Summary

The correctness of privacy protocols can be asserted in many different ways. One way is to set up a differential privacy model for it, which means that it is private, but over time it will lose privacy. Another way is to prove that it is private. This can be done by creating some assumptions about the usage of the protocol, and then proving that under those assumptions the protocol will provide privacy. Another way is simply to test it with different disclosure attacks and see if the protocol can withstand it. This approach however requires that the protocol is implemented, and implemented correctly, such that an analysis of it can be conducted, for example through a simulation. A disclosure attack can then be used on a network trace, from the simulation to try and disclose the users. If the attack can disclose the users correctly, then the protocol is not private, and if the attack cannot disclose the users, then the protocol is resistant towards that attack. But a protocol being resistant towards one attack is not the same as it being private, it only means that it can resist that attack. The good thing about testing is that it is easy, either it resist or it does not and by showing that it is not resistant to one attack then it is also proved that the protocol is not private. Therefore, the purpose of this study is to create a tool which can be used to test the privacy of different privacy protocols and use it to evaluate DenIM a deniable protocol.

The tool that we have created consists of different disclosure attacks. These attacks can be used to test an implementation of a protocol to see if the protocol is private. If the protocol has been proven private, like DenIM, then these attacks can test whether the protocol has been implemented correctly. Because if the attacks works against a protocol which has been formally proven to be private, then it must mean that it has been implemented incorrectly.

One of the attacks which has been implemented is the Normalized Statistical Disclosure Attack. This attack is designed to work on threshold mixes, and not continuous networks. Therefore, we have modified it slightly to work on continuous networks. This is accomplished by looking at time intervals of the network data of fixed time t, and utilizing these intervals for the attack instead of the thresholds which would otherwise be used. This is possible because a threshold in a mix network is some amount of messages passing through the server at a specific time, whereas the intervals are some messages which has passed through the server within some time. Therefore both thresholds and intervals represent some messages passing through the server at a time, where the only difference is that intervals uses a duration instead of a fixed time. We have used this attack to disclose users communicating through an application utilizing the Signal protocol. This would serve as a test to see whether our attacks worked, since the signal protocol is not private. From that, we can see that our attack works and can disclose Signal users.

With this attack in place, we can use it to test the implementation of DenIM which has been formally proved to be private. By using our attack as is, we can see that the implementation of DenIM fends of our attacks. DenIM is a privacy protocol which acts as an extension for the Signal protocol. It works by letting the users have some conversations which should be private, these are refereed to as deniable, and some conversations which are not. The messages of a deniable conversation is called deniable messages, and the messages of a regular conversation is called regular messages. In order for a user to send a deniable message it must be picky-bagged on a regular message. This is done by padding every regular message with a buffer and optionally store some of the deniable message in it. By doing this, the deniable messages will be hidden from the transport layer, making regular messages carrying deniable messages indistinguishable from regular messages which are not carrying deniable messages. The size of the buffer is determined by a value q which is used to describe a relation between the deniable and regular payload. When a regular message hits the server, it will drop of the part of the deniable message if it was carrying something in its buffer. After that it will check to see if the recipient of this deniable message has some deniable messages in its queue on the server, and if so piggybacks that and then it is forwarded to the recipient. So in order for a deniable message to go from the server to the destination, it must be piggybacked on some other regular message going to that destination.

DenIM operations under a specific user assumption which states that a users deniable behavior must not influence their regular behavior. This means that when a user has created a deniable message, he must not send more regular traffic to forward it than he otherwise would, because this can leak his privacy. We have addressed this assumption and tried to come op with a less restrictive assumption which states that a user in a hurry can send a larger volume of regular traffic than they otherwise would to forward the message. Then we have tested this variation of DenIM following this new user behavior assumption, to see how much privacy is actually lost. To accomplish this, we have created a new attack for our tool which can detect these increases of regular traffic and use that as an indicator of a deniable message having been forwarded. Then we can try to correlate these events to find possible deniable communication links between the users. From the attack which we have conducted on DenIM where users follow this new user behavior assumption we can see that the privacy lost is related to q. When q is large, the users are difficult to correlate, and when q is small, the users are easier to correlate. So a large q would seem like an obvious choice, but unfortunately that would lead to a high bandwidth.

Traffic Analysis Tool and The Deniable Disclosure Attack

Department of Computer Science
CS-25-DS-10-08 , Spring 2025

Semester Project

STUDENT REPORT
AALBORG UNIVERSITY

Title:
Traffic Analysis Tool and The Deniable Disclosure Attack

Theme:
Distributed Systems

Project Period:
Spring 2025

Project Group:
cs-25-ds-10-08

Participants:
Mads Møller Kristensen
Mikkel Boje Larsen

Supervisors:
Danny Bøgsted Poulsen
René Rydhof Hansen

Github Organization:
`https://github.com/orgs/cs-25-ds-10-08/`

Page Count:
26

Date of Completion:
2025-06-6

The Technical Faculty of IT and Design
Aalborg University
https://www.aau.dk

Abstract:

Messages sent through Signal and many other instant messaging applications are encrypted such that an adversary cannot read the content of the messages. But even if that is the case, an adversary can figure out who is communicating with who and when that communication took place. With this knowledge, the adversary will be able get and idea about the context of the messages. Therefore the metadata privacy of a system is just as important. In this project we have created a tool which consist of different disclosure attacks. This tool can then be used to test the privacy of different privacy protocols. Throughout this project, we have utilized this tool to test the privacy of Signal and DenIM. In Chapter 3 the focus is to test the privacy of Signal by using a variation of the Normalized Statistical Disclosure Attack which is developed in that chapter. In Chapter 4 we shift the focus to test the privacy of DenIM with the same attack as used on Signal. DenIM has a specific assumption about user behavior which the privacy is dependent on. In Chapter 5 we try to address this assumption, and come up with a less restrictive new assumption. In order to test the privacy of DenIM where users follow this new user behavior assumption, we developed a new attack in Chapter 6 which tries to detect this kind of behavior to correlate the clients. Finally, we discuss in Chapter 7 the results from using the tool on simulated data, and additionally discuss the new attack which we created to disclose DenIM where users would follow the new user behavior assumption.

AALBORG UNIVERSITY
STUDENT REPORT

The content of this report is freely available, but publication (with reference) may only be pursued after agreement with the author.

Acronyms

NSDA	Normalized Statistical Disclosure Attack	1–4, 7, 11, 15 f., 23
SDA	Statistical Disclosure Attack	1 f., 23
SNI	Server Name Identification	5

Contents

1 Introduction 1

2 Traffic Analysis Tool 2

3 Attacking Signal 3
3.1 The *Normalized Statistical Disclosure Attack* . . . 3
3.2 Attacking Signal in Practice . . . 5
3.3 Experiments . . . 5

4 DenIM-on-Signal 11
4.1 What is DenIM . . . 11
4.2 Initializing the Simulations . . . 12
4.3 Attacking DenIM . . . 12

5 DenIM's User Behavior Assumption 14

6 Analysis 15
6.1 Attacking DenIM Under the New User Behavior . . . 15
6.2 Results . . . 18

7 Discussion 21

8 Future Work 22

9 Related Work 23

10 Conclusion 24

Reference 25

1. Introduction

Many instant messaging applications like Signal and Facebook Messenger provide strong security guarantees through encryption. For Signal and Facebook Messenger [1] the security is provided through the Signal protocol which is formally secure [2] and provides resilience, forward security, and break-in recovery [3]. This means that an adversary cannot read the content of the messages. But an adversary will still be able to correlate communicating users with well known disclosure attacks [4, 5]. This can be just as compromising because an adversary can reason about and try to infer the context of the messages by knowing who has communicated and when. Making conversations private would then allow people to communicate without possible consequences as could be under e.g. authoritarian rule. Furthermore, former US government official General Hayden said, "We kill people based on metadata" [6], which further underlines why privacy is necessary.

