage of $\mathcal{B}$ in winning the DDH experiment is at most* $\mathsf{DDHAdv}(\mathcal{B}, G)$. *Then, modeling H as a random oracle, we have that*

$$\left|\Pr[\mathsf{Hyb}_6() = 1] - \Pr[\mathsf{Hyb}_7() = 1]\right| \leq \mathsf{DDHAdv}(\mathcal{B}, G).$$

Proof. We show how to build an adversary $\mathcal{B}$ who breaks DDH using an adversary $\mathcal{A}$ who distinguishes between the two hybrids. The adversary $\mathcal{B}$ begins by receiving the DDH challenge tuple X, Y, Z. It responds to random oracle queries by sampling a random $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H(\mathsf{rep}_i) \leftarrow g^{\alpha_i}$. In the AddHonUser oracle, it samples $\beta_u \xleftarrow{R} \mathbb{Z}_q$ and sets $\mathsf{pk}_u \leftarrow Y^{\beta_u}$. In the Report protocol, it samples $\gamma_i \xleftarrow{R} \mathbb{Z}_q$ and sets $w \leftarrow X^{\alpha_i \gamma_i}$, where α_i is chosen by querying the random oracle at rep. Moreover, it sets $v \leftarrow Z^{\alpha_i \gamma_i \beta_u}$. Instead of recording r when producing ct, the adversary $\mathcal{B}$ records γ_i. Finally, when running the Verify oracle for $\mathsf{ct} \in T$, The adversary $\mathcal{B}$ computes dupTag as $Y^{\alpha_i \beta_u \mathsf{sk}_{\mathsf{1rep}}}$ (straightforward bookkeeping can allow $\mathcal{B}$ to recover the correct choices of α_i, β_u). At the end of the experiment, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own.

Observe that if the DDH challenger has sent $\mathcal{B}$ a real DDH triple, i.e., $X = g^x, Y = g^y, Z = g^{xy}, x, y, z \in \mathbb{Z}_q$, then $\mathcal{B}$ is providing $\mathcal{A}$ with a perfect simulation of Hyb_6. This is because we have implicitly set $\mathsf{sk}_u = y\beta_u$ and $r = x\gamma_i$, and all the group elements that make up the adversary's view (w, v, dupTag) are consistent with this assignment of variables.

$$\begin{aligned}
w &\leftarrow X^{\alpha_i \gamma_i} = g^{x \alpha_i \gamma_i} = H(\mathsf{rep}_i)^{x \gamma_i} = H(\mathsf{rep}_i)^r \\
v &\leftarrow Z^{\alpha_i \gamma_i \beta_u} = g^{xy \alpha_i \gamma_i \beta_u} = H(\mathsf{rep}_i)^{x \gamma_i y \beta_u} = w^{\mathsf{sk}_u} \\
\mathsf{dupTag} &\leftarrow Y^{\alpha_i \beta_u \mathsf{sk}_{\mathsf{1rep}}} = g^{y \alpha_i \beta_u \mathsf{sk}_{\mathsf{1rep}}} = H(\mathsf{rep}_i)^{\mathsf{sk}_u \mathsf{sk}_{\mathsf{1rep}}} = t^{1/r}
\end{aligned}$$

On the other hand, if Z is a random group element, then we have that v is a random group element as well, but the other aspects of the adversary's view remain the same. This is a perfect simulation of Hyb_7. Thus the adversary $\mathcal{B}$ distinguishes a DDH triple from a random one with the same advantage that $\mathcal{A}$ distinguishes between the two hybrids. ◻

Lemma C.3. *Suppose that for any adversary $\mathcal{B}$ attacking DDH in G, the advantage of $\mathcal{B}$ in winning the DDH experiment is at most* $\mathsf{DDHAdv}(\mathcal{B}, G)$. *Then, modeling H as a random oracle, we have that*

$$\left|\Pr[\mathsf{Hyb}_7() = 1] - \Pr[\mathsf{Hyb}_8() = 1]\right| \leq \mathsf{DDHAdv}(\mathcal{B}, G).$$

