Improving Signal's Sealed Sender

Ian Martiny*, Gabriel Kaptchuk†, Adam Aviv‡, Dan Roche§, Eric Wustrow*
*University of Colorado Boulder, {ian.martiny, ewust}@colorado.edu
†Boston University, kaptchuk@bu.edu
‡George Washington University, aaviv@gwu.edu
§U.S. Naval Avademy, roche@usna.edu

***Abstract*—The Signal messaging service recently deployed a *sealed sender* feature that provides sender anonymity by cryptographically hiding a message's sender from the service provider. We demonstrate, both theoretically and empirically, that this one-sided anonymity is broken when two parties send multiple messages back and forth; that is, the promise of sealed sender does not *compose* over a conversation of messages. Our attack is in the family of Statistical Disclosure Attacks (SDAs), and is made particularly effective by *delivery receipts* that inform the sender that a message has been successfully delivered, which are enabled by default on Signal. We show using theoretical and simulation-based models that Signal could link sealed sender users in as few as 5 messages. Our attack goes beyond tracking users via network-level identifiers by working at the application layer of Signal. This make our attacks particularly effective against users that employ Tor or VPNs as anonymity protections, who would otherwise be secure against network tracing. We present a range of practical mitigation strategies that could be employed to prevent such attacks, and we prove our protocols secure using a new simulation-based security definition for one-sided anonymity over any sequence of messages. The simplest provably-secure solution uses many of the same mechanisms already employed by the (flawed) sealed-sender protocol used by Signal, which means it could be deployed with relatively small overhead costs; we estimate that the extra cryptographic cost of running our most sophisticated solution in a system with millions of users would be less than $40 per month.**

I. Introduction

Secure end-to-end encrypted messaging applications, such as Signal, protect the content of messages between users from potential eavesdroppers using protocols like off-the-record (OTR) messaging [6], [18]. These protocols guarantee that even the service provider itself is unable to read communication between users. However, these protocols do not protect conversation *metadata*, including sender, recipient, and timing. For instance, if Alice sends a message to Bob, the server will learn that there is a relationship between those two users and when they communicated.

Protecting metadata. While leaking metadata may appear reasonable when compared to revealing the contents of the messages, observing metadata can have serious consequences. Consider that Alice may be a whistleblower communicating with a journalist [41] or a survivor of domestic abuse seeking confidential support [25]. In these cases, merely knowing *to whom* Alice is communicating combined with other contextual information is often enough to infer conversation content without reading the messages themselves. Former NSA and CIA director Michael Hayden succinctly illustrated this importance of metadata when he said the US government "kill[s] people based on metadata" [29].

Signal's recent *sealed sender* feature aims to conceal this metadata by hiding the message sender's identity. Instead of seeing a message from Alice to Bob, Signal instead observes a message to Bob from an anonymous sender. This message can only be decrypted by Bob, who then learns from the payload that the message originated with Alice. Ideally, using the sealed sender protocol breaks the link between the sender and the receiver, preventing Signal from recording sender-recipient pairs, if ever compromised or compelled to do so.

While sealed sender is currently only deployed by Signal, Signal's design decisions are highly influential for other secure messaging platforms as it is a leader in deploying cutting-edge secure messaging features; the Signal protocol has been integrated into other services like WhatsApp. Understanding and uncovering flaws in sealed sender is therefore not only important to protecting the privacy of Signal's millions[1] of users [23], but also helps make sure sealed sender fully realizes its goal before it is integrated into other services with other sets of users.

A new SDA on message timings. We present a new statistical disclosure attack (SDA) applicable to messages in Signal's sealed sender, that would allow the Signal service—if compelled by a government or compromised—to correlate senders and receivers even when using the sealed sender feature. Previously, *statistical disclosure attacks* (SDAs) have been studied since the 2000s to link senders and recipients in anonymous mix networks [14], [40], [44], [16], [37]. These attacks work by correlating sender and receiver behavior across multiple rounds of the mix.

It is not immediately obvious how SDAs could be applied in the context of sealed sender messages, since there is no mix network and the identities of senders are (by design) never revealed. Thus, it is not clear how even the server could apply SDA attacks, since it only learns the destinations of messages, and never sources.

In this paper, we observe that, *by assuming that most messages receive a quick response*, we can overcome these

[1]Signal does not publicly disclose its user count, but the app has been downloaded millions of times.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.23180
www.ndss-symposium.org

seeming limitations of sealed-sender messaging and employ a SDA-style attack to de-anonymize sender-recipient pairs after passively observing enough messages.

Moreover, and crucially, *this quick-response assumption is guaranteed to be true in the presence of delivery receipts*, a feature of Signal's current implementation that cannot be disabled by the user. When Alice sends Bob a sealed sender message, Bob's device will automatically generate a delivery receipt that acknowledges Alice's message. Although this delivery receipt is also sent via sealed sender to Alice, the predictability of its timing makes our attack more effective.

The differences between sealed sender messaging and a general mix network allow us to develop a simple, tailored SDA-style attack, using ideas similar to [40], which can be used to de-anonymize a conversation between two parties. Compared to prior work, our attack is more limited in scope, but is also more efficient: it runs in linear-time in the amount of traffic observed, and we prove that the probability our attack succeeds increases exponentially with the number of observations.

We validate the practicality of the timing attack in two ways. First, using a probabilistic model of communication, we prove a bound on the probability that Alice can be identified as communicating with Bob after a finite number of messages, independent of other users' activity. The probability also scales logarithmically with the number of active users.

Second, we run simulations to estimate the effectiveness of the attack in practice. In the most basic simulation, Alice can be uniquely identified as communicating with Bob after fewer than 10 messages. We also add complicating factors such as multiple simultaneous conversations with Alice and/or Bob and high-frequency users in the system, and show that these delay but do not prevent Alice from being de-anonymized.

Sealed sender conversations. To fix this problem, we provide a series of practical solutions that require only modest changes to Signal's existing protocol. We first define a simulation-based security model for sealed sender *conversations* (rather than just single messages) that allows the original recipient of the sealed sender message to be leaked but never the initiator of that message (sender) through the lifetime of the conversation. We then present three solutions that accomplish the goal of sealed sender conversations. Each is based on ephemeral identities, as opposed to communicating with long-term identifiers, such as the keys linked to your phone number in Signal. Each additional solution provides additional security protections.

Our first solution provably provides *one-way sealed-sender* conversations, a new security guarantee for which we provide a formal, simulation based definition. In this protocol, Alice initiates a sealed-sender conversation by generating a new ephemeral, public/secret key and anonymously registers the ephemeral public key with an anonymous mailbox via the service provider. Alice then uses a normal sealed sender message to the receiver Bob to send the anonymous mailbox identifier for his replies. Alice can retrieve Bob's replies sent to that anonymous mailbox by authenticating with her ephemeral secret key, and the conversation continues using traditional sealed sender messages between Bob's long-term identity and the anonymous mailbox Alice opened.

We show that this solution can be further enhanced if both Alice and Bob use ephemeral identities, after the initial message is sent (using sealed sender) to Bob's long-term identity. This protocol provides both sender and receiver anonymity for the length of a conversation if the server is unable to correlate Bob's receipt of the initial message and his anonymous opening of a new mailbox, meaning the server has only one chance to deanonymize Bob. Importantly, even if the server is able to link these two events, this extension still (provably) provides *one-way sealed-sender*.

Neither of the above solutions offer authentication of anonymous mailboxes at the service provider, e.g., Signal. A malicious user could open large numbers of anonymous mailboxes and degrade the entire system. We offer an overlay solution of *blind-authenticated anonymous mailboxes* for either one-way or two-way sealed-sender conversations whereby each user is issued anonymous credentials regularly (e.g., daily) that can be "spent" (verified anonymously via a blind signatures) to open anonymous new mailboxes. To evaluate the practicality of using anonymous credentials in this way, we run a series of tests to compute the resource overhead required to run this overlay. We estimate that running such a scheme on AWS would cost Signal approximately $40 each month to support 10 million anonymous mailboxes per day.

Our contributions. In this paper, we will demonstrate

- A brief analysis of how the Signal protocol sends messages and notifications based on source code review and instrumentation (Section II-B);
- The first attack on sealed sender to de-anonymize the initiator of a conversation in Signal (Section III);
- Validation of the attack via theoretical bounds and simulation models (Section IV);
- A new security model that defines allowed leakage for sender-anonymous communication;
- A set of increasingly secure solutions, that are either one-way anonymous, two-way anonymous, and/or provide anonymous abuse protections. (Section VI);
- An evaluation of the resource overhead introduced by using blind signatures to prevent anonymous mailbox abuse, and estimates of its effective scalability to millions of users (Section VI-E); and
- Immediate stopgap strategies for Signal users to increase the difficulty of our attack (Section VII-A).

We include related work and the relevant citations in Section VIII. We also want to be clear about the limitations of our work and its implications:

- We do *not* consider network metadata such as leakage due to IP addresses. See Section II-C and the large body of existing work on anonymizing proxies such as Tor.
- We do *not* consider messaging with more than two parties, i.e. group messaging. This is important future work; see the discussion in Section VII-C.
- Our attack does *not* suggest that Signal is less secure than alternatives, or recommend that users discontinue using it. Other messaging services do not even attempt to hide the identities of message senders.
- We do *not* believe or suggest that Signal or anyone else is using this attack currently.

- While we have implemented the core idea of our solution in order to estimate the cost of wider deployment, we have *not* undergone the serious engineering effort to carefully and correctly integrate this solution with the existing Signal protocol software in order to allow for practical, widespread deployment.

Responsible Disclosure. We have notified Signal of our attack and solutions prior to publication, and Signal has acknowledged our disclosure.

II. Background

We now give some background on the structure and types of messages in the Signal protocol [39], used in both the Signal and WhatsApp applications.

A. Sealed Sender Messages

Although secure end-to-end encrypted messaging applications like Signal protect the contents of messages, they reveal metadata about *which* users are communicating to each other. In an attempt to hide this metadata, Signal recently released a feature called sealed sender [36] that removes the sender from the metadata intermediaries can observe.

To send a sealed sender message to Bob, Alice connects to the Signal server and sends an encrypted message to Bob anonymously[2]. Within the payload of this encrypted message, Alice includes her own identity, which allows Bob to authenticate the message. Importantly, Signal still learns Bob's identity, which is needed in order to actually deliver it. The structure of sealed sender messages are illustrated in Figure 1.

Due to sender anonymity, Signal cannot directly rate-limit users to prevent spam or abuse. Instead, Signal derives a 96-bit *delivery token* from a user's profile key, and requires senders demonstrate knowledge of a recipients' delivery token to send them sealed sender messages. By only sharing this delivery token with his contacts, Bob limits the users who can send him sealed sender messages, thus reducing the risk of abuse[3].

B. Types of Messages

We manually reviewed and instrumented the Signal messenger Android 4.49.13 source code [42] in order to understand the types of messages Signal sends. In addition to the messages that contain content to be delivered to the receiver, there are several event messages that can be sent automatically, as discussed below. All of these messages are first padded to the next multiple of 160 bytes, then encrypted and sent using sealed sender (if enabled), making it difficult for the Signal service to distinguish events from normal messages based on their length.

Normal message. A normal text message or multimedia image sent from Alice to Bob is the typical message we consider. A short (text) message will be padded to 160 bytes, and longer messages padded to a multiple of 160 bytes, before encryption.

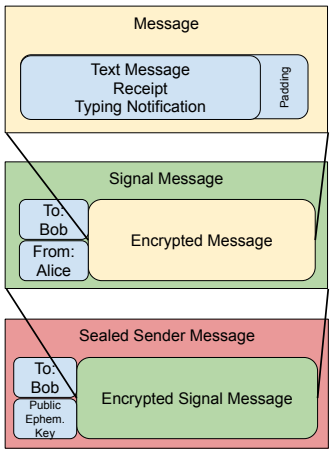

Fig. 1: **Structure of Signal Messages** — All messages Alice sends to Bob through Signal (receipts, text messages, or events) are first padded to the next multiple of 160 bytes. The padded message is then encrypted under the shared key between Alice and Bob and then combined with 'To: Bob' and 'From: Alice' metadata to form a Signal Message. If both Alice and Bob have sealed sender enabled then Alice will then generate an ECDHE key pair and derive a new shared secret with Bob's public key to encrypt the Signal Message and combine with 'To: Bob' and the public ephemeral key to form a sealed sender message that will be sent to Bob.