For an instant messaging application to be private, it must prevent meta data leakage that could allow an adversary to correlate the communicating parties. A system usually follows the anonymity tri-lemma [7], meaning that the system can provide low latency, low bandwidth, and privacy, but only two of the properties at a time. This means that an instant messaging application, which provides privacy for its users, must sacrifice either latency or bandwidth. There has already been proposed several privacy protocols. Vuvuzela [8], Karaoke [9], and Stadium [10] among others has taken a round based approach and utilizes mix networks. Generally the approaches send messages through their respective mix network architectures, adding a large amount of noise in the process making disclosure slow. These solutions generally use a differential privacy model making metadata leakage inevitable but bound, instead of being proven private. DenIM [11] is another privacy protocol which has recently emerged with proven privacy guarantees as a novel approach to the privacy problem. DenIM works by hiding so called deniable messages which, supported with various privacy guarantees, allows for private communication. The privacy is gained by their novel architecture and by hiding deniable messages within regular messages at the cost of an overhead on the regular messages. This overhead can contain a deniable message which means that one message sent by a client could in fact include two messages, however the adversary cannot make the distinction of the overhead either being a dummy message or a deniable message hence providing deniable properties. A further explanation of the DenIM protocol can be seen in Chapter 4. We choose to focus on DenIM since it takes a new approach towards privacy, they state that "*rather than offering privacy to all users all the time, let us offer privacy to all users some of the time.*" [11], which we find interesting. We also believe that this approach has better potential for use in current instant messaging services and therefore should be the focus of improvement. To see whether a protocol is private, one could either prove it mathematically like in DenIM, or one could test it with different traffic analysis attacks.

The purpose of traffic analysis attacks is to disclose communication links between clients. Many of these attacks rely on statistics such as the *Statistical Disclosure Attack* (SDA) proposed by [4], and variations thereof like *Normalized Statistical Disclosure Attack* (NSDA) proposed by [5]. Many of these traffic analysis attacks are, to the best of our knowledge, not implemented and publicly available or grouped together as a collection, making traffic analysis an inconvenient way to test privacy solutions. Therefore we propose to implement a selection of traffic analysis attacks and collect them into a tool for the purpose of testing privacy systems, an explanation of this tool can be seen in Chapter 2. We also propose two variations of the NSDA which work on continuous networks as opposed to mix networks, see Chapter 3. Furthermore, we also propose a new attack which can disclose a variation of DenIM where users do not follow the normal user behavior assumption as stated in [11], which can be seen in Chapter 6.

2. Traffic Analysis Tool

To further demonstrate the need for a privacy solution, we show that Signal is not private by attacking it with a well know traffic analysis attack. We also want to show how DenIM can solve this issue and attack it as well. For other studies which might also need to verify that their implementation is resistant to the attacks, we have chosen to group and automate the attacks within a tool. This tool is publicly available on our GitHub[1]. The tool consists of four different disclosure attacks, two of which is used in this study to disclose and test the privacy of Signal and DenIM.

Many disclosure attacks are based upon the SDA proposed by [4], as this attack has shown great results in the past, hence many have further developed upon it. One of such is the NSDA proposed in [5], which uses the original SDA and builds upon it. They show that it is a general improvement to the SDA. Their model, unlike the original, does not focus on a single user's communication pattern, but instead uses every users' communication patterns and extract more information thereby improving the algorithm, detailed in Section 3.1. Variations of these two attacks are implemented in the tool and work on continuous networks as opposed to their original purpose for mix networks. Our implementation is presented in Section 3.1.

Another disclosure attack within the tool is a variation of the SDA proposed by [12]. This attack is important since it coverts the SDA from working on a round based protocol to a continuous network stream. The approach we got from [12] was also used to implement the SDA and NSDA in the tool and therefore they work on continuous network streams.

According to our tests NSDA yields the best results compared to the two other disclosure attacks. It is therefore also the reason we used it to disclose conversing Signal users in Chapter 3 and to test the privacy of DenIM in Chapter 4.

Finally we have also implemented an attack for the purpose of testing and disclosing DenIM users following a new user behavior model described in Chapter 5. A detailed explanation of this attack can be seen in Section 6.1.

[1]`https://github.com/cs-25-ds-10-08/Traffic-Analysis-Tools`

3. Attacking Signal

To further demonstrate the need for a privacy solution, we show that plainly using the Signal protocol leaves services to be vulnerable to traffic analysis attacks. Using a fork[1] implementing a client-server architecture using the Signal protocol[2], we show that this implementation is not private. To illustrate Signal's lack of privacy, we employ a NSDA, inspired by [5]. This attack correlates users through their traffic patterns over Signal - disclosing users, meaning to reveal likely communication links between users.

3.1 The Normalized Statistical Disclosure Attack

The goal of this attack is to identify which two users have communicated with each other. The *Normalized Statistical Disclosure Attack*, as proposed in [5], is comprised of two steps to reach said goal, a profiling and disclosure step. The goal of the profiling step is to generate a profile for each user, a profile has a user's predicted communication pattern containing the probabilities of it sending a message towards each of the other users. Profiling therefore generates a pattern for users which then is used for correlation and disclosure of those users. When generating profiles for each user, we focus on a specific time frame, in [5] this is a round of a threshold mix. During this period, we consider each sender and update their profile based on potential interactions. For each sender, we assume they could have sent their message to any of the receivers within that time frame. Therefore, the profile update process involves adding a value of $\frac{1}{|S_i|}$ to the sender's profile for each receiver, where $|S_i|$ represents the number of senders in round i. This adjustment updates the belief of the sender having communicated with each of the different receivers. The joint profiles is a zero initialized matrix where the rows are senders and the columns are the receivers. The profiles is updated in the following manner, where P is the matrix of profiles:

1. For each round i, group the senders into the set S_i and the receivers into the set R_i.
2. Update the profiles of all senders by adding the likelihood of them having send a message to each receiver, which is depicted in Equation 3.1.

$$P'(s,r) = P(s,r) + \frac{1}{|S_i|}, s \in S_i, r \in R_i \tag{3.1}$$

Do note that [5] assumes a uniform distribution of the probability within rounds by adding $\frac{1}{|S_i|}$ to the profile of each sender in each round. This is because it is not certain which of the senders actually is communicating with the receiver.

With the profiles all set, the attack enters the disclosure step, where the goal is to use the profiles to make a guess on which users are communicating. The finalized profiles are then transformed into a doubly stochastic transition matrix, resulting in each row and column being normalized [5]. [5] uses the Sinkhorn algorithm [13] to convert the profiles into a doubly stochastic transition matrix, therefore we do the same. The intuition behind the transformation is a shift in perspective. Initially, the profiles hold all sent messages distributed across all senders. After the transformation each profile now describes, for

[1]The fork can be viewed here: `https://github.com/cs-25-ds-10-08/signal`
[2]The implementation can be viewed here: `https://github.com/Diesel-Jeans/signal`

a given user, how likely their messages are to go to each possible receiver. So rather than summarizing the sent messages the profiles now reflect the probability of a message's destination from the point of view of each sender. Using the transformed profiles, we can now identify communicating users based on their profile data. To conclude the attack, for any profile of interest, we select the receiver with the highest belief - our best guess as to whom the sender most likely communicated with.

The NSDA, originally proposed by [5], was designed for threshold mix networks. Therefore, we must adapt the attack slightly for it to be applicable for Signal and be a fit for the tool, as the focus is continuous networks and not threshold mix networks. To accommodate the issue we introduced an epoch-based approach, inspired by [12], which utilizes a new time frame that they call an `epoch`. The intuition behind using an epoch is that [12] models Signal as a threshold mix network and uses the epoch to act in place of the threshold. Therefore, the epoch is simply just a parameterized amount of time that can then be adjusted and as argued in [12] should allow for their round based attack to work for continuous networks instead. We propose two variations of the NSDA, `chunking` and `selected`, using epoch-based approaches. The goal of the different approaches is to create the sender and receiver sets to then perform the profiling.