Proof. We show how to build an adversary $\mathcal{B}$ who breaks DDH using an adversary $\mathcal{A}$ who distinguishes between the two hybrids. The adversary $\mathcal{B}$ begins by receiving the DDH challenge tuple X, Y, Z. It responds to random oracle queries by sampling a random $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H(\mathsf{rep}_i) \leftarrow X^{\alpha_i}$. In the Report protocol, it samples $\beta_i \xleftarrow{R} \mathbb{Z}_q$ and sets $w \leftarrow Z^{\alpha_i \beta_i}$, where α_i is chosen by querying the random oracle at rep. When running the Verify oracle for $\mathsf{ct} \in T$, the adversary $\mathcal{B}$ computes dupTag as $X^{\alpha_i \mathsf{sk}_u \mathsf{sk}_{\mathsf{1rep}}}$ (straightforward bookkeeping can allow $\mathcal{B}$ to recover the correct choices of α_i). At the end of the experiment, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own.

Observe that if the DDH challenger has sent $\mathcal{B}$ a real DDH triple, i.e., $X = g^x, Y = g^y, Z = g^{xy}, x, y, z \in \mathbb{Z}_q$, then $\mathcal{B}$ is providing

$\mathcal{A}$ with a perfect simulation of Hyb_7. This is because we have implicitly set $r = y\beta_i$ and explicitly set $H(\mathsf{rep}) = g^{x\alpha_i}$, and all the group elements that make up the adversary's view (w, v, dupTag) are consistent with this assignment of variables.

$$w \leftarrow Z^{\alpha_i\beta_i} = g^{xy\alpha_i\beta_i} = H(\mathsf{rep})^{y\beta_i} = H(\mathsf{rep})^r$$
$$v \xleftarrow{R} G$$
$$\mathsf{dupTag} \leftarrow X^{\alpha_i \mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}} = g^{x\alpha_i \mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$$

On the other hand, if Z is a random group element, then we have that w is a random group element as well, but the other aspects of the adversary's view remain the same. This is a perfect simulation of Hyb_8. Thus the adversary $\mathcal{B}$ distinguishes a DDH triple from a random one with the same advantage that $\mathcal{A}$ distinguishes between the two hybrids. □

Lemma C.4. *Suppose that for any adversary $\mathcal{B}$ attacking DDH in G, the advantage of $\mathcal{B}$ in winning the DDH experiment is at most $\mathsf{DDHAdv}(\mathcal{B}, G)$. Then, modeling H as a random oracle, we have that*

$$\left|\Pr[\mathsf{Hyb}_8() = 1] - \Pr[\mathsf{Hyb}_9() = 1]\right| \le \mathsf{DDHAdv}(\mathcal{B}, G).$$

Proof. We show how to build an adversary $\mathcal{B}$ who breaks DDH using an adversary $\mathcal{A}$ who distinguishes between the two hybrids. The adversary $\mathcal{B}$ begins by receiving the DDH challenge tuple X, Y, Z. It responds to random oracle queries by sampling a random $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H(\mathsf{rep}_i) \leftarrow X^{\alpha_i}$. In the AddHonUser oracle, it samples $\beta_u \xleftarrow{R} \mathbb{Z}_q$ and sets $\mathsf{pk}_u \leftarrow Y^{\beta_u}$. When running the Verify oracle for $\mathsf{ct} \in T$, the adversary $\mathcal{B}$ computes dupTag as $Z^{\alpha_i\beta_u \mathsf{sk}_{1\mathsf{rep}}}$ (straightforward bookkeeping can allow $\mathcal{B}$ to recover the correct choices of α_i, β_u). At the end of the experiment, $\mathcal{B}$ passes on $\mathcal{A}$'s output as its own.