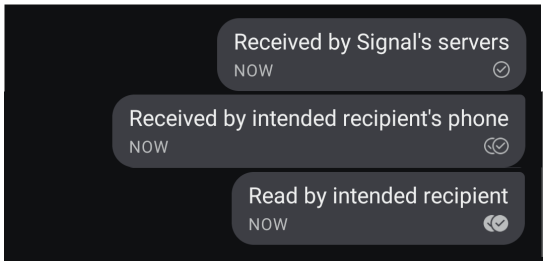

Fig. 2: **Stages of a Signal Message** — User Interface indicating message delivery status. One hollow check mark signifies that the message is en route. Two hollow check marks signifies the receipt of a delivery receipt for the message. Finally, two filled check mark signifies the receipt of a read receipt for the message.

Delivery receipt. When Bob's device *receives* a normal message, his device will automatically send back a delivery receipt to the sender. When Alice receives the delivery receipt for her sent message, her device will display a second check mark on her sent message to indicate that Bob's device has received the message (see Figure 2). If Bob's device is online when Alice sends her message, the delivery receipt will be sent back immediately. We measured a median time of 1480 milliseconds between sending a message and receiving a delivery receipt from an online device. (See Figure 3 for CDF of times.) These receipts *cannot be disabled* in Signal.

Read receipt (optional). Bob's device will (optionally) send a read receipt to the sender when he has *viewed* a normal message, triggering a UI update on Alice's device (see Fig-

[2]As we note in our threat model, we do not consider the information leakage from networking.

[3]There are a number of options available to Bob that can allow more fine-grained access control to his delivery token. Bob can opt to receive sealed sender messages from anyone even without knowledge of his delivery token, but this is disabled by default. Additionally, Bob can regenerate his delivery token and share it only with a subset of his contacts to block specific users.

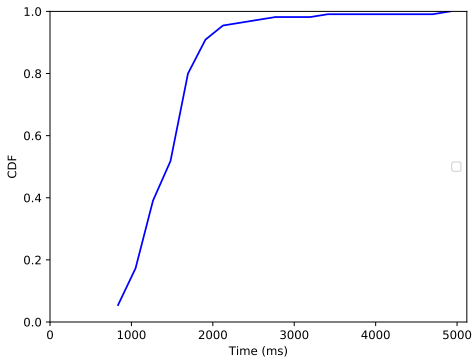

Fig. 3: **CDF of Delivery Receipt timing** — CDF of time between a device sending a message (to another online device) and receiving a Delivery Receipt. The median time is 1480ms and 90% of Delivery Receipts were received within 1909ms.

ure 2). Unlike delivery receipts, Bob can disable read receipts. However, Alice may still send read receipts for messages she receives from Bob. If Bob receives a read receipt but has the feature disabled, his user interface will not display the notification.

Typing notifications (optional). Alice's device will (optionally) send a *start typing* event when Alice is entering a message, which Bob's device will use to show that Alice is typing. If she does not edit the message for 3 seconds, a *stop typing* event will be sent. Each sent message is accompanied by a *stop typing* event to clear the receiver's typing notification. Like read receipts, typing notifications can be disabled such that the user will not send or display received notifications.

C. Threat Model

We assume that the service provider (e.g. Signal) passively monitors messages to determine which pairs of users are communicating. This models either an insider threat or a service provider compelled to perform surveillance in response to a government request. We assume Alice and Bob have already exchanged delivery tokens and they communicate using sealed sender. Once initiated, we assume that Alice and Bob will continue to communicate over time. Finally, we also assume that many other users will be communicating concurrently during Alice and Bob's conversation, potentially with Alice and/or Bob.

The service provider cannot view the contents of the encrypted sealed sender messages, but knows the destination user for these messages (e.g. someone sends a message to Bob). We assume that Alice and Bob have verified their respective keys out of band, and that the applications/devices they are using are secure. Although the service provider publishes the application, they typically distribute open-source code with deterministic builds, which we assume prevents targeting individual users.

We note that the service provider could infer a sender's identity from network metadata such as the IP address used to send a sealed sender message. However, this is a problem that could be solved by using a popular VPN or an anonymizing proxy such as Tor [45], [19]. For the purposes of this paper, we assume that users who wish to remain anonymous to Signal can use such proxies (e.g. Orbot [2]) when sending sealed sender messages (and, in our solution, when receiving messages to ephemeral mailboxes), and we do not use network metadata in our attack.

In terms of impact, we note that a recent study suggests as many as 15% of mobile users already use VPNs every day [28]; this prevalence is even higher in east Asia and, presumably, among vulnerable user populations.

III. Attack Description

We will present a kind of *statistical disclosure attack* (SDA) that can be used to de-anonymize a single user's contacts after a chain of back-and-forth messages, each of which is sent using sealed sender.

We first explain how, especially in the presence of delivery receipts, a sealed-sender messaging system can be viewed as a kind of mix network; this observation allows for the use of SDAs in our context and can be viewed as one of our contributions.

Next, we detail a simple attack for our specific use-case of sealed sender messaging, which can be viewed as a special case of an SDA attack proposed in [40].

A. From mixnets to sealed-sender

In anonymous networking, a simple *threshold mix* works as follows: When Alice wants to send a message to Bob, she instead encrypts it and sends it to a trusted party called the *mix*. Once the mix receives messages from a certain threshold τ number of other senders, the mix decrypts their destinations, shuffles them, and sends all messages out to their destinations at once. In this way, a network attacker can observe which users are sending messages and which are receiving message, but cannot easily infer which pairs of individuals are directly communicating.

The basis of SDAs, first proposed by [14], is that the messages sent through the mix over multiple rounds are not independent; a user such as Alice will normally send messages to the same associates (such as Bob) multiple times in different rounds. In the simplest case, if Alice sends messages only to Bob, and the other users in each round of mixing are random, then a simple *intersection attack* works by finding the unique common destination (Bob) out of all the mixes where Alice was one of the senders.

Over the last two decades, increasingly sophisticated variants of SDAs have been proposed to incorporate more complex mix networks [40], infer sender-receiver connections [37], adapt to the possibility of anonymous replies [16], and to use more powerful techniques to discover information about the entire network topology [17], [44]. Fundamentally, these all follow a similar anonymous networking model, where an attacker observes messages into and out of a mix network, and tries to correlate senders and receivers after a large number of observations.

At first, it seems that the setting of sealed-sender messaging is quite different: the server (acting as the mix) does not

apply any thresholds or delays in relaying messages, and the sender of each message is completely anonymous. Our key observation is that, *when many messages receive a quick reply*, as will be guaranteed in the presence of delivery receipts, a sealed-sender messaging system can be modeled as a kind of mix network:

- The *recipient* of a message, Bob, is more likely to send some reply in a short time window immediately after he receives a message: we call this time window an *epoch*.
- Bob's reply to Alice is "mixed" with an unknown, arbitrary number of other messages (which could be either normal messages or replies) during that epoch.
- The recipients of all messages during that epoch (following the message Bob received), can be considered as the message recipients out of the mix. Alice, who originally sent a message to Bob and is expected to receive a quick reply, will be among these recipients.

The task of our SDA, then, is to observe many such epochs following messages to a single target user, Bob, and attempt to discern the user Alice who is actually sending messages to Bob.

B. Attack Overview

Before proceeding to an overview of our attack, we first fix the terminology we will use:

Target/Bob The single user who is being monitored.
Associate/Alice Any user who sends some message(s) to the target Bob during the attack window
Non-associate/Charlie Any other user not sending messages to the target Bob.
Attack window The entire time frame under which the attack takes place, necessarily spanning multiple messages sent to the target Bob.
Target epoch A single epoch during the attack window immediately following a sealed sender message to the target. The epoch length is fixed depending on how long we should expect to see a response from the recipient.
Random epoch A single epoch during the attack window, of the same length as a Target epoch, but chosen uniformly at random over the attack window independently from Bob.

As discussed above, our attack setting is that a single user, Bob, is being targeted to discover an unknown associate Alice who is sending messages to Bob. Our SDA variant is successful when we can assume that Alice is more likely to appear as a message recipient in a *target epoch* immediately following a message received by Bob, than she is to appear in a *random epoch* when Bob did not receive a message.

Specifically, our attack is executed as follows:

1) Create an empty table of counts; initially each user's count is zero.
2) Sample a *target epoch*. For each user that received a message during the target epoch, increase their count in the table by 1.
3) Sample a *random epoch*. For each user that received a message during the random epoch, decrease their count in the table by 1.
4) Repeat steps 2 and 3 for n target and random epochs.
5) The users in the table with the highest counts are most likely to be *associates* of the target.

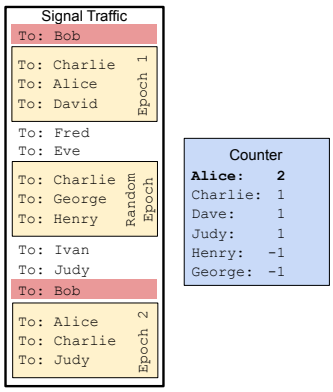

Fig. 4: **Attack Overview** — Our SDA variant has the service provider (Signal) keep count of all users who receive messages in the *epoch* after Bob receives a message to determine who is consistently messaging at the same time as Bob is receiving a message. Additionally, the service provider will begin an epoch at a random time to keep track of users which are messaging independent of the associates of Bob, and those users will be deducted from the counter. As such, "popular" users such as *Charlie* will not mask Alice's behavior.

Figure 4 gives a small example to illustrate this attack.

This is similar to the original SDA of [14], with a few of the improvements from [40] that allow for unknown recipient and background traffic distributions, more complex mixes such as pool mixes, and dummy traffic. In our setting, this means that we do not need to know *a priori* which users in the system, or which associates of the target user, are more or less likely to receive messages. We also do not need a guarantee that a reply is sent during *every* target epoch, or that the reply is always sent to the same associate Alice.

Essentially, our attack relies only on the assumptions that the distribution of background noise in each target/random epoch pair is the same, and that associates of the target are more likely to appear in target epochs than random epochs. Under only these assumptions, we can see that the expected count of any non-associate, over enough samples, is zero, while the expected count of any associate will increase linearly with the number of samples.

Compared to existing SDAs in the literature, our attack is more limited in scope: it does not attempt to model the complete distribution of all connections in the system, but merely to separate the associates from non-associates of a single target user. We also assume that the number of target and random epochs are the same (though this limitation would be easy to overcome). These limitations allow our attack to be very efficient for the attacker, who just needs to update a table for each user in each sample, and then find the largest values at the end to identify Bob's (likely) associates.

Clearly the question that remains is, how large must the number of samples n be in order for this attack to succeed? As we will see in the next section, the limited scope of our attack also makes it efficient in this sense: in reasonable settings, our

attack requires only a handful of epochs to identify the target's associates with high probability.

IV. Attack Evaluation

In this section, we evaluate our attacks from Section III first from a theoretical perspective, and second using a custom simulation.

While our attack is a variant of existing statistical disclosure attacks (SDAs) in the literature, the setting is slightly different, and our goals are more modest, seeking only to de-anonymize the contacts of a single target user.

A. Theoretical analysis of attack success

Here we provide statistical bounds to estimate the number of epochs needed for our attack to successfully de-anonymize one participant in a conversation. As before, say Bob is the target of the attack, and we wish to find which other users are communicating with Bob.

Roughly speaking, we demonstrate that (1) all users in conversations with Bob can be identified provided he is not in too many other simultaneous conversations with other users, and (2) the number of epochs needed for this de-anonymization depends logarithmically on the total number of users. These results hold under some regularity assumptions on communication patterns, which are most sensible for short periods of back-and-forth messaging.