The chunking variant utilizes the `epoch` to chunk the data into `epoch` sized chunks. It then partitions the chunk into a sender and receiver set, S_i and R_i respectively, and updates the senders' profiles as depicted in Equation 3.1. This can also be seen in algorithm 1 which depicts the entire chunking approach. The

Algorithm 1: The chunking NSDA variation

```
Input: data, epoch
Result: A matrix describing each clients profile
/* Each cell in the profiles matrix maps to 0                          */
1  $P(s, r) \mapsto 0$
2  $initial\_time \leftarrow data[0].time$
3  $chunk\_amount \leftarrow \lceil \frac{data.last.time - initial\_time}{epoch} \rceil$
4  for $i \leftarrow 0$ to $chunk\_amount$ do
5      $start \leftarrow initial\_time + i * epoch$
6      $end \leftarrow start\_time + epoch$
       /* The senders and receivers are not the server                     */
7      $S_i \leftarrow get\_senders\_between(data, start, end)$
8      $R_i \leftarrow get\_receivers\_between(data, start, end)$
9
10     foreach $s \in S_i$ do
11         foreach $r \in R_i$ do
12             if $s \neq r$ then
13                 $P(s, r)' \leftarrow P(s, r) + \frac{1}{|S_i|}$
14             end
15         end
16     end
17 end
18 return $P$
```

chunking method suffers from an issue where it can correlate a previous receiver with a later sender within a chunk. As an example if there is a receive event followed by a sending event, chunking would still correlate that the message originating from the sender could be received by that recipient, which is clearly not possible since the receive event happened earlier that the send event.

To accommodate the issue of chunking, we made the selected variation which does not chunk the data. This approach will read one line at a time from the network trace, and concludes whether the sender is a client or the server. If the sender is a client, it will simply add the client and the current timestamp to a buffer S. If the sender is the server, then the recipient must be a client, and all the senders in S could have sent a message to that recipient and therefore each sender's profile is updated accordingly, as depicted in Equation 3.1. Clients will be removed from S once they have been in S for epoch time. This can also be seen in algorithm 2 which depicts the entire selected approach.

Algorithm 2: The selected NSDA variation

```
Input: data, epoch, server
Result: A matrix describing each clients profile
/* Each cell in the profiles matrix maps to 0                                */
$P(s, r) \mapsto 0$
$S \leftarrow \emptyset$
foreach $row \in data$ do
    $S' \leftarrow remove\_expired\_events(S, row.time, epoch)$

    if $row.sender = server$ then
        foreach $s, \_ \in S$ do
            if $s \neq row.receiver$ then
                $P(s, row.receiver)' \leftarrow P(s, row.receiver) + \frac{1}{|S|}$
            end
        end
    else
        $S' \leftarrow S \cup \{(row.sender, row.time)\}$
    end
end
return $P$
```

3.2 Attacking Signal in Practice

We consider how the attack can perform in the real world on the production Signal server to show the relevance of a privacy solution. For the attack to work the attack needs to run on Signal messaging data, therefore it is important to deliberate on how this data is actually obtained. To obtain the data, the attacker has to be able to sniff the network traffic at the server which the two clients' messages pass through. A TLS handshake is initiated when a new connection to the Signal server is made. TLS handshakes made with the Signal server contain the *Server Name Identification* (SNI) of the Signal server which the user needs to connect with. The SNI can be used to filter information not coming from Signal - removing unnecessary noise. Doing this on the running production Signal traffic data also showed that using the SNI is an effective method for filtering out noise to get the necessary data needed for the attack. We tried this in practice by starting a hotspot, connection phones to the hotspot and communicating through the Signal App, then we sniffed the packages routed through the hotspot. This resulted in Signal data mingled with auxiliary data which after filtering as previously described, left us with only the desired Signal data. Further filter might be needed if the network trace contains Signal data from other services than the Signal messages, such as Signal's video chat data.

3.3 Experiments

To test the privacy of the Signal implementation mentioned in Chapter 3, we simulate client interaction. Before starting the simulation, we create all the clients and connect them to the server. Furthermore, we partition the clients into groups of 2 to 5 members, allowing only intra-group communication as a previous study [14] found that people typically communicate with 2 to 5 others, which informed our choice of group size. While this study provides insight into communication patterns, it focuses solely on employee communication between and within businesses and does not fully represent instant messaging services as a whole. To the best of our knowledge, no studies have yet clarified such general communication patterns.

The clients are run on a single machine to perform the simulation, following the approach taken in previous studies [5, 12]. To enable independent client behavior, we introduce parallelism using the Tokio async runtime[3], which allows tasks to run across multiple threads[4]. However, Tokio does not guarantee full parallelism, as its tasks are scheduled on a limited set of worker threads. While Tokio controls task scheduling, the operating system ultimately manages the execution of those threads, including context switching. This can introduce slight variations in execution timing and delays beyond our control. A potential solution for achieving full parallelism would be to run each client on a separate machine.

Each client starts by receiving all messages directed for it. It then probabilistic decides if it should reply to the message. This chance was set to 95% for all the results on the Signal client. If the client replied to a message, there is further a 10% chance it will send a new message to a random member in the group, and if the client did not reply it will always send a new message to a random member of its group. To ensure conversations automatically begin on startup, we also decided that each client should send out a message in the first round forcefully. The reason for running the simulation in iterations, and not within a fixed time, is to make the simulation more consistent since there always will be send and received close to the same amount of messages. Where as with a fixed time, the amount of messages may be more variable depending on the background load on the PC.

The traffic generated between the server and clients are captured on localhost with Wireshark[5], this also means that clients will be distinguishing based on their statically assigned port number. The network trace captured by Wireshark can unfortunately not be reliably reproduced because the output relies on network timing and delays, which we have no control over. In Wireshark we apply a filter on TCP ports that forces all packets to have either the source or destination port be the server port, in our case 4444. This isolates the client-server communication and we then run the attacks on this filtered captured data.

The tool is required to run on a .csv file which is created from the .pcapng file that Wireshark generates while collecting the traffic data.

3.3.1 Results

The result indicates that both the chunking and selected approach allows to disclose users with high belief over short conversations, even with significant amount of noise from other users. The results show the algorithm's belief in that the target user communicated with the expected user. Additionally, if the algorithm correlates correctly that the target user's expected receiver is the actual receiver, we mark the result as a `Hit`, otherwise a `Miss` indicating that the belief is of a different receiver than the actual receiver.

To have a high belief, the epoch of the attack must be set accordingly. This may require some tweaking and is highly dependent on the actual network capture, but as a goal the epoch should be long enough to contain both a sender event and the correlated receive event. Do note that the reason for the very low epoch used in the tests is due to the fact that the tests has been running on localhost with no delays. This epoch should be increased if a network trace was captured with transmission delays, such as if the server was running on another machine.

We can see on the data analyzed with the tool, that as the client amount increases and the iterations do not, our belief drops - as expected. This indicates that when there is many clients, but little data it is hard to be certain of the communication links, so in a sense, they are hiding within each others noise.

The analysis revealed that capturing more messages generally improves our belief. This is evident when approaching a local optimum, for example at $epoch = 5 * 10^{-4}$, where the belief increases consistently with the number of iterations. In contrast, at $epoch = 5 * 10^{-3}$, our belief declines as the number of iterations increases.