Observe that if the DDH challenger has sent $\mathcal{B}$ a real DDH triple, i.e., $X = g^x, Y = g^y, Z = g^{xy}, x, y, z \in \mathbb{Z}_q$, then $\mathcal{B}$ is providing $\mathcal{A}$ with a perfect simulation of Hyb_8. This is because we have implicitly set $\mathsf{sk}_u = y\beta_u$ and explicitly set $H(\mathsf{rep}) = g^{x\alpha_i}$, and all the group elements that make up the adversary's view (w, v, dupTag) are consistent with this assignment of variables.

$$w \xleftarrow{R} G$$
$$v \xleftarrow{R} G$$
$$\mathsf{dupTag} \leftarrow Z^{\alpha_i\beta_u \mathsf{sk}_{1\mathsf{rep}}} = g^{xy\alpha_i\beta_u \mathsf{sk}_{1\mathsf{rep}}} = H(\mathsf{rep})^{\mathsf{sk}_u \mathsf{sk}_{1\mathsf{rep}}}$$

On the other hand, if Z is a random group element, then we have that dupTag is a random group element as well, but the other aspects of the adversary's view remain the same. This is a perfect simulation of Hyb_9. Thus the adversary $\mathcal{B}$ distinguishes a DDH triple from a random one with the same advantage that $\mathcal{A}$ distinguishes between the two hybrids. □

□

Proof of Theorem 6.3 (reporter anonymity).

Proof. The proof proceeds by a series of indistinguishable hybrids.

- Hyb_0: This hybrid is the security experiment $\mathsf{RANON}[\mathcal{A}, \Pi, \lambda, b = 0]$.
- Hyb_1: This hybrid is identical to the preceding hybrid, except in calls to the HonReport oracle, the experiment omits producing or verifying the proofs π_u and π_s.
  This change does not affect the view of the adversary in the experiment because the adversary never sees the transcript of interactions in HonReport, and the proofs have perfect completeness, meaning omitting them will not change the probability that the Report protocol outputs $\perp$.
- Hyb_2: This hybrid is identical to the preceding one, except instead of encrypting $\mathsf{hd} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{1\mathsf{rd}}, \mathsf{rd})$ in the HonReport oracle, we replace rd with a string of zeros of the appopriate length.
  This hybrid is indistinguishable from the preceding one by the CPA security of the encryption scheme. This can be proven via a standard reduction, which we omit.
- Hyb_3: This hybrid is identical to the preceding one, except instead of computing $v \leftarrow w^{\mathsf{sk}_u}$ in the HonReport oracle, we compute $v \xleftarrow{R} G$.
  In Lemma C.5, we prove that this hybrid is indistinguishable from the preceding one by the hardness of DDH in G and the fact that H is modeled as a random oracle.
- Hyb_4: This hybrid is identical to the preceding one except we switch the experiment's input b from $b = 0$ to $b = 1$.
  Observe that nothing in the adversary's view in Hyb_3 depends on b, so this hybrid is identical to the preceding one.
- Hyb_5: In this hybrid, instead of computing $v \xleftarrow{R} G$ in HonReport, we compute $v \leftarrow w^{\mathsf{sk}_u}$. This undoes the change made in Hyb_3.
  As in Hyb_3, this hybrid is indistinguishable from the preceding one by the hardness of DDH in G and the fact that H is modeled as a random oracle. The proof is analogous to that of Lemma C.5.
- Hyb_6: In this hybrid, the experiment resumes using rd as the plaintext that gets encrypted to produce hd. This undoes the change made in Hyb_2.
  As in Hyb_2, this hybrid is indistinguishable from the preceding one by the CPA security of PKE.
- Hyb_7: In this hybrid, the experiment resumes computing and verifying the proofs π_u and π_s in the HonReport oracle. This undoes the change made in Hyb_1.
  As in Hyb_1, this change does not affect the view of the adversary in the experiment. Note that this hybrid is identical to $\mathsf{RANON}[\mathcal{A}, \Pi, \lambda, b = 1]$.

The proof of the theorem follows from the indistinguishability of adjacent pairs of hybrids and the triangle inequality.