Statistical Model. Our statistical analysis relies on the following assumptions:

1) The probability of receiving a message during any epoch is independent of receiving a message during any other epoch.
2) Each user u (both associates and non-associates) has a fixed probability r_u of receiving a message during a random epoch.
3) Any *associate* u has a fixed probability t_u of receiving a message during a target epoch, where $t_u > r_u$.
4) Every *non-associate* u has the same probability of receiving a message during a target or random epoch, i.e., $t_u = r_u$.

The last assumption states that the communications of non-associates is not correlated with the communication patterns of Bob, which makes intuitive sense, as they are not involved in conversations with Bob. The regularity (that these probabilities are fixed and the events are independent) is most reasonable when considering short attack windows, during which any user's activity level will be relatively constant.

Theoretical attack success bound. In our attack, all users in the system are ranked according to their chances of being an associate of Bob after some number of target and random epochs. We now provide bounds on the number of epochs necessary to ensure that an arbitrary associate Alice is ranked higher than *all* non-associates.

Theorem 1. *Assume m total users in a messaging system. Let Alice be an associate of the target Bob with probabilities r_a, t_a of appearing in a random or target epoch respectively. Then, under the stated assumptions above, the probability that Alice is ranked higher than all non-associates after n random and target epochs is at least*

$$1 - \frac{m}{{c_a}^n},$$

where $c_a = \exp((t_a - r_a)^2/4) > 1$ is a parameter that depends only on Alice's probabilities t_a and r_a.

The proof is a relatively standard analysis based on Hoeffding's inequality [27], and can be found in Appendix A.

We point out a few consequences of this theorem:

- The success of the attack depends only on the target user Bob and his sought-after associate Alice, not on the relative activity of any other users.
- The number of epochs needed to de-anonymize Alice with high probability scales *logarithmically* with the total number of users.
- The attack succeeds most quickly when Bob is in few other conversations (so t_a is large) and Alice is communicating mostly just with Bob (so r_a is small).

The following corollary, which results from solving the inequality of Theorem 1 and applying a straightforward union bound, gives an estimate on how many epochs are necessary to discover all of Bob's contacts with high probability.

Corollary 2. *Let $0 < p < 1$ be a desired probability bound, and assume m total users in a messaging system, of whom b are associates of a target user Bob, where the i'th associate has probabilities r_i, t_i of appearing in a random or target epoch respectively. Then, under the previous stated assumptions, with probability at least p, all b associates of Bob are correctly identified after observing*

$$\frac{4}{\min_i (t_i - r_i)^2} \left(\ln(m) + \ln(b) + \ln\left(\tfrac{1}{1-p}\right) \right)$$

target and random epochs.

Comparing to prior work, the closest SDA which has a similar theoretical bound is from Danezis [14][4]. That work makes much stronger regularity assumptions than our model, assuming essentially that (1) all epochs contain the same number of messages (2) every target epoch contains exactly one reply from Bob, (3) Bob receives a message from each associate with uniform probability, and (4) all other users, and recipients, are selected uniformly at random from all m users. Later work also includes a theoretical bound [44], but their model is much more general than ours, where they seek to reveal the entire network rather than a single target user.

B. Attack simulation

We cannot directly validate the effectivenesses of our attacks in practice, as we do not have access to Signal's servers and there is no public sample dataset of Signal sealed sender messages. Instead, we perform *simulations* based on generalized but realistic assumptions on message patterns. We do not claim our simulations will reveal the exact number

[4] Unfortunately, there appear to be at least three slightly different versions of this bound in the published literature ([14, equation (6)]; [15, equation (9.8)]; [40, page 5]), making it difficult to compare bounds.

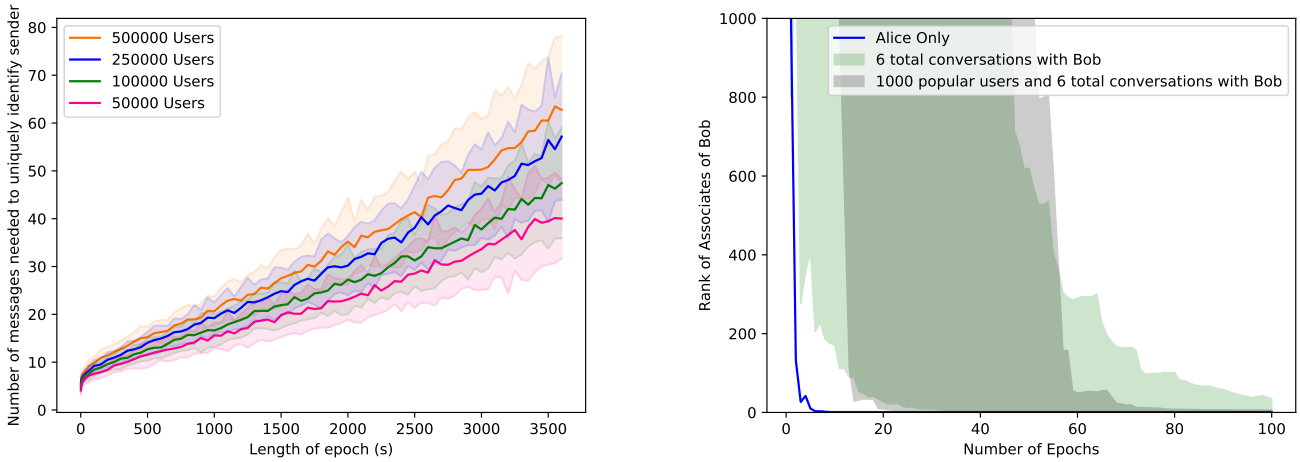

Fig. 5: **Left: Effect of delayed Read Receipts** — The attack assumes that each epoch lasts one second, and thus the log collects all delivery receipts that are sent within 1 second of Bob receiving a sealed sender message. A possible simple solution to this attack is to delay delivery receipts. We tested the effectiveness of the attack with variably sized epochs and determined that if delivery receipts were delayed a full hour (making them effectively worthless for their purpose) that with a user base of 500,000 users (each sending 50 messages a day) Bob would need to receive 60 messages from the victim user to identify Alice as the sender.
Right: Effect of popular users in our SDA — We examined the effectiveness of our SDA variant by examining the cases where only Alice is messaging Bob and where Bob is being messaged by Alice and 5 other users. The graph shows the rank of those messaging Bob, how many users have received more messages than those messaging Bob. When only Alice is messaging Bob each of the attack epochs are started by her, meaning her rank will very quickly drop. When multiple users are messaging Bob there is a range of ranks, represented by the green band which bounds the lowest ranked user messaging Bob (on the bottom) and the highest ranked individual messaging Bob (on the top). When epochs are begun by multiple users, an individual's rank takes a while to drop. The graph shows that for over 45 epochs one of the users messaging Bob has a rank of over 1000, while another user messaging Bob has dropped to a rank of 0 (meaning they have received a message after Bob received a message the most of any user in the system). The black band considers the same situation, but with 1000 popular users in the system which our variant accounts for.

of messages needed to deanonymize a particular user, as that would depend on exact messaging patterns. Rather, our simulations give a sense of the order of magnitude of messages needed to deanonymize a user.

We simulated sequences of target and random epochs (e.g. epochs where Bob does or does not receive a message) and ranked users by their score. Recall that a user's score increases if they appear in a target epoch. We simulated 1 million active users, with 800 messages per epoch. This corresponds to users sending on average about 70 messages per day, with 1 second epochs[5].

Within each epoch, we select a random set of 800 message destinations. In a target epoch, Alice (the associate) is sent a message to represent Bob's delivery receipt that would be sent to her automatically. The remaining messages are chosen randomly: 25% of messages are selected as "repeat" messages (same sender and receiver) from prior epochs (representing one side of a prior conversation), and another 25% are selected as "responses" to messages in prior epochs (representing a conversation's response). The remaining 50% of messages are messages from and to a random pairing of users from the set of 1 million active users. We find that the percent of repeats/replies has limited impact on the number of epochs to identify an associate until over nearly all messages are repeats (i.e. each epoch is essentially the same small set of senders/receivers). We choose half of the epochs to be target epochs (where Alice messages Bob) and half as random (where Alice does not message Bob).

[5]Based off our observation of round-trip delivery receipt times

Social graph significance. We note our experiment does not rely on a particular social graph (or rather, assumes a fully connected one), as any user can message any other. In preliminary experiments, we examined the impact of several different graph generators that are designed to simulate social networks, but found no noticable change in our results. Specifically, we used the Erdös-Rényi [20] model, Barabási-Albert [3] model, Watts-Strogatz [52] model, and a fully connected graph, but found they all resulted in a similar number of epochs needed to deanonymize the associate (Alice). Given this result, we opted to use the fully connected graph model for simplicity.

Figure 5 shows the result of several attack simulations. We ran each attack simulation for 100 runs, and at each epoch, report the average rank of Alice's score based on our attack. First, in the "Alice Only" variant, only Alice messages Bob (and no one else). Even though there are thousands of other users messaging randomly, Alice's score quickly becomes the top ranked user: within 5 messages, she is uniquely identified as messaging Bob.

If multiple users are also messaging Bob while Alice does, it takes more total epochs to identify Alice (and her co-associates messaging Bob). In this scenario, each target epoch is selected to be either Alice or one of 5 co-associates that messages Bob (6 total conversations with Bob).

If there are popular users present (e.g. users that receive messages in a large fraction of all epochs), then it may be more difficult to identify Alice without accounting for them. However, since we remove users that also appear in a large fraction of random epochs, Alice is still eventually ranked uniquely as messaging Bob.

Finally, we combine our popular users and multiple messagers into a single simulation, which is dominated by the multiple messagers effects.

Summary. In the worst case, it takes on the order of 60 epochs to identify the users messaging Bob. Note that only half of these are messages to Bob, and the other half are random epochs. If only one person is messaging Bob, the number of messages needed is under 5 to identify Alice as the associate of Bob.

V. Formalizing Sealed Sender Conversations

Sealed sender messages were initially introduced in Signal to obscure the communication graph. As we have just shown, the current instantiation fails to accomplish this goal. Before we present our solutions to this problem, we briefly discuss formalizations for the properties that a perfect implementation should accomplish. We call such a system a *sealed sender conversation*, because unlike sealed sender messages, the anonymity properties must be maintained throughout the lifetime of the conversation.

Our goal in introducing this formalization is to specify *exactly* how much information a service provider can learn when it runs a sealed sender conversation protocol. In a sealed sender conversation between two users, the mediating service provider should learn only the identity of the *receiver* of the first message, no matter the messaging pattern of the users. Unlike sealed sender messages, the anonymity of the sender must be maintained across the full conversation, not just individual messages. As such, we require a definition that argues about the privacy of the users at the *conversation* level, rather than at the message level, as in sealed sender messaging. We formalize sealed sender conversations by giving an *ideal functionality*, presented in Figure 6. We note that this definition fundamentally reasons over conversations, even if it does this in a message-by-message way by using an internal conversation table. Our ideal functionality captures our desired properties by specifying the maximum permissible information leakage for each message, depending on which member of the conversation sent the message.

Our ideal functionality models a sealed sender conversation and explicitly leaks certain information to the service provider. Users are able to do three things: (1) start a conversation, (2) send messages in an existing conversation, and (3) receive messages. When a user starts a new conversation, the initial receiver's identity is leaked to the service provider, along with a unique conversation identifier `cid`. All subsequent messages sent in this conversation are linked with the identifier `cid`. If they are being sent to the initial receiver, their destination is leaked. Otherwise, the service provider learns that the message is part of some known `cid`, but never learns the identity of that end of the conversation. While we do not explicitly include timestamps in our modeling, timestamps are implicitly captured by our model because the service provider is notified immediately whenever the ideal functionality receives a message. This is equivalent because the absolute time at which a message is sent is not important in our context, just the relative time between messages.

Users receive messages via pull notifications. These pull notifications leak no more information than the message itself does; if the receiver is anonymous, then the pull notification process leaks no information about the receiver. While we formalize this notion using pull notifications, this is compabile with Signal-style push notifications, where the receiver and the server maintain long-lived TLS connections. These communication channels are equivalent to a continuous pull notification, and thus a simulator can easily translate between the two communication paradigms. Finally, because the service provider may arbitrarily drop messages, we give the service provider the power to approve or deny any pull notification request.