[3] `https://tokio.rs/`
[4] `https://docs.rs/tokio/latest/tokio/runtime/`
[5] `https://www.wireshark.org/`

Epoch	**Iterations**	**Clients**	**Belief**	**Hit/Miss**
$5 * 10^{-3}$s	10	100	2.50%	Hit
$5 * 10^{-4}$s	10	100	8.48%	Hit
$5 * 10^{-5}$s	10	100	0.00%	Miss
$5 * 10^{-3}$s	100	10	89.14%	Hit
$5 * 10^{-4}$s	100	10	93.61%	Hit
$5 * 10^{-5}$s	100	10	14.96%	Miss
$5 * 10^{-3}$s	100	100	82.49%	Hit
$5 * 10^{-4}$s	100	100	90.99%	Hit
$5 * 10^{-5}$s	100	100	0.45%	Miss
$5 * 10^{-3}$s	100	1000	84.58%	Hit
$5 * 10^{-4}$s	100	1000	86.99%	Hit
$5 * 10^{-5}$s	100	1000	0.06%	Miss
$5 * 10^{-3}$s	1000	100	88.69%	Hit
$5 * 10^{-4}$s	1000	100	92.50%	Hit
$5 * 10^{-5}$s	1000	100	63.38%	Hit

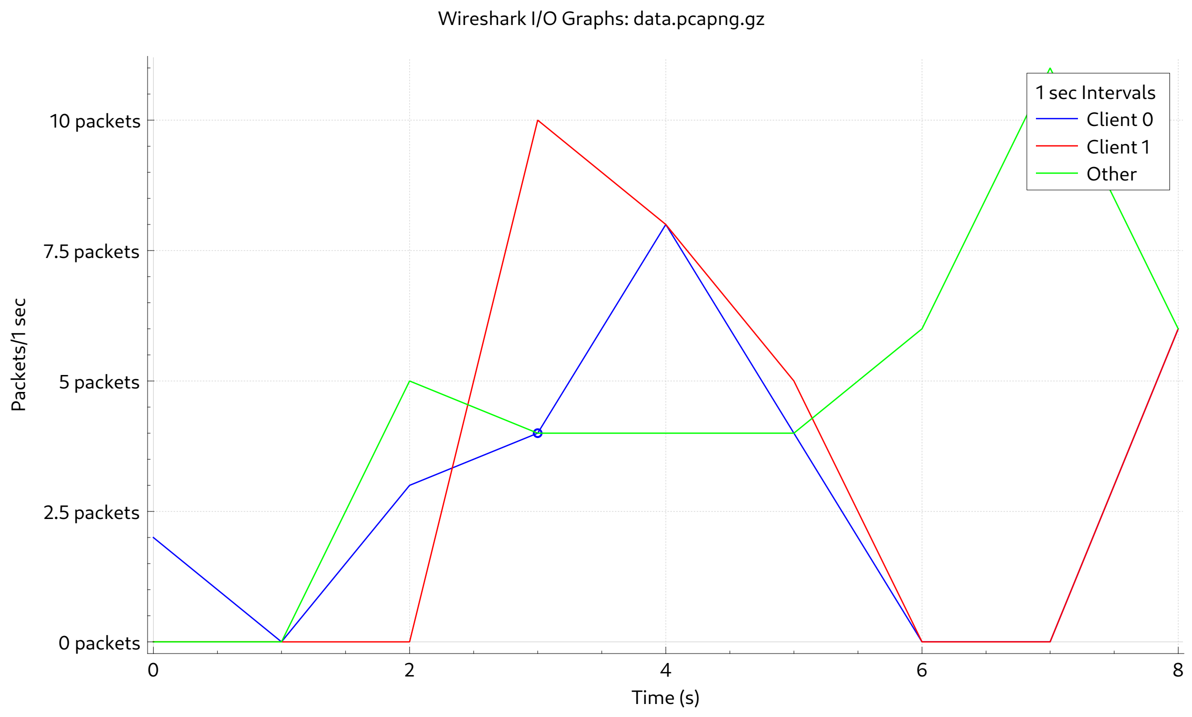
Table 3.1: Results for running chunking NSDA to correlate client 0 and client 1

Epoch	**Iterations**	**Clients**	**Belief**	**Hit/Miss**
$5 * 10^{-3}$s	10	100	2.91%	Hit
$5 * 10^{-4}$s	10	100	3.36%	Hit
$5 * 10^{-5}$s	10	100	0.00%	Miss
$5 * 10^{-3}$s	100	10	88.37%	Hit
$5 * 10^{-4}$s	100	10	95.51%	Hit
$5 * 10^{-5}$s	100	10	11.70%	Miss
$5 * 10^{-3}$s	100	100	79.57%	Hit
$5 * 10^{-4}$s	100	100	91.46%	Hit
$5 * 10^{-5}$s	100	100	1.09%	Miss
$5 * 10^{-3}$s	100	1000	34.25%	Hit
$5 * 10^{-4}$s	100	1000	86.88%	Hit
$5 * 10^{-5}$s	100	1000	0.07%	Miss
$5 * 10^{-3}$s	1000	100	36.67%	Hit
$5 * 10^{-4}$s	1000	100	95.71%	Hit
$5 * 10^{-5}$s	1000	100	79.71%	Hit

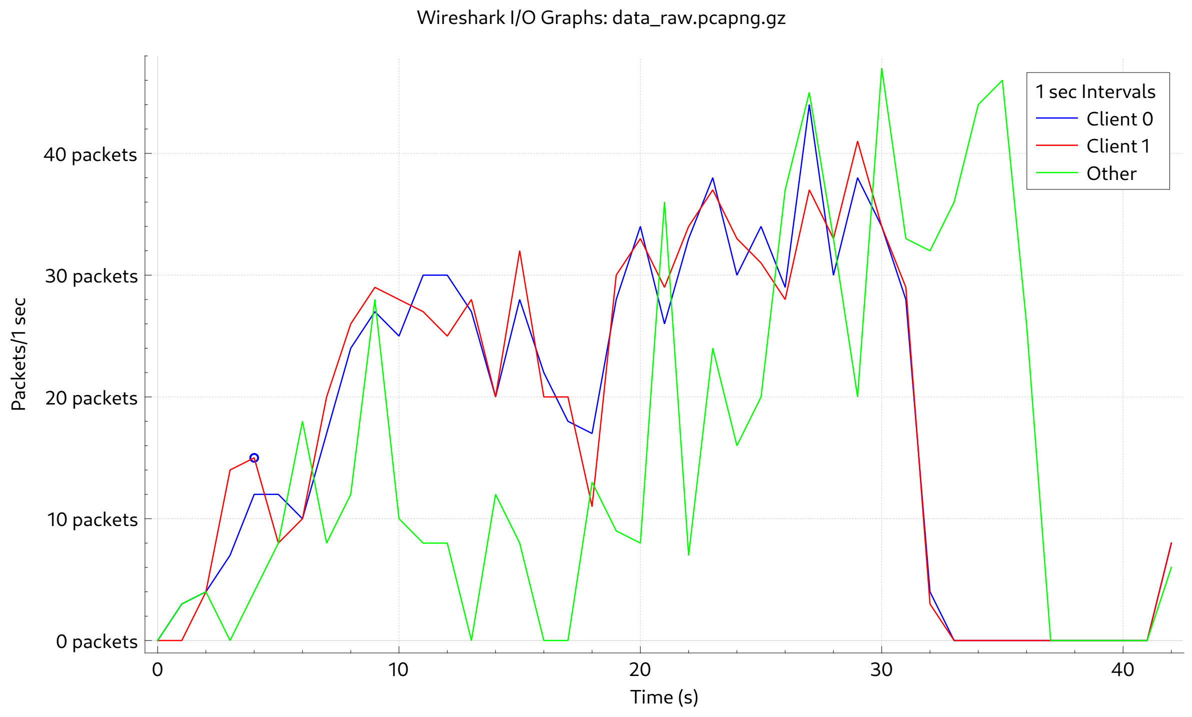
Table 3.2: Results for running selected NSDA to correlate client 0 and client 1

Using NSDA as a black box is not transparent, therefore to make sense of the results we graph out the network trace which the tool, using the two variations of the NSDA, was used on. The network trace behind each combination of iterations and client above can be depicted as an I/O graph with Wireshark[6]. The graphs shows the amount of packages uploaded and downloaded to and from the server over accumulated intervals of time where most are accumulated over a one second interval, but a two second interval for Figure 3.1e. The graphs depict the targeted client, client 0, and the expected client, client 1, which are the same two clients correlated in the tables above, and some other client. The traffic patterns of client 0 and client 1 can indicate whether they have communicated. When one client sends a message, the other receives it, resulting in both clients generating a similar volume of traffic in terms of packets. Therefore, communicating clients may generates similar I/O graphs indicating a correlation. This indication is not apparent for Figure 3.1a due to the low amount of iterations and a large amount of clients, hence the results show high uncertainty. The indication is likewise not apparent for the graphs depicted in Figure 3.1e. The rest of the data is more balanced between iterations and amount of clients, hence the results can also be seen to correlate much more accurately as expected from looking at the graphs.