Lemma C.5. *Suppose that for any adversary $\mathcal{B}$ attacking DDH in G, the advantage of $\mathcal{B}$ in winning the DDH experiment is at most $\mathsf{DDHAdv}(\mathcal{B}, G)$. Then, modeling the hash function H as a random oracle, we have that*

$$\left|\Pr[\mathsf{Hyb}_2() = 1] - \Pr[\mathsf{Hyb}_3() = 1]\right| \le \mathsf{DDHAdv}(\mathcal{B}, G).$$

Proof. We show that if there exists an adversary $\mathcal{A}$ who distinguishes between the two hybrids, then we can build an adversary $\mathcal{B}$ who breaks DDH in G. $\mathcal{B}$ plays the role of the adversary in the DDH security game and the role of the challenger in the reporter anonymity game with $\mathcal{A}$. Given the DDH challenge tuple $(X = g^x, Y = g^y, Z = g^z)$ where $z = xy$ or $z \xleftarrow{R} \mathbb{Z}_q$, algorithm $\mathcal{B}$ programs the random oracle H so that for each query rep, $H(\mathsf{rep}) \leftarrow X^{\alpha}$ where $\alpha \xleftarrow{R} \mathbb{Z}_q$. Moreover, it sets the public key of each honest user to $\mathsf{pk}_u \leftarrow Y^{\beta}$ for $\beta \xleftarrow{R} \mathbb{Z}_q$. Finally, when computing v in the HonReport oracle, instead of computing $v \leftarrow H(\mathsf{rep})^{r\mathsf{sk}_u}$, it sets $v \leftarrow Z^{r\alpha\beta}$ where α and β are selected based on the message being hashed and the user doing the reporting. $\mathcal{B}$ passes on $\mathcal{A}$'s distinguishing bit b' as its own output.

Observe that if $z = xy$, then $\mathcal{B}$ has set $v = g^{rxy\alpha\beta} = (g^{x\alpha})^{ry\beta} = H(\mathsf{rep})^{r\mathsf{sk}_u}$, whereas if z is random, $\mathcal{B}$ has set $v = g^{rz}$, which is distributed uniformly at random in G. The former is exactly the view of the adversary in Hyb_2, whereas the latter is exactly the view of the adversary in Hyb_3. Thus $\mathcal{B}$ distinguishes between the two hybrids with the exact same advantage as $\mathcal{A}$. □

□

Proof of Theorem 6.4 (report uniqueness).

Proof. The proof proceeds through a series of hybrid experiments, each of which increases the adversary's advantage by at most a negligible probability.

- Hyb_0: This hybrid is the security experiment $\mathsf{RUNIQ}[\mathcal{A}, \Pi, \lambda, Q]$.
- Hyb_1: In this hybrid, the experiment keeps a table T_{MAC} of messages MACed by S_1, indexed by the MAC tags σ, i.e., $T[\sigma] = (w, t)$. The experiment aborts and outputs 0 if it ever calls the Verify function ever receives a MAC tag $\sigma \notin T_{\mathsf{MAC}}$ but does not output $\perp$.
  This hybrid is indistinguishable from the preceding one by the existential unforgeability of the MAC scheme. We omit the proof of indistinguishability for this hybrid because it is a standard reduction to the existential unforgeability of the MAC scheme.
- Hyb_2: This hybrid is identical to the preceding one, except the experiment keeps track of queries made to the random oracle H and aborts if there are ever queries rep, rep′ made to the oracle such that $\mathsf{rep} \neq \mathsf{rep}'$ but $H(\mathsf{rep}) = H(\mathsf{rep}')$.
  This event occurs with negligible probability because the probability of two queries to the random oracle having the same output is negligible in the length of the output.
- Hyb_3: In this hybrid, the experiment runs the extractor guaranteed to exist by the proof of knowledge property of PoK to recover the value sk_u for each proof π_u presented in $\mathsf{MalReport}(\mathsf{pk}_u)$. The experiment outputs $\perp$ should any extractor fail. The experiment also modifies its bookkepping to replace each element $\mathsf{pk}_u \in M$ with the tuple $(\mathsf{sk}_u, \mathsf{pk}_u)$.
  This hybrid is indistinguishable from the preceding one by the proof of knowledge property of the proof system PoK. In particular, the experiment aborts with probability $\mathsf{PoKAdv}(\mathsf{PoK})$, the probability of the extractor failing, for each invocation of the MalReport oracle. Thus the overall additional failure probability introduced by this change is $Q_{\mathsf{MalReport}} \cdot \mathsf{PoKAdv}(\mathsf{PoK})$, which remains negligible so long as $\mathsf{PoKAdv}(\mathsf{PoK})$ is negligible.
- Hyb_4 : In this hybrid, the experiment keeps a table T_{ct} and each time the HonReport oracle computes a ciphertext, the experiment sets $T_{\mathsf{ct}} \leftarrow (\mathsf{rep}, t, \sigma, r, \mathsf{hd})$. The experiment also replaces any ciphertext computed in HonReport with an encryption of all zeroes of the same length, using T_{ct} to recover the plaintext whenever it encounters a ciphertext $\mathsf{ct} \in T_{\mathsf{ct}}$.
  In Lemma C.6, we show that the advantage of an adversary in this hybrid is at most $\mathsf{CCAAdv}(\mathcal{B}, \mathsf{PKE.Enc})$ greater than in the previous one. This quantity is negligible by the CCA-security of PKE.
- Hyb_5: In this hybrid, the experiment aborts and outputs 0 if, during a call to the Process oracle, there is ever a $\mathsf{ct} \notin T_{\mathsf{ct}}$ but for which $\mathsf{PKE.Dec}(\mathsf{pk}, \mathsf{ct}) = (\mathsf{rep}, t, \cdot, r, \cdot) \in T_{\mathsf{ct}}$.
  This event occurs with negligible probability because the view of the adversary is independent of values of r (and therefore t) produced in the HonReport oracle. Thus the abort criterion can only be triggered if the adversary guesses the random choice of r used in one of the calls to HonReport and includes it in a ciphertext ct passed to the Submit oracle.
- Hyb_6: In this hybrid, the experiment aborts if there exists $\mathsf{ct}, \mathsf{ct}' \in S$ where, when the ciphertexts are decrypted in Verify and Verify does not output $\perp$, they yield $(\mathsf{rep}, r), (\mathsf{rep}', r')$ such that $(\mathsf{rep}, r) \neq (\mathsf{rep}', r')$, but $H(\mathsf{rep})^r = H(\mathsf{rep}')^{r'}$.
  In Lemma C.7, we show that the advantage of an adversary in this hybrid is at most $Q_H \cdot \mathsf{DLAdv}(\mathcal{B}, G)$ greater than in the previous one, where Q_H denotes the number of queries the adversary makes to the random oracle. This quantity is negligible by the hardness of discrete log in G.