While leaking the conversation identifier might seem like a relaxation of sealed sender messages, we note that our timing attack succeeds by guessing with high likelihood the sender of a message. As such, Signal's sealed sender does not meet this ideal functionality, as our timing correlation attack in Section III shows. This is because the `cid` of a message, although not explicitly sent with the ciphertext, can be inferred with high probability by its timing. One final note is our definition does not prevent a service provider from using auxiliary information about a conversation (*e.g.* time zone information) to reidentify the initiator of the conversation. Such attacks are incredibly difficult to formalize and are beyond the scope of our work. Rather, we only require that the *protocol* itself cannot be used to reidentify the participants.

A. Security Definition for One-Way Sealed Sender Conversations

We now give a formal definition for one-way sealed sender conversations using a simulation based security definition. We present the ideal functionality for one-way sealed sender conversations in Figure 6. Importantly, this definition does not rule out learning information about the sender based on timing of sending messages, *e.g.* the sender's time zone. We model the service provider as a party P_{service} that can control delivery of messages and delivery receipts. Note that the ideal functionality leaks the contents of the message m to the service provider only if the receiver of that message is corrupted. This models that if the service provider can decrypt the messages it is relaying, it may make delivery decisions based on knowledge of the plaintext.

We say that a protocol securely realizes this ideal functionality (in the stand alone model) if a corrupted service provider and an arbitrary number of corrupted users cannot determine if they are interacting in the *real experiment* or with the *ideal experiment* with non-negligible probability in the security parameter λ. In the real experiment, the adversary starts by statically corrupting the service provider and any number of users. Then, each honest user follows it own arbitrary strategy, interacting with the service provider using the protocol. The corrupt parties can follow an adversarially chosen strategy. In the ideal experiment, the adversary again begins by statically

Ideal Functionality For Sealed Sender Conversation System

- $P_1, \ldots, P_n$: A set of n (possibly corrupt) users of the system
- P_{service}: A single corrupt service provider that is in charge of relaying messages between users
- Active Conversation Table C_{active} with entries of the form $(\texttt{convo-id}, \texttt{initiator}, \texttt{receiver})$, Delivery Pending Message Table M_{pending} with entries of the form $(\texttt{convo-id}, \texttt{sender}, \texttt{receiver}, \texttt{plaintext})$

Start Conversation: Upon receiving a message $(\mathsf{StartConvo}, P_j)$ from a user P_i, the ideal functionality generates a unique identifier $\texttt{cid}$, and performs the following:

- If P_i or P_j is corrupt, send $(\mathsf{ApproveNewConvoCorrupt}, P_i, P_j, \texttt{cid})$ to P_{service}
- If both P_i and P_j are honest, $(\mathsf{ApproveNewConvo}, P_j, \texttt{cid})$ to P_{service}

P_{service} responds to either message with $(\mathsf{Approve})$ or $(\mathsf{Disapprove})$

- If P_{service} responds with $(\mathsf{Disapprove})$, the ideal functionality halts
- If P_{service} responds with $(\mathsf{Approve})$, the ideal functionality sends $(\mathsf{NewConvo}, P_i, P_j, \texttt{cid})$ to both P_i and P_j and adds $(\texttt{cid}, P_i, P_j)$ to C_{active}.

Send Message: Upon receiving a message $(\mathsf{SendMessage}, \texttt{cid}, m)$ from party P_i, the ideal functionality checks the active conversations table C_{active} for an entry $(\texttt{cid}, P_j, P_i)$ or $(\texttt{cid}, P_i, P_j)$. If no such entry exists, the ideal functionality drops the message. The ideal functionality generates a unique identifier $\texttt{mid}$ and performs the following:

- If there is an entry and P_j is corrupted, the ideal functionality sends $(\mathsf{NotifySendMessageCorrupt}, \texttt{cid}, \texttt{mid}, m, P_i, P_j)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.
- If an entry $(\texttt{cid}, P_i, P_j)$ exists, send $(\mathsf{NotifySendMessage}, \texttt{cid}, \texttt{mid}, P_j, |m|)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.
- If an entry $(\texttt{cid}, P_j, P_i)$ exists, send $(\mathsf{NotifyAnonymousSendMessage}, \texttt{cid}, \texttt{mid}, |m|)$ to P_{service}, and add $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ to M_{pending}.

Receive Message: Upon receiving a message $(\mathsf{ReceiveMessage}, \texttt{cid})$ from party P_j, the ideal functionality checks C_{active} for an entry $(\texttt{cid}, P_j, P_i)$ or $(\texttt{cid}, P_i, P_j)$. If such an entry exist, it performs one of the following:

- If P_i is corrupt, the ideal functionality then sends $(\mathsf{ApproveReceiveMessageCorrupt}, \texttt{cid}, P_i, P_j)$ to P_{service}, which responds with tuples of the form $(\texttt{cid}, P_i, P_j, m)$. The ideal functionality then sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j for each such tuple.
- If there is an entry $(\texttt{cid}, P_j, P_i)$ in C_{active} and entries $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ in M_{pending}, the ideal functionality sends $(\mathsf{ApproveAnonymousReceiveMessage}, \texttt{cid}, \texttt{mid}, |m|)$ to P_{service} for each such entry. P_{service} responds to each message with either $(\mathsf{Approve}, \texttt{mid})$ or $(\mathsf{Disapprove}, \texttt{mid})$. If P_{service} responds with $(\mathsf{Approve}, \texttt{mid})$, the ideal functionality sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j.
- If there is an entry $(\texttt{cid}, P_i, P_j)$ in C_{active} and entries $(P_i, P_j, \texttt{cid}, \texttt{mid}, m)$ in M_{pending}, the ideal functionality sends $(\mathsf{ApproveReceiveMessage}, \texttt{cid}, \texttt{mid}, |m|, P_j)$ to P_{service} for each such entry. P_{service} responds to each message with either $(\mathsf{Approve}, \texttt{mid})$ or $(\mathsf{Disapprove}, \texttt{mid})$. If P_{service} responds with $(\mathsf{Approve}, \texttt{mid})$, the ideal functionality sends $(\mathsf{Sent}, P_i, P_j, \texttt{cid}, m)$ to P_j.

Fig. 6: Ideal functionality formalizing the leakage to the service provider for a one-way sealed sender conversation.

corrupting the service provider and any number of users. Then, the honest players follow an arbitrary strategy but interact directly with the ideal functionality. The service provider and corrupted users interact with a simulator Sim, which mediates interaction between the adversary and the ideal functionality. At the end of each experiment, a distinguisher algorithm takes in the views of the service provider and the corrupted parties and attempts to determine if the interaction was in the real experiment or the ideal experiment. Note that because the simulator may not know which parties are interacting, it cannot leak this information to the adversary.

We denote the output of the ideal world experiment for any ideal world adversary Sim and honest players with arbitrary strategies P_H on inputs x as $\mathsf{Ideal}_{P_H,\mathsf{Sim}}(1^\lambda, x)$. We denote the output of the real experiment with adversary $\mathcal{A}$ running protocol Π on input x as $\mathsf{Real}_{P_H,\mathcal{A},\Pi}(1^\lambda, x)$. We say that a protocol Π securely realizes the ideal functionality described in Figure 6 if there exists a simulator Sim such that

$$\left|\mathsf{Ideal}_{P_H,\mathsf{Sim}}(1^\lambda, x) - \mathsf{Real}_{P_H,\mathcal{A},\Pi}(1^\lambda, x)\right| < \mathbf{negl}(\lambda)$$

VI. Solutions

We now present three protocols that follow the security definition from Section V and, in particular, prevent the attacks presented in Section III. We first outline a *one-way* sealed sender conversation in Section VI-B, in which the initiator of the conversation remains anonymous. We prove that our construction meets the definition presented in Section V-A. In Section VI-C, we extend this protocol to give better privacy to the receiver using a *two-way* sealed sender conversation. Finally, in Section VI-D, we address denial of service attacks that malicious users could launch against the server.

Overview of Solutions. Our key observation is that the attack described in Section III is only possible because both users in a conversation are sending messages to the other's long-term identity. Over time, these messages can be correlated, revealing the identities of the users. On the other hand, if *anonymous* and *ephemeral* identities are used instead, then user's true identities can remain hidden. However, anonymous identities lead to a bootstrapping problem: *how do users initiate and authenticate a conversation if they are using fresh, pseudonyms?*

In a *one-way sealed sender conversations*, the identity of one side of the conversation is leaked, namely the initial message receiver, in order to solve this bootstrapping problem. This closely models the situation of a whistle-blower, where the informant wishes to stay anonymous, but the reporter receiving the information can be public. At a high level, the initiator of the conversation begins by creating a fresh, anonymous identity and then sends this identity to a receiver via a normal sealed sender message (thus solving the bootstrapping problem). The conversation proceeds with the initiator of the conversation sending messages to the receiver using sealed sender (one way), and the conversation receiver sending replies to the initiator's anonymous identity. Importantly, the identity of the initiator is never leaked, as no messages exchanged in the conversation contain that person's long-term identity. We prove that out protocol securely realizes the definition of sealed sender conversations presented in Section V-A.

A straightforward extension is to move towards *two-way sealed sender conversations* where both parties use anonymous identities. This solution is described in Section VI-C. When an initiator starts a conversation as described above, the receiver also creates a new anonymous identity and sends it via sealed sender back to the conversation initiator. This protocol offers a single opportunity to link the receiver to their new, anonymous identity (by correlating the timing of the received message and the registering of a new public key), but, as we have shown, network noise makes it difficult to re-identify users with only a single event. Even in the unlikely case that the conversation receiver is linked to their long-term identity, we show that the conversation initiator remains anonymous.

Both protocols place the service provider at risk of denial of service attacks, and so in Section VI-D, we aim to limit the power of users to arbitrarily register anonymous identities. Allowing users to create unlimited anonymous identities would lead to strain on the service provider if there is no way to differentiate between legitimate anonymous identities and malicious ones. To prevent these attacks, users are each given a limited number of anonymous credentials that they can "spend" to register anonymous keys, reminiscent of the earliest e-cash systems [8]. These credentials can be validated by the service provider to ensure that a legitimate user is requesting an anonymous identity without revealing that user's identity. We use blind signatures to implement our anonymous credentials. We evaluate the practicality of this approach in Section VI-E and show that it could be deployed cheaply for either one-way or two-way sealed sender conversations.

For simplicity, we assume that communicating users have already exchanged delivery tokens. Any protections derived from these delivery tokens can be added to the following protocols in a straightforward manner. Additionally, we assume users connect to the service provider via an anonymous channel, e.g., Tor or Orbot.

A. Preliminaries

Sealed Sender We assume that the service provider implements the sealed sender mechanism described in Section II-A. Specifically, we assume that a client can generate a public/private key pair and publish their public key as an address registered with the service. If the server permits it through some verification process, the server will allow messages to be sent to that public key without a sender.

More formally, we assume that the system has a sealed sender encryption scheme Π_{ssenc}. While Signal does not give a proof of security for the scheme it uses, for our constructions we will assume that Π_{ssenc} is a signcryption scheme that satisfies ciphertext anonymity [35] and adopt the notation presented in [51] for its algorithms[6]. We say a sealed sender encryption scheme Π_{ssenc} is a set of three algorithms:

- $\mathsf{SSKeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ generates a public/private key pair.
- $\mathsf{SSEnc}(m, \mathsf{sk}_s, \mathsf{pk}_r) \to c$ takes in a message m, the sender's secret key sk_s and the receiver's public key pk_r, and outputs a ciphertext c
- $\mathsf{SSDecVer}(\mathsf{sk}_r, c) \to \{(m, \mathsf{pk}_s), \perp\}$ takes in the receiver's private key sk_r and a ciphertext c and either outputs a message m,and the public key of the sender pk_s, or returns the error symbol $\perp$. (Note that this actually constitutes decryption followed by verification in the notation of [51], returning $\perp$ when either step fails.)