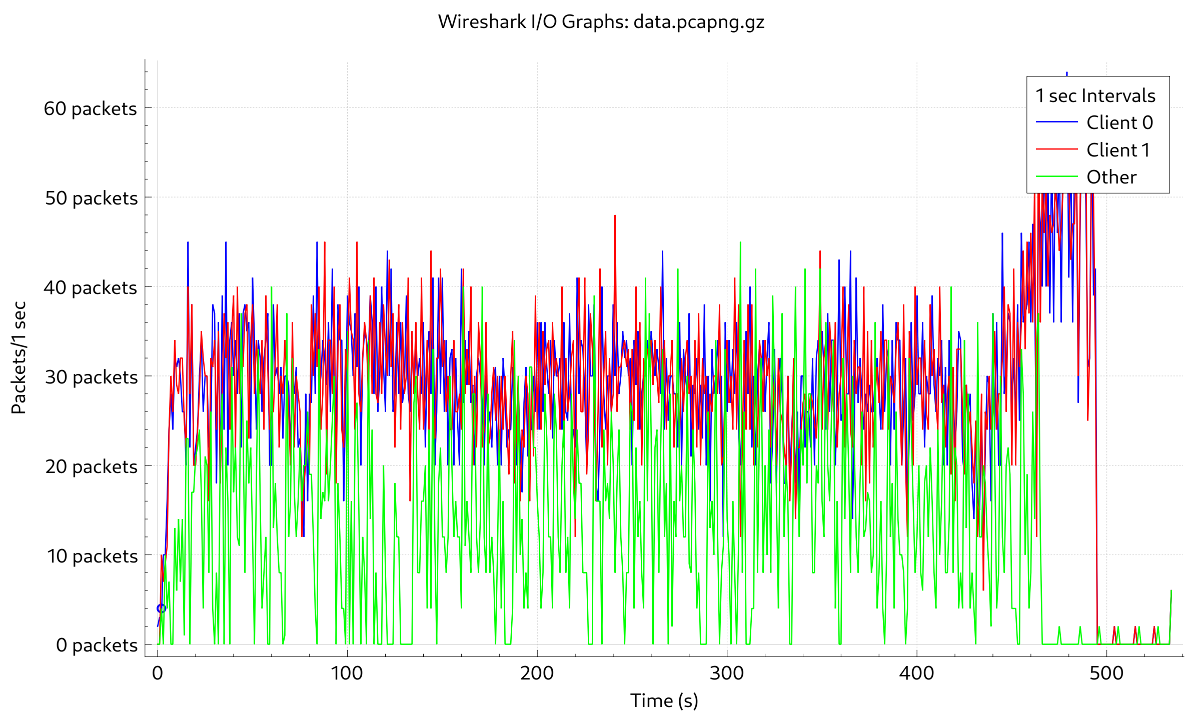
[6]https://www.wireshark.org/docs/wsug_html_chunked/ChStatIOGraphs.html



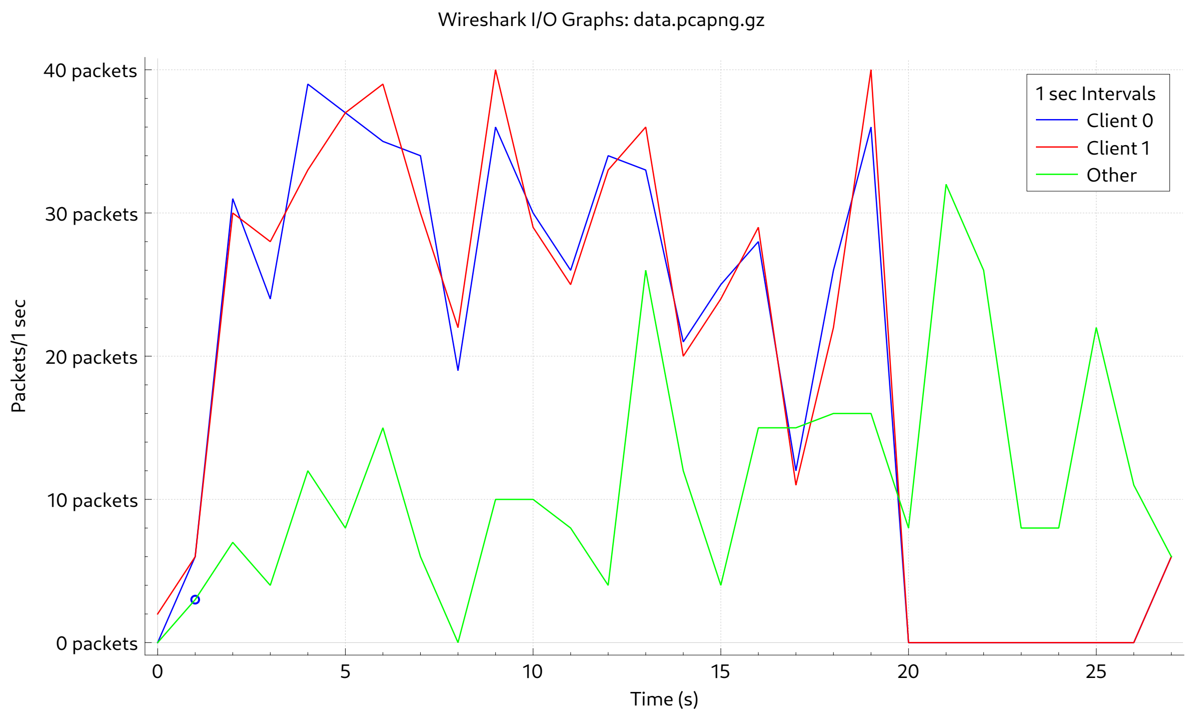
(a) Experiment with 100 clients and 10 rounds, showing the correlation of client 0 and client 1, the red and blue graph, whom we expect to have communicated.



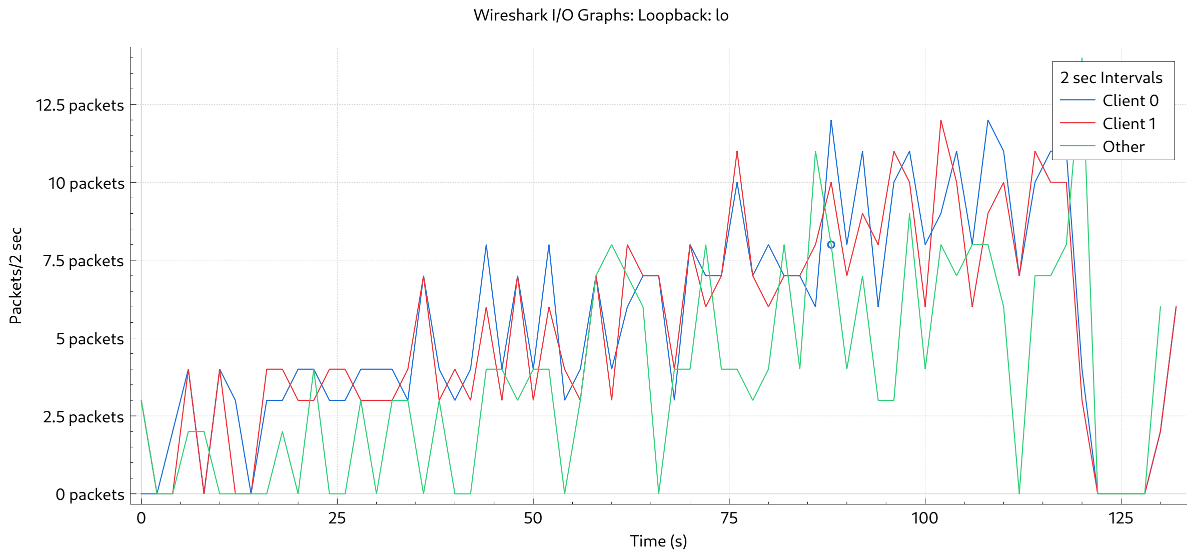
(b) Experiment with 100 clients and 100 rounds, showing the correlation of client 0 and client 1, the red and blue graph, whom we expect to have communicated.