We now prove that the advantage of any adversary in Hyb_6 is 0.

First, let T_{Dec} be a table that maps those ciphertexts $\mathsf{ct} \in S$ for which $\mathsf{Verify}(\mathsf{sk}_s, \mathsf{sk}_2, \mathsf{ct}) \neq \perp$ to their decryptions $(\mathsf{rep}, t, \sigma, r, \mathsf{hd})$. Note that for a ciphertext to be included in T_{Dec}, its decrypted contents must pass MAC verification, which means that $(H(\mathsf{rep})^r, t) \in T_{\mathsf{MAC}}$. This means that only those (w, t) values that come from a successful interaction with HonReport or MalReport (the only times an experiment MACs a message) can be included in T_{Dec}. Note that for a ct to increase the count in R, it must at least be included in T_{Dec}. Moreover, for a ct to be included in the difference between R and HonR – call the table of differences R' – its corresponding (w, t) value must have been MACed in the MalReport oracle, or else the experiment would abort for violating the criterion specified in Hyb_5. We will refer to the subset of T_{Dec} that includes ciphertexts $\mathsf{ct} \notin T$ as T'_{Dec}. Since only ciphertexts $\mathsf{ct} \in T'_{\mathsf{Dec}}$ can contribute to count′, this means that count′ is upper bounded by the number of calls to MalReport. Since each successful call to MalReport increases count by 1, this means that, $\mathsf{count}' \leq \mathsf{count}'$.