Formal security definitions are given in [51]. In short, the scheme satisfies (1) message indistinguishability, (2) unforgeability, and (3) ciphertext anonymity, meaning the ciphertext reveals nothing about the sender or receiver.

Blind Signatures The mechanism to prevent abuse for the creation of anonymous accounts relies on the cryptographic primitive of *blind signatures*, as first proposed by [8]. Blind signature schemes have 5 algorithms: BSKeyGen, BSBlind, BSSign, BSExtract and BSVerify. BSBlind takes in the public key of the signer, a message, and some randomness and outputs a blinded message. BSSign takes in the signer's private key and a blinded message and outputs a blinded signature. BSExtract takes in a blinded signature and the randomness used in blinding and outputs a normal signature. Finally, BSVerify takes in a message and the signer's public key and decides if the signature is valid.

The interaction between a server with the signing keypair sk, pk and a client is as follows:

1) Client generates the blinded message $b \leftarrow \mathsf{BSBlind}(m, pk; r)$ for $r \leftarrow_\$ \{0,1\}^\lambda$
2) Client sends b to the server for signing.
3) Server computes the blinded signature $s_{blind} \leftarrow \mathsf{BSSign}(b, sk)$ and returns it to the client.
4) Client extracts the real signature $s \leftarrow \mathsf{BSExtract}(s_{blind}, pk; r)$
5) Client, in a different network connection, sends the initial message m and the real signature s to the server, who runs $\mathsf{BSVerify}(pk, m, s)$

The blind signature scheme should have the usual signature unforgeability property. Additionally, it should be impossible for a server to link the blinded message and blinded signature to the real message and real signature. We use the RSA-based construction of blind signatures from [8].

[6]We note that ciphertext anonymity is actually a stronger primitive than required, as there is no need for receiver anonymity.

B. One-way Sealed Sender Conversations

First, we provide the construction of sealed sender conversations which we build on in this solution and those that follow. Recall that a sealed sender conversation reveals the flow of the conversation (including message timing, etc.) and the identity of the initial receiver, but at no point can the service provider identify the initial sender.

The intuition behind our solution is straightforward: when initiating a new conversation, a sender generates an ephemeral, per-conversation key pair. This key pair is registered with the service provider anonymously, but otherwise is treated as a normal identity in the system. Throughout the lifetime of the conversation, this identity key is used instead of the long-term identity of conversation initiator. As long as the ephemeral public key is never associated with the long-term identity, and never used in any other conversations, the service provider cannot learn anything about the true identity of the user that generated that ephemeral identity.

Generally, the flow of a sealed sender conversation is as follows. During the setup, each sender P_s with long-term keys $(\mathsf{pk}_s, \mathsf{sk}_s)$ creates entries $(P_r, \mathsf{pk}_r, \mathsf{pk}_s)$ for each receiver P_r with public key pk_r. Some user, who we call the initiator, starts the conversation by running the **Initiate Conversation** protocol below where P_s generates and registers an ephemeral identity for a receiver P_r. Whenever the receiver comes online (or possibly immediately by receiving a push notification) and receives the appropriate information, they will locally associate the ephemeral key with the initiator for the duration of the conversation. From this point, both users may send messages using the **Send Message** protocol and receive those messages from the service provider via **Push Message**, over an open, long-term connection. The protocol **Open Receiver Connection** is used to establish a channel for such push notifications, either for a user's long-term mailbox, or for an ephemeral mailbox created for a single conversation.

Every user must maintain a *conversation table*, to remember where messages should be sent in all ongoing conversations. Each table entry stored by a user P_s is a tuple $(P_r, \mathsf{pk}_\beta, \mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$, where P_r is the actual message recipient, pk_β is the recipient's mailbox (public key) to which the message is addressed, and $(\mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$ is the key pair used to sign and encrypt the message. Depending on who initiated the conversation, one of pk_β or pk_α will correspond to an ephemeral identity pk_e, and the other will correspond to one of the long-term identities pk_r or pk_s.

Initiate One-Way Sealed Conversation to P_r:

1) Initiator P_s does the following:
   a) looks up P_r's long-term public key pk_r
   b) generates fresh ephemeral keys $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$
   c) encrypts $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{``init''} \| \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$
   d) connects to the service provider anonymously and sends $c \| \mathsf{pk}_e$ for pk_r
   e) appends $(P_r, \mathsf{pk}_r, \mathsf{pk}_e, \mathsf{sk}_e)$ to the conversation table
   f) Registers a new mailbox for the public key pk_e and uses **Open Receiver Connection** with keypair public key $\mathsf{pk}_e, \mathsf{sk}_e$ to establish a connection for push notifications.
2) The service provider delivers c (sealed sender) to P_r based on pk_r, either immediately pushing the message or waiting for the receiver to come online.
3) When the receiver P_r receives the message to its long-term mailbox pk_r, it:
   a) decrypts and verifies $(\texttt{``init''} \| \mathsf{pk}_e, x, \mathsf{pk}_s) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_r, c)$
   b) appends $(P_s, \mathsf{pk}_e, \mathsf{pk}_r, \mathsf{sk}_r)$ to the conversation table
   c) uses **Send Message** to send a delivery receipt to P_s (which now goes to pk_e from the conv. table)

Send Message to P_*

1) Sender looks up freshest entry $(P_*, \mathsf{pk}_\beta, \mathsf{pk}_\alpha, \mathsf{sk}_\alpha)$ in the conversation table.
2) Sender encrypts $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m, \mathsf{sk}_\alpha, \mathsf{pk}_\beta)$
3) Sender sends c for pk_β to the service provider, anonymously if necessary.
4) If there is an open connection associated with pk_β, the service provider uses **Push Message** for c over that connection. Otherwise, the service provider sets the message as pending in the mailbox associated with pk_β

Open Receiver Connection for $(\mathsf{pk}_\beta, \mathsf{sk}_\beta)$

1) Receiver connects to the service provider and demonstrates knowledge of key pair $(\mathsf{pk}_\beta, \mathsf{sk}_\beta)$ such that there is a registered mailbox for public key pk_β
2) The receiver and the server build a long-term connection for message delivery, indexed by pk_β
3) If there are any pending messages in the mailbox associated with pk_β, use **Push Message** for those messages.

Push Message c to pk_β

1) Service provider looks up an open connection indexed by pk_β. If such a connection exists, the service provider sends c over it
2) Receiver decrypts c as $(m, \mathsf{pk}_\alpha) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_\beta, c)$ and verifies an entry $(P_*, \mathsf{pk}_\alpha, \mathsf{pk}_\beta, \mathsf{sk}_\beta)$ exists in the conversations table, dropping it otherwise.

We prove that this construction securely realizes the definition Figure 6 in the standalone model in Appendix B. The proof is straightforward: we construct a simulator and show that an adversary corrupting the service provider and any number of clients cannot distinguish between the real protocol and interacting with the ideal functionality.

C. Two-way Sealed Sender Conversations

While the construction above successfully realizes sealed sender conversations, the identity of the receiver is still leaked to the service provider. Ideally, we would like for both users in a conversation to communicate using only ephemeral identities, so that the service provider sees only the flow of messages in a conversation but does not learn either party's long-term identity. However, this again leads to a bootstrapping problem: if both users use fresh, anonymous identities, *how do they exchange this ephemeral contact information while remaining anonymous?*

While heavyweight cryptography (such as PIR or ORAMs) may provide a more robust solution, in this work we focus on scalable solutions that might plausibly be adopted by secure messaging platforms. As such, we present a natural extension of our one-way sealed sender conversation protocol.

After an initiator creates an ephemeral key pair, opens a new mailbox, and sends this to the receiver, the receiver responds by doing the same thing: creating a second ephemeral key pair, opening a second mailbox, and sending this back to the initiator as part of the initial delivery receipt. After this, *both* the conversation initiator and receiver will have conversation table entries of the form $P_*, \mathsf{pk}_{e1}, \mathsf{pk}_{e2}, \mathsf{sk}_{e2}$, with two different ephemeral keys for sending and receiving messages in the conversation.

This requires minimal changes to the previous protocol. Essentially, the **Initiate** protocol gains another section for the recipient to create their own ephemeral identity, but the **Send**, **Open Connection**, and **Push Message** protocols are identical. In Appendix C we provide the full details of these updated protocols, along with an additional protocol **Change Mailbox** which is used to update an ephemeral key pair for one side of an existing conversation.

Security. We have two security goals for this protocol. First, we require that this protocol is a secure instantiation of a one-way sealed sender conversation, just like the protocol above. This is clear, as the only party whose behavior changes from the protocols in Section VI-B is the initial receiver. Simulating their behavior is easy because that user's identity is already leaked by the ideal functionality. As such, the proof remains nearly identical to that in Appendix B.

Second, we require that the service provider has only one chance to identify the initial receiver. Note that besides the initial messages, all sent messages are only linked to the anonymous identities. Thus, no information about the users' true identities are leaked by these messages. This only source of information about these identities comes from the timing of the mailbox's initial opening, so this is the only chance to identify the initial receiver. As described in our simulations, in a reasonably busy network it is difficult to link two events perfectly. Instead, it requires many epochs of repeated behavior to extract a link. Therefore, giving the service provider only a single chance to de-anonymize the receiver will most likely (*though not provably*) provide two-sided anonymity. To further decrease the chance of a successful attack, the initial receiver can introduce some initial random delay in opening and using a new mailbox.

Obscuring the Conversation Flow. A natural generalization of this approach is to switch mailboxes often throughout a conversation, possibly with *each message*. This may provide further obfuscation, as each mailbox is only used once. While analyzing *how well* this approach would obscure the conversation flow is difficult, as linking multiple messages together requires the service provider to find a timing correlation between the various mailboxes' activities, it is clear it provides no worse anonymity than the above construction.

D. Protecting against Denial of Service

Both constructions presented above require users to anonymously register public keys with the service provider. This provides an easy way for attackers to launch a denial of service attack: simply anonymously register massive numbers of public keys. As such, we now turn our attention to bounding the number of ephemeral identities a user can have open, without compromising the required privacy properties.

We build on *anonymous credential* systems, such as [8]. Intuitively, we want each user in the system to be issued a fixed number of anonymous credentials, each of which can be exchanged for the ability to register a new public key. To implement this system, we add two additional protocols to those presented above: **Get signed mailbox key** and **Open a mailbox**.

In **Get signed mailbox key**, a user P_s authenticates to the service provider with their long-term identity pk_s and uses a blind signature scheme to obliviously get a signature σ_{es} over fresh public key pk_{es}. We denote the service provider's keypair $(pk_{\text{sign}}, sk_{\text{sign}})$. In **Open a mailbox**, a user P_s anonymously connects to the service provider and presents $(\mathsf{pk}_{es}, \sigma_{es})$. If σ_{es} is valid and the service provider has never seen the public key pk_{es} before, the service provider opens a mailbox for the public key pk_{es}. These protocols are described below:

Get signed mailbox key

1) User authenticates using their longterm public key. Server checks that the client has not exceeded their quota of generated ephemeral identities.
2) Client generates $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$
3) Client blinds the ephemeral public key $b \leftarrow \Pi_{\text{bs}}.\mathsf{BSBlind}(\mathsf{pk}_e, pk_{\text{sign}}; r)$ with $r \leftarrow \{0,1\}^\lambda$.
4) Server signs the client's blinded public key with $s_{blind} \leftarrow \Pi_{\text{bs}}.\mathsf{BSSign}(b, sk_{\text{sign}})$ and returns the blinded signature to the client.
5) Client extracts the real signature locally with $\sigma_e \leftarrow \Pi_{\text{bs}}.\mathsf{BSExtract}(s_{blind}, pk_{\text{sign}}; r)$

Open a mailbox

1) Client connects anonymously to the server and sends pk_e, σ_e
2) Server verifies $\Pi_{\text{bs}}.\mathsf{BSVerify}(pk_{\text{sign}}, \sigma_e) = 1$ and checks pk_e has not been used yet.
3) Server registers an anonymous mailbox with key pk_e with an expiration date.