(c) Experiment with 100 clients and 1000 rounds, showing the correlation of client 0 and client 1, the red and blue graph, whom we expect to have communicated.



(d) Experiment with 10 clients and 100 rounds, showing the correlation of client 0 and client 1, the red and blue graph, whom we expect to have communicated.



(e) Experiment with 1000 clients and 100 rounds, showing the correlation of client 0 and client 1, the red and blue graph, whom we expect to have communicated.

Figure 3.1: Graphs of client 0, client 1, and a third client's network packages sent and received, over the duration of the experiment, grouped in intervals

4. DenIM-on-Signal

In order to reason about DenIM's privacy guarantees it is imperative to understand the protocol itself. Therefore, this section gives an overview of the protocol and its threat model. Following the overview, we attack DenIM with the NSDA used for Signal to verify that DenIM fends off this attack. The attack is conducted on captured DenIM data, simulated with the DenIM implementation used in [11].

4.1 What is DenIM

DenIM is a protocol which provides impeccable privacy guarantees at the cost of either latency or bandwidth [11], the effect can be seen in Section 4.3. DenIM can provide these privacy guarantees through its use of deniable messages. Deniable messages has the property where the sender can deny ever having send the message. In DenIM, this is done by hiding the message on the transport layer by piggybacking them, resulting in the adversary never being able to observe the deniable messages sent. A deniable message can only be sent and received when it is piggybacked on a regular message. A regular message in DenIM is like a normal instant message with no privacy guarantees, but it is extended with a buffer which may contain a piggybacked deniable message. The buffer is filled with a deniable message if the user has one in his local deniable queue, otherwise it will use dummy data. The deniable queue stores sent deniable messages which has yet to be piggybacked on a regular message. The size of the deniable payload which can be piggybacked on a regular message is determined by a constant q which is known by every user. The size of the deniable payload is calculated as: $|d| = |r| * q$ where d is the deniable payload and r is the regular payload. A large q will then lead to a high bandwidth consumption and low latency, and a small q will then lead to a high latency but little bandwidth consumption. When a regular message reaches the server, the server splits the message into its regular message part and the deniable message part. The server then checks whether the intended recipient of the regular message has any deniable messages in their deniable queue. If the deniable queue is not empty the server pops the deniable queue and piggybacks that deniable message onto the regular message. In case the queue was empty the server piggybacks dummy data instead. Finally the server forwards the message to the intended recipient. When an incoming message contains a deniable message, that deniable message is queued for its intended recipient in their deniable queue on the server.

The threat model of DenIM assumes a Global Active Adversary, an entity capable of observing all communication between the server and the user. Additionally, the adversary can inject, modify, or replay messages, participate in the protocol itself, and even control nodes within the network. The adversary's objective is to uncover either the content of a deniable message or the existence of one. That includes determining whether a regular message carries a deniable payload or identifying if two clients are engaging in deniable communication. Despite the strength of the adversary model, DenIM operates under certain assumptions for its privacy guarantees to hold. Notably, it assumes that users trust the receivers of their deniable traffic. Furthermore, all forwarding servers are considered trustworthy and it is assumed that deniable behavior does not influence a user's regular behavior (see [11] for detail). The authors of DenIM has chosen to create the DenIM protocol on top of the Signal protocol due to the provided security guarantees [11]. [57]

4.2 Initializing the Simulations

The implementation of DenIM used for simulation is provided to us as it is open source from [11]. The implementation is used without modifications except for which contacts each client has, stressing this change having no effect on the DenIM protocol. To initialize the simulation the clients must be assigned two contact lists. The contact lists purpose is to dictate who the clients can converse with. One of the contact lists are for their regular contacts, while the other is for their deniable contacts.

To create the two contact lists, we begin by enumerating all the clients. Based on this enumeration, we generate two other lists by distributing clients with the even indices into one list and the clients with odd indices into the other list. These lists are organized into a matrix as its rows. We call this matrix the regular matrix and its transposed version the deniable matrix, and use these matrices to derive the contact lists. Each client's regular and deniable contacts are extracted from the regular and deniable matrix and coincides with the row containing their enumeration index.

An example with 10 clients would be:

$$R = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 \\ 1 & 3 & 5 & 7 & 9 \end{bmatrix},\ R^T = D = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \\ 8 & 9 \end{bmatrix}$$

Where R is the regular matrix containing regular contact lists and D is the deniable matrix containing deniable contact lists, and each row represents a contact list. From the example matrix the client with index 0 would have regular contracts as $[\ 0,\ 2,\ 4,\ 6,\ 8\ ]$ and deniable contacts as $[\ 0,\ 1\ ]$, lastly the clients remove themselves from their contacts.

4.3 Attacking DenIM

To attack DenIM we will perform the same attacks as used to attack the Signal implementation in Section 3.1. As can be seen by the results below, DenIM is resilient towards these attacks, and do not allow such attackers to disclose the users as expected. The results presented are based on simulations using either 10 or 20 clients. These values were chosen because a lower number of clients simplifies the correlation process. Therefore, when the algorithm fails to correlate even under these favorable conditions, it strengthens our confidence in the effectiveness of DenIM. However, there is one outlier (see †) in the below table, but we suspect this to be due to the small amount of clients used in the experiment. An example where an outlier would emerge is if client A and client B is communicating through deniable messages but A sends a message to C, and D sends a message to B. If these two events happen within the same epoch the two approaches would still correlate that A could have sent a message to B. Therefore, outliers are possible with these models, however they become less likely the more iterations and clients are introduced.

Epoch	**Duration**	**Clients**	**Belief**	**Hit/Miss**
$5 * 10^{-3}$s	2m	10	0.00%	Miss
$5 * 10^{-4}$s	2m	10	4.23%	Miss
$5 * 10^{-3}$s	2m	20	0.00%	Miss
$5 * 10^{-4}$s	2m	20	0.12%	Miss
$5 * 10^{-3}$s	4m	10	0.00%	Miss
$5 * 10^{-4}$s	4m	10	23.50%	Miss
$5 * 10^{-3}$s	4m	20	0.00%	Miss
$5 * 10^{-4}$s	4m	20	14.33%	Miss
$5 * 10^{-3}$s	8m	10	0.00%	Miss
$5 * 10^{-4}$s	8m	10	0.00%	Miss
$5 * 10^{-3}$s	8m	20	0.00%	Miss
$5 * 10^{-4}$s	8m	20	0.01%	Miss

Table 4.1: Results for running chunking NSDA to correlate client 0 and client 1 with $q = 0.36$

Epoch	**Duration**	**Clients**	**Belief**	**Hit/Miss**
$5 * 10^{-3}$s	2m	10	0.00%	Miss
$5 * 10^{-4}$s	2m	10	0.01%	Miss
$5 * 10^{-3}$s	2m	20	0.00%	Miss
$5 * 10^{-4}$s	2m	20	0.00%	Miss
$5 * 10^{-3}$s	4m	10	0.00%	Miss
$5 * 10^{-4}$s	4m	10	0.00%	Miss
$5 * 10^{-3}$s	4m	20	0.00%	Miss
$5 * 10^{-4}$s	4m	20	0.00%	Miss
$5 * 10^{-3}$s	8m	10	0.00%	Miss
$5 * 10^{-4}$s	8m	10	0.00%	Miss
$5 * 10^{-3}$s	8m	20	0.00%	Miss
$5 * 10^{-4}$s	8m	20	0.00%	Miss

Table 4.2: Results for running selected NSDA to correlate client 0 and client 1 with $q = 0.36$

5. DenIM's User Behavior Assumption

Using DenIM under their threat model provides strong privacy guarantees. However, we argue that for a practical solution, user behavior should not be a determining factor for privacy. This is because clients are able to act freely, making any assumption that they will behave in a particular way unrealistic, as behavior cannot be controlled.