Next, observe that for any $\mathsf{ct} \in T'_{\mathsf{Dec}}$, the decrypted values of $H(\mathsf{rep})^r = w$ and t must have the relationship $t = w^{\mathsf{sk}_{1\mathsf{rep}} \cdot \mathsf{sk}^*_u}$ by construction, where $\mathsf{sk}^*_u \in M$. But since there are no colliding $H(\mathsf{rep})$ values in the experiment, and no colliding $w = H(\mathsf{rep})^r$ values either, this means that for each $\mathsf{rep} \in T'_{\mathsf{Dec}}$, it must hold for all entries $(\mathsf{rep}, t, \sigma, r, \mathsf{hd}) \in T'_{\mathsf{Dec}}$ that $t^{1/r} = H(\mathsf{rep})^{r\mathsf{sk}_{1\mathsf{rep}} * \mathsf{sk}^*_u / r} = H(\mathsf{rep})^{\mathsf{sk}_{1\mathsf{rep}} \mathsf{sk}^*_u}$. Since there are at most $|M|$ possible choices of sk^*_u, there cannot be more than $|M|$ entries in R' for each unique rep, which means the adversary can never win with $\mathsf{diff} > |M|$.

We have now ruled out both ways for the experiment to set $\mathsf{win} \leftarrow 1$, meaning the adversary has advantage 0 in Hyb_6, and completing the proof.

Lemma C.6. *Suppose that for any adversary $\mathcal{B}$ attacking the CCA security of public key encryption scheme* PKE*, the advantage of $\mathcal{B}$ in winning the CCA security experiment is at most* $\mathsf{CCAAdv}(\mathcal{B}, \mathsf{PKE})$.

Then, we have that

$$\left|\Pr[\mathsf{Hyb}_3() = 1] - \Pr[\mathsf{Hyb}_4() = 1]\right| \leq \mathsf{CCAAdv}(\mathcal{B}, \mathsf{PKE}).$$

Proof. We show that for any adversary $\mathcal{A}$ who distinguishes between Hyb_3 and Hyb_4, we can build an adversary $\mathcal{B}$ who uses $\mathcal{A}$ to break the CCA security of PKE.

The adversary $\mathcal{B}$ performs the role of the challenger in the Hyb_4 experiment with a few changes. The public key pk_2 is set to be the public key provided by the CCA challenger. Whenever the HonReport oracle requires the honest user to encrypt a message, $\mathcal{B}$ encrypts messages by sending the two plaintexts $(\mathsf{rep}, t, \sigma, r, \mathsf{hd})$ and all zeroes to the CCA security challenger. All decryptions of ciphertexts returned by the CCA challenger are decrypted via lookup table, and decryptions of other ciphertexts are handled via the CCA decryption oracle.

Observe that if the CCA challenger has input bit $b = 0$, then $\mathcal{B}$ provides the adversary $\mathcal{A}$ with a perfect simulation of Hyb_3, whereas if $b = 1$, then $\mathcal{B}$ provides a perfect simulation of Hyb_4. Thus $\mathcal{B}$ wins the CCA security game with the same advantage that $\mathcal{A}$ distinguishes between the two hybrids. □

Lemma C.7. *Suppose that for any adversary $\mathcal{B}$ attacking discrete logarithm in G, the advantage of $\mathcal{B}$ in computing a discrete logarithm is at most* $\mathsf{DLAdv}(\mathcal{B}, G)$*. Then, for an adversary who makes at most Q_H queries to the random oracle H, we have that*

$$\left|\Pr[\mathsf{Hyb}_5() = 1] - \Pr[\mathsf{Hyb}_6() = 1]\right| \leq Q_H \cdot \mathsf{DLAdv}(\mathcal{B}, G).$$

Proof. We show that for any adversary $\mathcal{A}$ who triggers the abort condition introduced in Hyb_6, we can build an adversary $\mathcal{B}$ who uses $\mathcal{A}$ to solve discrete logarithms in G with probability $1/Q_H$. Since the abort condition is the only difference between Hyb_5 and Hyb_6, this proves that the advantage of an adversary against Hyb_6 is at most $Q_H \cdot \mathsf{DLAdv}(\mathcal{B}, G)$ greater than that of an adversary against Hyb_5. The proof proceeds by programming the random oracle and using a guessing argument to solve discrete logarithms.

Given a discrete log challenge (g, h), the adversary $\mathcal{B}$ begins by sampling a random $i^* \xleftarrow{R} \{1, ..., Q_H\}$. Then, during the experiment, for the ith query rep_i to H, $i \neq i^*$, $\mathcal{B}$ responds by sampling $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and setting $H(\mathsf{rep}_i) \leftarrow g_i^{\alpha}$. For $i = i^*$, $\mathcal{B}$ sets $H(\mathsf{rep}_{i^*}) \leftarrow h$. The experiment keeps a table of T_H of $(\mathsf{rep}_i, \alpha_i)$ pairs.