Integrating these protocols into one-way sealed sender conversations and two-way sealed sender conversations is straightforward. At the beginning of each time period (*e.g.* a day), users run **Get signed mailbox key** up to k times, where k is an arbitrary constant fixed by the system. Then, whenever a user needs to open a mailbox, they run the **Open a mailbox** protocol. Sending and receiving messages proceeds as before.

It is important that (1) the signing key for the blind signature scheme public key pk_{sign} be updated regularly, and (2) anonymous mailboxes will eventually expire. Without these protections, malicious users *eventually* accumulate enough anonymous credentials or open mailboxes that they can effectively launch the denial of service attack described above.

Network Conditions	ECDSA KeyGen	Get Signed Mailbox Key	Open a Mailbox
End-to-End	0.049	0.061	0.039
User Local	0.049	0.032	0.024
Server Local	N/A	0.013	0.001

TABLE I: Timing results (in seconds) for protocols of Section VI-D, using RSA-2048 ciphertexts and ECDSA.

Additionally, each time period's pk_{sign} must be known to all users; otherwise the server could use a unique key to sign each user's credentials, re-identifying the users.

E. Blind Signature Performance

To test the feasibility of using blind signatures, we implemented the protocols in Section VI-D for a single client and server. This represents the *cryptographic overhead* of applying our solution, as the remainder (sending and receiving messages, registering keys) are services already provided by Signal.

The networking for both the client and server are written in Python, with the Django web framework [1] on the server. Starting with the code provided in [4], we implement an RSA-2048 blind signature [8] library in Java that can be called via RPC. Although RSA ciphertexts are large, they are very fast to compute on modern hardware.

We evaluated our implementation by running the server on an AWS instance with 2 Intel Xeon processors and 4 GB of RAM. The client was running on a consumer-grade laptop, with a 2.5 GHz Intel i7 with 16 GB of RAM, located in the same region as the AWS server. We report the timing results in Table I for each protocol. To better isolate the overhead from network delay, we also report the execution time when server and client are running locally on the same machine.

Importantly, ECDSA KeyGen can be run in the background of the client, long before the interactive phase of the protocol starts. For maximum security, a user may close an old mailbox and get a new signed key (with the same anonymous connection), and then open a new mailbox with each message that they send. This incurs an overhead of less that 100ms, even including network delay. The communication overhead of running this full protocol is less that 1KB, constituting 3 RSA-2048 ciphertexts and 1 ECDSA public key.

F. Deployment Considerations

Key Rolling. It is critical that the server maintain a database of ephemeral identities previously registered on the system in order to check for re-use of old ephemeral identities. Note that to prevent reuse, this database must be maintained for as long as the identities are valid and grows with the number of mailboxes, not the number of users.

We suggest that Signal update their mailbox signing key at regular intervals, perhaps each day, and leave two or three keys valid for overlapping periods of time to avoid interruptions in service. Because the validity of a signed mailbox key is tied to the signing key, each update allows the server to "forget" all the keys that it saw under the old signing keys as they cannot be reused.

Mailbox Opening. It is important that users perform **Get signed mailbox key** (where Signal learns a user's identify) and **Open a mailbox** in an uncorrelated way. Otherwise, Signal could link the two and identify the anonymous mailbox. We recommend performing **Get signed mailbox key** at regular intervals (e.g. the same time each day), but careful consideration must be taken for users that are offline during their usual time. Users should not come online and perform both operations immediately if sending to a new conversation. To avoid this, clients should maintain a small batch of extra signed mailbox keys for new conversations.

Cost Overhead. We analyze the worst case cost of scaling our protocol. We generously assume that 10 million anonymous mailboxes will be opened every day. The server's part of opening these mailboxes constitutes calls to BSVerify and BSSign and a database query (to check for repeated identities). In our experiments, the two blind signature operations, including the Django networking interface, took a cumulative .014 seconds. Using AWS Lambda, supporting 10 million messages each day would cost approximately $10 per month. We estimate that doing 10 million reads and writes a day to a DynamoDB database would cost approximately $20 per month, using AWS's reserved capacity pricing.

Using the key rolling scheme described above, the database contains at most the number of messages delivered in a day times the number of simultaneously valid keys. At 10 million messages each time with a two overlapping valid keys, this means the database would contain at most 20 million ephemeral identities. Assuming 256-bit identity values, the entire database would never exceed a few GB of data. Therefore, we conservatively estimate that the marginal cost of supporting our protocol for 10 million ephemeral identities per day would be under $40 per month. We note our analysis does not consider the personnel cost associated with developing or maintaining this infrastructure. Ideally, this would be amortized along with Signal's existing reliability and support infrastructure.

VII. Discussion

A. Other solutions

In this section, we consider alternative, *minor changes* to the existing sealed sender protocol and evaluate their effectiveness.

Random delays. Users could send delivery or read receipts after a random delay, making it harder for attackers to correlate messages. This forces an attacker to increase the *epoch duration* to perform the same attack. We analyze the effect of varying epoch duration in Figure 5, and find that even with hour-long epochs—likely rendering delivery receipts useless—users could still be identified within 60 messages. We conclude that injecting random delays is an ineffective way to achieve anonymity.

Cover traffic. Users could send random sealed-sender messages that are transparently ignored by the recipient in order

to cover for the true pattern of ongoing conversations. Based on our experiments, we again see that cover traffic slows down our attack, but at a linear rate with the amount of extra traffic: even with 10x extra messages, the anonymity set of potential senders to Bob after 100 messages is under 1000 users. This mitigation strategy has obvious costs for the service provider, without significant benefit to user anonymity.

Disable automatic receipts. While Signal users can disable read receipts and typing notifications, they currently cannot turn off delivery receipts. Adding an option for this would give users the choice to greatly mitigate this attack. We note disabling would have to be mutual: Alice turning off delivery receipts should also prevent Bob from sending them, different from how Signal currently disables read receipts. We also note users could potentially still be linked purely by their messages eventually, making this only a partial mitigation.

B. Drawbacks and Likelihood of Adoption

We believe that the solution we have proposed in Section VI is both practical and cost-effective. However, there are a few drawbacks. Most importantly, it adds complexity to the system, and complexity always increases the likelihood of error and vulnerability. In particular, the key rolling scheme we suggest in Section VI-F requires increased complexity in the back-end key management system. While the compromise of these keys would not leak message content, it could allow for a cheap resource denial attack on Signal.

A second important drawback of our solution is the assumption that a malicious service provider cannot use network information to identify users. As mentioned, using Tor [19], [2] would address this, but only if enough users did so to increase the anonymity set.

Finally, our ephemeral identities may increase complexity for users that use Signal on multiple devices. Signal would need to securely share or deterministically generate these keys with other devices in a privacy-preserving way.

Given the limited scope and impact of these drawbacks, we believe that is reasonable to believe that Signal or other secure messengers could potentially adopt our solution.

C. Group messaging

The OTR and Signal protocols were first designed for pairwise communication, and we have focused on such conversations in this work. However, group messaging is an important use case for private messaging services, and has recently shown to be vulnerable to different kinds of attacks [46], [10], [47].

An interesting direction for future work would be to extend our attacks to this setting. It is clear that received receipts and read receipts do not work the same way in groups as they do for two-way conversations. On the other hand, group messages have additional *group management messages* which are automatically triggered, for example, when a new member attempts to join the group. It would be interesting to understand if, for example, our attack could exploit these message to de-anonymize *all* members of a given group chat.

Fortunately, it does seem that our main solution proposed in Section VI would be applicable to the group chat setting: all members of the group chat would create new, anonymous mailboxes used only for that particular group. However, this would still leave the difficulty of the initial configuration and key management, which would be more complicated than that two-party setting. We consider this to be important and useful potential future work.

VIII. Related Work

Attacks on mobile messaging. Mobile messaging services have been hugely popular for decades, but the SMS protocol was designed primarily for efficiency and not with privacy in mind [26]. Usability studies have shown that many users want or even assume that their text messages are private [24], which has made SMS a "Goldmine to exploit" for state surveillance [5], [21]. Even encrypted alternatives to SMS are still targeted by hackers and state-level surveillance tools, as seen for example by the NSO group's Pegasus spyware, which was used to target the text messages of journalists and politicians in multiple countries [38].

Statistical disclosure attacks. SDAs were first proposed as an attack on mix networks by [14], and later strengthened to cover more realistic scenarios with fewer or different assumptions [40], [37], [16]. More recent variants consider the entire network, and attempt to learn as much as possible about all sender-receiver correlations over a large number of observations [17], [44], [30]. See [43] for a nice overview and comparison of many existing results.

Private messaging. Perhaps in response to these highly-publicized attacks, third-party applications which provide end-to-end encrypted messaging, such as WhatsApp (since 2016), Telegram, and Signal, are rapidly gaining in popularity [32]. A good overview for the interested reader would be the SoK paper of Unger et. al. from 2015 [49].

The first cryptographically sound, scalable system for end-to-end encrypted messaging is the OTR protocol from 2004 [6], which had significant influence on the popular systems used today [39], [22], [9].

Since OTR, significant research has investigated how to remove or hide metadata to provide anonymous chat applications. Indeed, similar problems have been noted in mix-nets [34]. Many such as Ricochet [7] rely on Tor [19]. Other techniques for obscuring metadata are injecting noise, like Pond [33] and Stadium [48], or decentralization [31]. Many of these solutions require sharing cryptographic identities out-of-band, rather than build off human-friendly or already known identities.

DC-net based messengers like Dissent [12] or Verdict [13] have also been proposed, but suffer problems in scaling to the number of users seen on popular messaging applications [49], [50]. Others such as Riposte [11] have made use of private information retrieval to achieve anonymity, but this is also expensive in practice. We focus on sealed sender in this paper, as it is the most widely-deployed in practice attempt to provide sender anonymity in secure messaging.

IX. Conclusion

In this work we analyze and improve upon Signal's sealed sender messaging protocol. We first identify a type of statistical disclosure attack (SDA) that would allow Signal to identify who is messaging a user despite sealed sender hiding message sources. We perform a theoretical and simulation-based analysis on this attack, and find that it can work after only a handful of messages have been sent to a user. Our attack is possible because of two features of the sealed sender protocol: (1) metadata (specifically, recipient and timing) is still revealed, and (2) Signal sends automatic delivery receipts back to the sender immediately after a message is received.

We suggest a protection against this attack, in which users anonymously register ephemeral mailbox identities with Signal, and use those to communicate rather than long-term identities such as phone numbers. To prevent abuse, we suggest Signal use anonymous credentials, implemented with blind signatures, and implement a prototype that demonstrates our solution is performant and cost-effective to deploy.

Signal has taken a first step into providing anonymous communication to millions of users with the sealed sender feature. Signal's design puts practicality first, and as a result, does not provide strong protection against even known disclosure attacks. Nonetheless, we believe this effort can be improved upon without sacrificing practicality, and we hope that our work provides a clear path toward this end.

References

[1] "Django," https://www.djangoproject.com/, accessed: 2019-11-21.

[2] "Orbot: Tor for Android," https://guardianproject.info/apps/orbot/, 2020.

[3] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[4] A. Athanasiou, "Blind-RSA," https://github.com/arisath/Blind-RSA, accessed: 2019-11-21.

[5] J. Ball, "NSA collects millions of text messages daily in 'untargeted' global sweep," *The Guardian*, Jan 2014. [Online]. Available: https://www.theguardian.com/world/2014/jan/16/nsa-collects-millions-text-messages-daily-untargeted-global-sweep

[6] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use PGP," in *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '04. New York, NY, USA: ACM, 2004, pp. 77–84. [Online]. Available: http://doi.acm.org/10.1145/1029179.1029200

[7] J. Brooks *et al.*, "Ricochet: Anonymous instant messaging for real privacy," 2016.

[8] D. Chaum, "Blind signatures for untraceable payments," 1982, pp. 199–203.

[9] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the Signal messaging protocol," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, April 2017, pp. 451–466.

[10] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On Ends-to-Ends Encryption: Asynchronous Group Messaging with Strong Security Guarantees," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 1802–1819.