An example where users carelessly could compromise their privacy is when they are in a hurry. A user familiar with the system would understand that the latency of their deniable messages is tied to their regular behavior. This is because the user knows that they have to forward their deniable message through piggybacking in order to send their deniable message. Consequently, if the user needs to quickly relay information through a deniable channel, they could generate a large volume of regular traffic, either by sending many smaller messages or one large message, to promptly forward their deniable messages to the server.

Therefore, for practical usage, we study DenIM in a scenario where users do not conform to the assumption *"Users' deniable behavior does not influence their regular behavior, [...]"* [11] and instead assume *"Users sending a deniable message may forward it immediately by sending more regular traffic to ensure that the entirety of the deniable message is forwarded to the server"*. Altering this assumption provides more lenience to the users' behavior, sacrificing the privacy guarantees. Chapter 6 analyzes the loss in privacy guarantees and in what degree this change affects user privacy, and how varying the server parameter q can keep user's private even under this new assumption.

6. Analysis

In this section we focus on analyzing the new threat model's effect on user privacy. We show the practical effects by attacking users using DenIM, where the users are now acting under the new assumption described in Chapter 5. We show that users can indeed be disclosed under the new assumptions but also reason about how to protect users through tweaking DenIM's q parameter.

6.1 Attacking DenIM Under the New User Behavior

To attack DenIM with users acting within the confines of the new defined assumption, we designed a new attack for the tool, inspired by the NSDA which is detailed in Section 3.1. We continue with the idea of collecting profiles of users and use Sinkhorn to perform a transformation on which we can disclose communication links. To effectively identify deniable communication links under DenIM with the new user behavior assumption we are going to focus on that new user behavior. This is because DenIM is designed to hide the deniable messages within the regular messages assuring that the regular messages which carries deniable messages are indistinguishable from regular messages which carries dummy data. Therefore it is insufficient to look at the send and receive events as is, because they reveal no information about the deniable messages. Instead, we now look for when users abusing their newfound freedom by forwarding deniable messages in a hurry, which in our case is when users send many smaller regular messages to forward their deniable ones.

Armed with this knowledge we can begin to look at users who sent a rapid stream of regular messages within a very short period of time, this behavior will be referred to as a burst. We can now attack based on the assumption that when we encounter a burst it must mean that the same user has also queued a deniable message shortly prior to the burst and tried to forward it with the burst. For the eventual receiver of that deniable message to actually read it, the message must still be forwarded from the server by other users, as such is DenIM. This means we now look for which client have received at least the same amount of regular messages as the size of the burst since it happened. We look for this as it would indicate that the receiver could have received the entire deniable message if it was the target for that deniable message. If that receiver then follows these receive events by sending a burst shortly after, we can then reason that the two are communicating since this pattern would indicate that a reply was sent out. However, to confirm this we wait for the original sender to receive at least the amount of regular messages from the reply burst, and then look for another burst from the original sender. To summarize this pattern with an example we look for a user Alice sending a burst, denoted as A of length 2, user Bob following receives twice, denoted for each as b. Bob then sends out a burst of length 3, denoted as B, and then Alice receives those messages from other users, denoted a for each, and sends another burst A. The pattern we are looking for would then be denoted as $[A, b, b, B, a, a, a, A]$. This pattern would then indicate a possible conversation between Alice and Bob.

The pattern relies on the assumption that in the use case of DenIM, even deniable traffic will have similar patterns as the regular traffic, meaning it will be used for shorter text messages. We also assume that these regular and deniable messages would be of equal length on average. For example if you are living under an authoritarian regime but want to speak private and freely with other people, DenIM could be used for that.

The main difference between this proposed attack and the original NSDA attack lies in the profiling step and the fact that we no longer use the sender set directly for profiling. The sender set is now used exclusively to identify where bursts occur. The receiver set remains unchanged, containing all receive events that occurred in the trace.

To identify burst we assume that each small message in a burst is sent within a short period of time, call this dt. Therefore, we look at each message and for each we evaluate if the next message from that sender happened within dt time, if so we group the two and call them a burst. We continue from the group to see if the time of the last message is within dt of the next message sent by that sender, if so we include it in the group, and if not it will be seen as the start of a potential new burst. Finally, this will give us the whole burst, a set where each message is from the same sender and each message happened within dt time of each other. Therefore, doing this for each message we end up with all bursts in the network trace. All identified bursts are collected into a burst set, which consists of pairs of senders and the timestamps at which the bursts occurred, here we have chosen to identify when a burst happened using the timestamp of the last message in the set. We decided to use the last timestamp in the burst because this is the moment when the deniable message is in total on the server. Note that if we chose the first timestamp we would have the time when the deniable message was sent instead, we elaborate on the differences between the two approaches and the potential benefits of using the first timestamp in Chapter 8. However, the research does not benefit from this change and therefore we stick with this approach.

Moving on, we use both the burst set and the receiver set such that we can efficiently search for the pattern described earlier in this section. To find this pattern, we start by selecting a user and identifying their first two bursts. The timestamps of these bursts is used as a window of time wherein we search for a sub-pattern in the network trace. Continuing from the earlier example, this sub-pattern might look like $[\,x, x, X, a, a, a\,]$, where x represents any client. Clients matching the sub-pattern is considered potential communication partners of Alice.

To identify such sub-patterns, we first locate all bursts that occur within the time window. Note that none of these bursts originate from the sender, sender being the user who sent the bursts defining the time window. For each of these other clients, we check whether they received the amount of messages sent in the first burst of the window before they reply with a new burst (within the time window). Additionally, the sender is also checked such that it received the amount of messages as was in the new burst after that burst was sent, since otherwise it could not have sent it as a reply. Clients matching the sub-pattern are added to a list R_i, which represents the possible receivers of the message. The approach to finding such sub patterns is also showed in algorithm 4. Using this list, we then update the sender's s profile $P(s, r)$ as follows:

$$P'(s,r) = P(s,r) + \frac{1}{|R_i|} \; where \; r \in R_i, \; s \in S \tag{6.1}$$

The entire attack is depicting in algorithm 3.

We continue updating the profiles for each client picking two bursts as: $succ^n(B, s)$ and $succ^{n+1}(B, s)$ where $s \in S$ and B is the set of all bursts. On the last bursts from the client where $succ^{n+2}(B, s) = \emptyset$ we instead use the end of the network trace as timestamp instead of one from a burst. After computing all profile updates we can then continue on as the normal NSDA approach to disclose the communication partners.

Algorithm 3: DenIM Disclosure Attack

```
Input: burst_events, receive_events
Result: A matrix describing each clients profile
/* Each cell in the profiles matrix maps to 0                    */
$P(s, r) \mapsto 0$
for $i \leftarrow 0$ to $|burst\_events|$ do
    /* s is the sender of the burst                              */
    $s \leftarrow burst\_events[i]$
    for $j \leftarrow i$ to $|burst\_events|$ do
        if $s.id \neq burst\_events[j].id$ then
            continue
        end
        $start \leftarrow s.time$
        $end \leftarrow burst\_events[j].time$

        $burst\_events\_slice \leftarrow get\_events\_between(burst\_events, start, end)$
        $receive\_events\_slice \leftarrow get\_events\_between(receive\_events, start, end)$
        $R_i \leftarrow filter\_receivers(burst\_events\_slice, receive\_events\_slice, s.id, s.size)$

        foreach $r \in R_i$ do
            $P[s.id][r.id]' \leftarrow P[s.id][r.id] + \frac{1}{|R_i|}$
        end
        break
    end
end
return $Sinkhorn(P)$
```