We now show that whenever the abort criterion introduced in Hyb_6 occurs, $\mathcal{B}$ solves the discrete logarithm challenge with probability $2/Q_H$. Consider the values $(\mathsf{rep}, r), (\mathsf{rep}', r')$ such that $H(\mathsf{rep})^r = H(\mathsf{rep}')^{r'}$ triggers the abort condition. The adversary $\mathcal{B}$ aborts if $\mathsf{rep} = \mathsf{rep}_{i^*}$ and $\mathsf{rep}' \neq \mathsf{rep}_{i^*}$, or vice versa. Since i^* is chosen uniformly at random, the probability that $\mathcal{B}$ does not abort is at least $2/Q_H$.

Suppose without loss of generality that $\mathsf{rep} = \mathsf{rep}_{i^*}$. Then we have that $H(\mathsf{rep}) = h$ and $H(\mathsf{rep}') = g^{\alpha}$, where $\alpha = \alpha_i$ for some $i \neq i^*$. Thus we have that

$$H(\mathsf{rep})^r = H(\mathsf{rep}')^{r'}$$
$$h^r = (g^{\alpha})^{r'} = g^{\alpha r'}$$
$$h = g^{\alpha r'/r}.$$

Since $\mathcal{B}$ knows α, r', and r, it outputs $\alpha r'/r$ to the discrete log challenger, and wins the discrete log security experiment. □

□

Proof of Theorem 6.5 (threshold unforgeability). We only provide a sketch of this proof, as the arguments are very similar to those made in the other theorems.

Proof (sketch). First, we run the extractor for each call to the Verify oracle to recover the values $(\mathsf{rep}, r, w, \mathsf{duptag}, t)$ used in each clause of each call to Verify. These values are deduplicated and stored in a table T of size at most $Q_{\mathsf{HonReport}} < Q_H$ (because the Report protocol includes a call to H).

Next, we invoke the fact that the random oracle behaves as a collision-resistant hash function to rule out the possibility of collisions in $H(\mathsf{rep})$, except with negligible probability. We also invoke the hardness of discrete log in G to rule out the possibility of colliding (rep, r) and (rep', r') where $H(\mathsf{rep})^r = H(\mathsf{rep}')^{r'}$, similarly to the argument made in the proof of report uniqueness.

Observe that since each (w, t) held by the verifier is associated with a single extracted value r, that $H(\mathsf{rep})^r = w$, and that there are no colliding values of $H(\mathsf{rep})^r$, we can conclude that each (r, w, t) uniquely determines rep, which in turn uniquely determines the dupTag such that $\mathsf{dupTag}^r = t$.

From here, the proof is a reduction to discrete logarithm. Given a discrete logarithm challenge $h = g^x$, we pick a random query rep^* to the random oracle and program it to be

$$r^* \xleftarrow{R} \mathbb{Z}_q; H(\mathsf{rep}^*) \leftarrow h^{r^*}.$$

We call rep the *special* query. All other queries to the random oracle are programmed as

$$r_{\mathsf{rep}} \xleftarrow{R} \mathbb{Z}_q; H(\mathsf{rep}) \leftarrow g^{r_{\mathsf{rep}}}.$$

Now, whenever the adversary wins the security experiment, there must be some $(w, t) \in T$ for which the extracted values (rep', r') differ from the (rep, r) initially produced by the experiment during HonReport. Since we chose rep^* at random from among queries to H, there is at least a $1/Q_H$ probability that $\mathsf{rep} = \mathsf{rep}^*$ in this case. But then, because the proof π_v verified, we have that

$$w = H(\mathsf{rep}')^{r'} = g^{r_{\mathsf{rep}'} r'} = h^{r^* r} = H(\mathsf{rep}^*)^r.$$

This implies that $h = g^{\frac{r_{\mathsf{rep}'} r'}{r^* r}}$, so we can recover the discrete logarithm $x = \frac{r_{\mathsf{rep}'} r'}{r^* r}$. □