[11] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An Anonymous Messaging System Handling Millions of Users," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 321–338.

[12] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable Anonymous Group Messaging," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. ACM, 2010, pp. 340–350.

[13] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, "Proactively accountable anonymous messaging in verdict," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 147–162.

[14] G. Danezis, "Statistical Disclosure Attacks," in *Security and Privacy in the Age of Uncertainty*. Springer US, 2003, pp. 421–426.

[15] ——, "Better anonymous communications," Ph.D. dissertation, University of Cambridge, 2004.

[16] G. Danezis, C. Diaz, and C. Troncoso, "Two-sided statistical disclosure attack," in *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2007, pp. 30–44.

[17] G. Danezis and C. Troncoso, "Vida: How to use bayesian inference to de-anonymize persistent communications," in *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2009, pp. 56–72.

[18] M. Di Raimondo, R. Gennaro, and H. Krawczyk, "Secure off-the-record messaging," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '05. New York, NY, USA: ACM, 2005, pp. 81–89. [Online]. Available: http://doi.acm.org/10.1145/1102199.1102216

[19] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251396

[20] P. Erdös and A. Rényi, "On random graphs, I," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[21] A. Fifield, "Chinese app on Xis ideology allows data access to users' phones, report says," *The Washington Post*, Oct. 2019.

[22] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is TextSecure?" in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, March 2016, pp. 457–472.

[23] A. Greenberg, "Signal is finally bringing its secure messaging to the masses," Feb 2020. [Online]. Available: https://www.wired.com/story/signal-encrypted-messaging-features-mainstream/

[24] J. Häkkilä and C. Chatfield, "'It's like if you opened someone else's letter': User perceived privacy and social practices with SMS communication," in *Proceedings of the 7th International Conference on Human Computer Interaction with Mobile Devices & Services*, ser. MobileHCI '05. ACM, 2005, pp. 219–222.

[25] S. Havron, D. Freed, R. Chatterjee, D. McCoy, N. Dell, and T. Ristenpart, "Clinical computer security for victims of intimate partner violence," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 105–122. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/havron

[26] F. Hillebrand, F. Trosby, K. Holley, and I. Harris, *Short Message Service (SMS): The Creation of Personal Global Text Messaging*. Wiley, 2010.

[27] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, pp. 13–30, 1963. [Online]. Available: http://www.jstor.org/stable/2282952

[28] G. W. Index, "VPN users around the world," 2018. [Online]. Available: https://www.globalwebindex.com/reports/vpn-usage-around-the-world

[29] Johns Hopkins Foreign Affairs Symposium, "The price of privacy: Re-evaluating the NSA," https://youtu.be/kV2HDM86XgI?t=1079, April 2014.

[30] D. Kesdogan, D. Agrawal, Vinh Pham, and D. Rautenbach, "Fundamental limits on the anonymity provided by the MIX technique," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006, pp. 14 pp.–99.

[31] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally scaling strong anonymity," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 406–422.

[32] B. LaBelle, "Secure messaging apps are growing faster in corrupt countries," *Apptopia blog*, 2018. [Online]. Available: https://blog.apptopia.com/secure-msging-growth-corrupt

[33] A. Langley, "Pond," 2015.

[34] H. Leibowitz, A. M. Piotrowska, G. Danezis, and A. Herzberg, "No right to remain silent: isolating malicious mixes," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1841–1858.

[35] B. Libert and J.-J. Quisquater, "Efficient signcryption with key privacy from gap Diffie-Hellman groups," 2004, pp. 187–200.

[36] J. Lund, "Technology preview: Sealed sender for signal," Oct 2018. [Online]. Available: https://signal.org/blog/sealed-sender/

[37] N. Mallesh and M. Wright, "The reverse statistical disclosure attack," in *Information Hiding*. Springer Berlin Heidelberg, 2010, pp. 221–234.

[38] B. Marczak, J. Scott-Railton, S. McKune, B. A. Razzak, and R. Deibert, "Hide and Seek: Tracking NSO Groups Pegasus Spyware to Operations in 45 Countries," University of Toronto, Tech. Rep. 113, Sep. 2018.

[39] M. Marlinspike, "Advanced cryptographic ratcheting," *Signal Blog*, Nov. 2013.

[40] N. Mathewson and R. Dingledine, "Practical traffic analysis: Extending and resisting statistical disclosure," in *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2005, pp. 17–34.

[41] S. E. McGregor, P. Charters, T. Holliday, and F. Roesner, "Investigating the computer security practices and needs of journalists," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 399–414. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/mcgregor

[42] Open Whisper Systems, "Signal source code," https://github.com/signalapp, 2013.

[43] S. Oya, C. Troncoso, and F. Pérez-González, "Meet the family of statistical disclosure attacks," in *IEEE Global Conference on Signal and Information Processing, GlobalSIP*. IEEE, 2013, pp. 233–236. [Online]. Available: https://arxiv.org/abs/1910.07603

[44] F. Pérez-González and C. Troncoso, "Understanding statistical disclosure: A least squares approach," in *Privacy Enhancing Technologies*. Springer Berlin Heidelberg, 2012, pp. 38–57.

[45] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 482–494, May 1998.

[46] P. Rösler, C. Mainka, and J. Schwenk, "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, April 2018, pp. 415–429.

[47] M. Schliep and N. Hopper, "End-to-end secure mobile group messaging with conversation integrity and deniability," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, ser. WPES'19. ACM, 2019, pp. 55–73.

[48] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 423–440.

[49] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure messaging," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 232–249.

[50] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. ACM, 2015, pp. 137–152.

[51] Y. Wang, M. Manulis, M. H. Au, and W. Susilo, "Relations among privacy notions for signcryption and key invisible "Sign-then-Encrypt"," 2013, pp. 187–202.

[52] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, p. 440, 1998.

APPENDIX

A. Proof of Theorem 1

Consider first an arbitrary non-associate Charlie, with probability r_c of appearing in a random or target epoch. We first analyze the probability that Alice appears above Charlie in the ranking after n random and target epochs.

Recall that the attack maintains a "score" for each user, increasing by 1 each time the user appears in a target epoch, and decreasing by 1 each time the user appears in a random epoch. Define $2n$ random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$, corresponding to the signed difference in Alice and Charlie's scores during each of the n random epochs (X_i's) and target epochs (Y_i's). So each $X_i, Y_i \in \{-1, 0, 1\}$ and the sum $\bar{X} = \sum_{1 \le i \le n}(X_i + Y_i)$ is the difference in Alice and Charlie's score at the end of the attack. We wish to know the probability that $\bar{X} > 0$.

By the stated probability assumptions, we know the expected value of all of these random variables: $\mathbb{E}[X_i] = r_c - r_a$, $\mathbb{E}[Y_i] = t_a - r_c$, and therefore by linearity of expectation, $\mathbb{E}[\bar{X}] = n(t_a - r_a)$. Crucially, note that this is *independent of Charlie's probability* r_c because we included the same number of random and target epochs.

We can now apply Hoeffding's inequality [27] over the sum of these $2n$ independent, bounded random variables X_i, Y_i to conclude that $\Pr[\bar{X} \le 0] \le \exp(-n(t_a - r_a)^2/4)$.

Noting that this bound does not depend on the particular non-associate Charlie in any way, we can apply a simple union bound over all $\le m$ non-associates to obtain the stated result.

B. Proof Of Security For One-Way Sealed Sender Conversations

We now give a proof that the protocol in Section VI-B realizes the ideal functionality in Figure 6. As mentioned, we give this proof in the standalone model with static corruptions.

We define the simulator Sim as follows:

Setup: At startup, Sim generates long-term key pairs $(\mathsf{pk}_{P_i}, \mathsf{sk}_{P_i})$ for each honest user $P_i \in P_H$. Next, Sim receives a public key pk_{P_j} for each corrupt user $P_j \in P_C$ from the adversary.

Sim initializes an empty table $\mathbb{T}$ with format

$$(\mathtt{cid}, P_s, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

where P_s is the identity of the conversation initiator, $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}})$ is the keypair used by P_s in conversation $\mathtt{cid}$, P_r is the initial receiver, and $(\mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$ is the keypair used by P_r in conversation $\mathtt{cid}$. Some elements in these entries may be empty if Sim does not know the value. We will represent unknown elements with $\cdot$.

Sim also initializes an empty message table $\mathbb{M}$ with format

$$(\mathtt{cid}, \mathtt{mid}, c)$$

Note that the definition presented in Figure 6 is in terms for *pull notifications*, while the protocol in Section VI-B is in terms of *push notifications*. However, the push notification in the protocol, modeled after how Signal actually works, are essentially a sustained pull. That is, opening a longterm connection is equivalent to having the receiver continuously sending pull requests to the server. To bridge this gap, the simulator maintains a list of open connections. At each time step, the simulator iterates through the list of open connections are sends a ReceiveMessage to the ideal functionality of that players part. Similarly, we expect that honest users will do this if they want push-style notifications.

1) **Honest user starts a conversation with an honest user.** When Sim receives the message $(\mathsf{ApproveNewConvo}, P_r, \mathtt{cid})$ from the ideal functionality, samples $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$. Sim retrieves the longterm information for user P_r, *i.e.* $\mathsf{pk}_{P_r}, \mathsf{sk}_{P_r}$. Add the entry

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{P_r}, \mathsf{sk}_{P_r})$$

to $\mathbb{T}$ and then does the following:

a) Encrypt $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\text{``init''} \| \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, \mathsf{pk}_{P_r})$
b) Send c to P_{service}

If Sim gets c' from P_{service} for P_r and $c = c'$, Sim performs the following

a) sends an acknowledgment to P_{service} on behalf of P_r for P_s
b) receives the acknowledgment on behalf of P_s
c) sends $(\mathsf{Approve})$ to ideal functionality

Otherwise, Sim sends $(\mathsf{Disapprove})$ to the ideal functionality

2) **Honest user starts a conversation with an corrupt user.** When Sim receives the message $(\mathsf{ApproveNewConvoCorrupt}, P_s, P_r, \mathtt{cid})$ from the ideal functionality, samples $(\mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$. Sim retrieves the longterm information for user P_r, *i.e.* pk_{P_r}. Add the entry

$$(\mathtt{cid}, P_s, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{P_r}, \cdot)$$

to $\mathbb{T}$ and then does the following:

a) Encrypt $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\text{``init''} \| \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{P_s}, \mathsf{pk}_{P_r})$
b) Send c to P_{service}

If Sim gets an acknowledgment from P_{service} for P_s, Sim sends $(\mathsf{Approve})$ to ideal functionality. Otherwise, Sim sends $(\mathsf{Disapprove})$ to the ideal functionality.

3) **Corrupt user starts a conversation with an honest user.** When Sim receives a message $c \| \mathsf{pk}_e$ from P_{service} for an honest player P_h, Sim retrieves the longterm information for that player, *i.e.* $\mathsf{pk}_{P_s}, \mathsf{sk}_{P_s}$. Sim then does the following:

a) Decrypt and verify $(\text{``init''} \| \mathsf{pk}_e, x, \mathsf{pk}_c) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{P_h}, c)$ On failure, Sim halts.
b) Find a player P_c with longterm public key $\mathsf{pk}_{P_c} = \mathsf{pk}_c$. If no such player exists, Sim halts.
c) Send $(\mathsf{StartConvo}, P_h)$ to the ideal functionality on behalf of P_c and receive $(\mathsf{ApproveNewConvoCorrupt}, P_c, P_h, \mathtt{cid})$ in return. Sim responds with $(\mathsf{Approve})$. Sim drops the resulting notification.
d) Generate an acknowledgment message using pk_e and sk_h and send it to P_{service} on behalf of P_h for the identity pk_e

Finally, Sim adds the entry

$$(\mathtt{cid}, P_h, \mathsf{pk}_{P_h}, \mathsf{sk}_{P_h}, P_c, \mathsf{pk}_e, \cdot)$$

to $\mathbb{T}$

4) **Anonymous honest user sends a message to another honest user.** When Sim receives the message $(\mathsf{NotifyAnonymousSendMessage}, \mathtt{cid}, \mathtt{mid}, |m|)$ from the ideal functionality, Sim looks up the entry

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$ and performs the following:

a) Samples $m_0 \leftarrow_\$ \{0,1\}^{|m|}$
b) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m_0, \mathsf{sk}_{r,\mathtt{cid}}, \mathsf{pk}_{s,\mathtt{cid}})$
c) Sends c to the P_{service} for $\mathsf{pk}_{s,\mathtt{cid}}$ from $\mathsf{pk}_{r,\mathtt{cid}}$
d) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

5) **Non-anonymous honest user sends a message to another honest user.** When Sim receives the message $(\mathsf{NotifySendMessage}, \mathtt{cid}, \mathtt{mid}, P_r, |m|)$ from the ideal functionality, Sim looks up the entry

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$ and performs the following:

a) Samples $m_0 \leftarrow_\$ \{0,1\}^{|m|}$
b) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m_0, \mathsf{sk}_{s,\mathtt{cid}}, \mathsf{pk}_{r,\mathtt{cid}})$
c) Sends c to the P_{service} for $\mathsf{pk}_{r,\mathtt{cid}}$ from $\mathsf{pk}_{s,\mathtt{cid}}$
d) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

6) **Honest user sends a message to a corrupt user.** When Sim receives the message $(\mathsf{NotifySendMessageCorrupt}, \mathtt{cid}, \mathtt{mid}, m, P_h, P_c)$ from the ideal functionality, Sim looks up the entry

$$(\mathtt{cid}, P_h, \mathsf{pk}_{h,\mathtt{cid}}, \mathsf{sk}_{h,\mathtt{cid}}, P_c, \mathsf{pk}_{c,\mathtt{cid}}, \cdot)$$

in $\mathbb{T}$ and performs the following:

a) Computes $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(m, \mathsf{sk}_{h,\mathtt{cid}}, \mathsf{pk}_{c,\mathtt{cid}})$
b) Sends c to the P_{service} for $\mathsf{pk}_{c,\mathtt{cid}}$ from $\mathsf{pk}_{h,\mathtt{cid}}$
c) Records the entry $(\mathtt{cid}, \mathtt{mid}, c)$ in $\mathbb{M}$

7) **Anonymous honest user receives a message from an honest user.** When Sim receives a set of messages

$$\{(\mathsf{ApproveAnonymousReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m_i|)\}_{i \in [k]}$$

from the ideal functionality, Sim looks up

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$. Additionally, for each message, Sim looks for an entry $(\mathtt{cid}, \mathtt{mid}_i, c_i)$ in $\mathbb{M}$. The ideal functionality authenticates to P_{service} with the identity $\mathsf{pk}_{s,\mathtt{cid}}$ and receives messages $\{a'_j \| c'_j\}_{j \in [k']}$ in return. Sim does the following:

a) For each message $(\mathsf{ApproveAnonymousReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m_i|)$ and associated entry $(\mathtt{cid}, \mathtt{mid}_i, a_i, c_i)$, if P_{service} sent a message $a'_j \| c'_j$ such that $a'_j = a_i$ and $c'_j = c_i$, sends $(\mathsf{Approve}, \mathtt{mid})$. If no such $a'_j \| c'_j$ exists, Sim sends $(\mathsf{Approve}, \mathtt{mid})$.
b) For each message $a'_j \| c'_j$, if there does not exist and entry $(\mathtt{cid}, \mathtt{mid}, a'_j, c'_j)$ for some value of $\mathtt{mid}$, Sim decrypts $(m_j, \mathsf{pk}_j) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{s,\mathtt{cid}}, c_i)$. If $\mathsf{pk}_j = \mathsf{pk}_{r,\mathtt{cid}}$, the simulator aborts with an error.

8) **Non-anonymous Honest user receives a message from an honest user.** When Sim receives the set of messages

$$\{(\mathsf{ApproveReceiveMessage}, \mathtt{cid}, \mathtt{mid}_i, |m|, P_r)\}_{i \in [k]}$$

from the ideal functionality, Sim looks up

$$(\mathtt{cid}, \cdot, \mathsf{pk}_{s,\mathtt{cid}}, \mathsf{sk}_{s,\mathtt{cid}}, P_r, \mathsf{pk}_{r,\mathtt{cid}}, \mathsf{sk}_{r,\mathtt{cid}})$$

in $\mathbb{T}$. Additionally, for each message, Sim looks for an entry $(\texttt{cid}, \texttt{mid}_i, c_i)$ in $\mathbb{M}$. The ideal functionality authenticates to P_{service} with the identity $\mathsf{pk}_{r,\texttt{cid}}$ and receives messages $\{c'_j\}_{j\in[k']}$ in return. Sim does the following:

a) For each message $(\mathsf{ApproveReceiveMessage}, \texttt{cid}, \texttt{mid}_i, |m_i|, P_r)$ and associated entry $(\texttt{cid}, \texttt{mid}_i, c_i)$, if P_{service} sent a message c'_j such that $c'_j = c_i$, sends $(\mathsf{Approve}, \texttt{mid})$. If no such c'_j exists, Sim sends $(\mathsf{Approve}, \texttt{mid})$.

b) For each message c'_j, if there does not exist and entry $(\texttt{cid}, \texttt{mid}, c'_j)$ for some value of mid, Sim decrypts $(m_j, \mathsf{pk}_j) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{r,\texttt{cid}}, c_i)$. If $\mathsf{pk}_j = \mathsf{pk}_{s,\texttt{cid}}$, the simulator aborts with an error.

9) **Honest user receives a message from a corrupt user.** When Sim receives the message $(\mathsf{ApproveReceiveMessageCorrupt}, \texttt{cid}, P_s, P_r)$ from the ideal functionality, it looks up

$$(\texttt{cid}, P_h, \mathsf{pk}_{h,\texttt{cid}}, \mathsf{sk}_{h,\texttt{cid}}, P_c, \mathsf{pk}_{c,\texttt{cid}}, \cdot)$$

in $\mathbb{T}$. Sim authenticates to P_{service} with $\mathsf{pk}_{h,\texttt{cid}}$ and gets a set of messages $\{c_i\}_{i\in[k]}$ from P_{service}. For each c_i Sim does the following:

a) decrypts $(m_i, \mathsf{pk}_i) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_{h,\texttt{cid}}, c_i)$. If it fails, the message is dropped.

b) send the tuple $(\texttt{cid}, P_c, P_h, m_i)$ to the ideal functionality

Although the simulator is quite involved, the security argument is quite straight forward hybrid argument, starting with the real experiment $\mathcal{H}_0$. In $\mathcal{H}_1$, conversation opening messages between honest parties take the ephemeral secret key instead of the sender's longterm secret key. Due to the ciphertext anonymity of Π_{ssenc}, the distance between $\mathcal{H}_0$ and $\mathcal{H}_1$ is negligible. In $\mathcal{H}_2$, the plaintext contents of messages between honest users are replaced with random messages of the same length. Due to the security of Π_{ssenc}, the distance between $\mathcal{H}_1$ and $\mathcal{H}_2$ is negligible. In $\mathcal{H}_3$, if the service provider delivers a message on behalf of an anonymous honest user that the honest user did not send, the experiment aborts. Due to the authenticity property of Π_{ssenc}, the distance between $\mathcal{H}_2$ and $\mathcal{H}_3$ is negligible. In $\mathcal{H}_4$, if the service provider delivers a message on behalf of a non-anonymous honest user that the honest user did not send, the experiment aborts. Due to the authenticity property of Π_{ssenc}, the distance between $\mathcal{H}_3$ and $\mathcal{H}_4$ is negligible. Finally, in $\mathcal{H}_5$ keys are generated randomly by the simulator instead of the honest parties. Because the keys are sampled at random, the distributions of $\mathcal{H}_4$ and $\mathcal{H}_5$ are the same. $\mathcal{H}_5$ and the simulator above are distributed identically, so the proof is done.

C. Protocols for Two-Way Sealed Sender Conversations

This appendix provides more details for the two-way sealed sender solution discussed in Section VI-C.

Recall how this solution works: after an initiator sends a sealed sender message to the long-term identity of the receiver communicating the sender's ephemeral identity, the receiver generates a fresh, ephemeral identity of their own and sends it to the sender's ephemeral identity via sealed sender. After this initial exchange, the two users communicate using only their ephemeral identities and sealed sender messages, in both directions (two-way).

Notation	Type	Meaning	Anonymous
P_s	User	Sender/Initiator	-
P_r	User	Receiver	-
$(\mathsf{pk}_s, \mathsf{sk}_s)$	Π_{ssenc} Keys	Sender/Initiator key	N
$(\mathsf{pk}_r, \mathsf{sk}_r)$	Π_{ssenc} Keys	Receiver key	N
$(\mathsf{pk}_e, \mathsf{sk}_e)$	Π_{ssenc} Keys	Ephemeral key	Y
$(\mathsf{pk}_{es}, \mathsf{sk}_{es})$	Π_{ssenc} Keys	Sender/Initiator eph. key	Y
$(\mathsf{pk}_{er}, \mathsf{sk}_{er})$	Π_{ssenc} Keys	Receiver eph. key	Y

Fig. 7: Notation for two-way sealed sender protocols

The protocol proceeds as follows: When some conversation initiator P_s wants to start a conversation with a user P_r, the initiator executes **Initiate Two-Way Sealed Conversation** (see below). P_s starts by generating a keypair $(\mathsf{pk}_{es}, \mathsf{sk}_{es})$ and registering pk_{es} with the service provider. P_s then runs the **Change Mailbox** protocol (see below), which informs the receiver of pk_{es} by sending a message to pk_r. The receiver P_r then generates a keypair $(\mathsf{pk}_{er}, \mathsf{sk}_{er})$ and registers pk_{er} with the service provider. Finally, the P_r runs the **Change Mailbox** protocol, informing P_s about pk_{er} by sending a message to pk_{es}. P_s and P_r communicate using **Send message**, **Open Connection**, and **Push Message** as in Section VI-B (for brevity, these protocols have not been replicated below).

Initiate Two-Way Sealed Conversation to P_r:

1) P_s looks up the P_r's long-term key pk_r
2) P_s generates keys $(\mathsf{pk}_{es}, \mathsf{sk}_{es}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$ and opens a mailbox with public key pk_{es}
3) P_s runs the subroutine **Change Mailbox**$(P_r, \mathsf{pk}_{es}, \mathsf{sk}_s, \mathsf{pk}_r)$
4) P_r generates keys $(\mathsf{pk}_{er}, \mathsf{sk}_{er}) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSKeyGen}(1^\lambda)$ and opens a mailbox with public key pk_{er}
5) P_r runs the subroutine **Change Mailbox**$(P_s, \mathsf{pk}_{er}, \mathsf{sk}_r, \mathsf{pk}_{es})$
6) P_s records $(P_r, \mathsf{pk}_{er}, \mathsf{pk}_{es}, \mathsf{sk}_{es})$ and P_r records $(P_s, \mathsf{pk}_{es}, \mathsf{pk}_{er}, \mathsf{sk}_{er})$ in their respective conversation tables
7) Both P_s and P_r use **send message** to send a read-receipt acknowledgment to pk_{er} and pk_{es} respectively.

Change Mailbox$(P_r, \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$:

1) User changing mailbox P_s does the following (note that this user may be the conversation initiator or the conversation receiver)
   a) encrypts $c \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSEnc}(\texttt{``init''} \| \mathsf{pk}_e, \mathsf{sk}_s, \mathsf{pk}_r)$
   b) connects to the server provider anonymously and sends $c \| \mathsf{pk}_e$ to the service provider addressed to pk_r.
2) The service provider opens a mailbox with public key pk_e and delivers c to pk_r (sealed sender)
3) When the other user P_r calls receive message, it decrypts and verifies $(\texttt{``init''} \| \mathsf{pk}_e, \mathsf{pk}_s) \leftarrow \Pi_{\text{ssenc}}.\mathsf{SSDecVer}(\mathsf{sk}_r, c)$.