Algorithm 4: The pseudo code for filter_receivers

```
Input: burst_events, receive_events, id, size
Result: $R_i$
$burst\_events' \leftarrow reverse(burst\_events)$
$receive\_events' \leftarrow reverse(receive\_events)$
$i \leftarrow 0$
$count \leftarrow 0$
$counts \leftarrow \emptyset$
foreach $receiver \in receive\_events$ do
    while $i < |burst\_events| \wedge receriver.time < burst\_events[i].time$ do
        $burst\_event \leftarrow burst\_events[i]$
        if $burst\_event.id \notin counts \wedge burst\_event.size \leq count$ then
            $counts[burst\_event.id] \leftarrow 0$
        end
        $i' \leftarrow i + 1$
    end

    if $receiver.id = id$ then
        $count' \leftarrow count + 1$
    else if $receiver.id \in counts$ then
        $counts[receiver.id]' \leftarrow counts[receiver.id] + 1$
    end
end
$potential\_receivers \leftarrow filter(burst\_events, burst\_event \rightarrow counts[burst\_event.id] \geq size))$
return $remove\_duplicates(potential\_receivers)$
```

6.2 Results

In the section, our objective is to evaluate privacy under the updated threat model, detailed in Chapter 5. This is achieved by varying the value of q and observing its effect on privacy. We simulate conversations where clients may communicate both regularly and deniably. We vary three key parameters: the value of q, the number of clients, and the proportion of clients attempting to forward their deniable messages immediately, in accordance with the new threat model. The resulting network traffic from these simulations is captured using Wireshark and analyzed using the tool by executing the attack described in Section 6.1.

The simulation data is generated on a single machine, with each client and server running in separate node environments. This setup promotes parallelism and helps maintain client independence. However, as discussed earlier, all operations are ultimately scheduled by the host operating system. Consequently, we cannot guarantee full parallelism or complete independence. This limitation may introduce minor timing artifacts due to OS-level context switching, however we did not experience any problems in the simulation due to this. Furthermore, the number of clients we are able to simulate is limited due to the high memory usage of the current DenIM implementation. Simulating 20 clients can consume up to 15 GB of memory. As a result, a more in-depth study of the effects of scaling the number of clients is left for future work, when a more memory-efficient version of DenIM becomes available.

The effectiveness of the attack depends on a parameter dt, which defines the maximum allowable time between two messages for them to be considered part of the same burst. Choosing a large dt may cause unrelated messages to be grouped into a single burst (false positives), while a small dt may split actual bursts (false negatives). Thus, it is crucial to choose a dt to be small enough so that it would not include false positive bursts, yet big enough to include as many of the correct bursts.

The results demonstrate that clients who eagerly forward their deniable messages are at risk of being identified and that the value of q has an effect on the privacy of the system. The impact of q having an effect on privacy is discussed further in Chapter 7. In Table 6.1, we show the correlation results between client 0 and client 1, who are communicating deniably. In this simulation, only these two clients are actively attempting to forward their deniable messages immediately. We observe that they can be reliably identified up to $q = 0.96$, implying that values of q above this threshold are necessary to ensure their privacy. Consequently, this implies an overhead exceeding 96% on all messages, which is substantial in practical use cases.

In Table 6.4, we present a scenario in which all clients eagerly forward their deniable traffic. This creates significant noise, making it more challenging for an attacker to correlate client 0 and client 1. Nevertheless, even in this noisier setting, we are able to correlate the two clients reliably up to $q = 0.36$. Thus, to preserve privacy in this scenario, q must exceed 0.36, still implying a significant overhead of more than 36% on all messages.

The optimal value of q therefore depends on the system's intended use. A conservative configuration aiming for stronger privacy should choose $q > 0.96$, while a more cost-aware configuration may settle for $q > 0.36$. However, in realistic conditions, it is unlikely that all clients will simultaneously attempt to forward their deniable messages eagerly. Therefore, a practical value for q likely lies somewhere between 0.36 and 0.96.

	Run 1		**Run 2**		**Run 3**	
q	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**
0.12	94.46%	Hit	98.98%	Hit	95.07%	Hit
0.24	86.45%	Hit	90.06%	Hit	98.75%	Hit
0.36	99.60%	Hit	87.22%	Hit	93.47%	Hit
0.48	97.63%	Hit	92.54%	Hit	92.61%	Hit
0.60	95.39%	Hit	94.77%	Hit	91.07%	Hit
0.72	79.98%	Hit	97.34%	Hit	93.81%	Hit
0.84	30.73%	Hit	44.58%	Hit	56.47%	Hit
0.96	22.88%	Hit	0.00%	Miss	28.93%	Hit
1.08	11.76%	Miss	13.69%	Miss	0.00%	Miss
1.20	0.00%	Miss	27.07%	Hit	0.00%	Miss

Table 6.1: Correlation of client 0 and client 1 where only these two clients forwards their deniable traffic through bursts with the experiment running over a duration of 8 minutes with 10 clients and $dt = 0.2$.

	Run 1		**Run 2**		**Run 3**	
q	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**
0.12	2.18%	Miss	0.56%	Miss	9.71%	Miss
0.24	75.31%	Hit	79.47%	Hit	48.40%	Hit
0.36	95.15%	Hit	69.39%	Hit	85.14%	Hit
0.48	68.40%	Hit	70.05%	Hit	45.19%	Hit
0.60	45.23%	Hit	18.56%	Miss	25.68%	Hit
0.72	7.33%	Miss	4.42%	Miss	10.05%	Miss
0.84	15.22%	Miss	10.04%	Miss	10.67%	Miss
0.96	12.94%	Miss	0.00%	Miss	0.00%	Miss
1.08	0.00%	Miss	11.59%	Miss	0.00%	Miss
1.20	0.00%	Miss	0.00%	Miss	0.00%	Miss

Table 6.2: Correlation of client 0 and client 1 where every client forwards their deniable traffic through bursts with the experiment running for 8 minutes with 10 clients and $dt = 0.2$.

	Run 1		**Run 2**		**Run 3**	
q	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**
0.12	94.34%	Hit	86.99%	Hit	87.31%	Hit
0.24	89.18%	Hit	85.71%	Hit	90.98%	Hit
0.36	84.74%	Hit	84.83%	Hit	87.04%	Hit
0.48	84.79%	Hit	86.98%	Hit	86.77%	Hit
0.60	83.39%	Hit	84.28%	Hit	88.97%	Hit
0.72	82.84%	Hit	60.71%	Hit	30.31%	Hit
0.84	10.50%	Miss	11.06%	Hit	5.49%	Miss
0.96	4.82%	Miss	5.31%	Miss	4.34%	Miss
1.08	5.15%	Miss	4.76%	Miss	6.04%	Miss
1.20	4.44%	Miss	6.70%	Miss	6.45%	Hit

Table 6.3: Correlation of client 0 and client 1 where only these two clients forwards their deniable traffic through bursts with the experiment running for 8 minutes with 20 clients and $dt = 0.2$.

	Run 1		Run 2		Run 3	
q	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**	**Belief**	**Hit/Miss**
0.12	6.58%	Miss	0.51%	Miss	13.94%	Miss
0.24	35.07%	Hit	25.72%	Hit	2.83%	Miss
0.36	69.77%	Hit	41.07%	Hit	17.64%	Hit
0.48	11.71%	Miss	13.74%	Miss	20.44%	Hit
0.60	3.17%	Miss	14.52%	Hit	11.02%	Hit
0.72	4.63%	Miss	7.68%	Miss	6.35%	Miss
0.84	5.98%	Hit	6.47%	Miss	5.31%	Miss
0.96	4.76%	Miss	4.12%	Miss	5.37%	Miss
1.08	5.18%	Miss	5.65%	Miss	5.34%	Miss
1.20	5.26%	Miss	5.84%	Miss	5.93%	Miss

Table 6.4: Correlation of client 0 and client 1 where all forwards their deniable traffic through bursts with the experiment running for 8 minutes with 20 clients and $dt = 0.2$.

Table 6.2 and Table 6.4 showed results with $q = 0.12$ which we did not expect. Logically, larger bursts leads to easier detection and therefore the results are perplexing. However, Table 6.5 shows an experiment with $q = 0.12$ and the deniable participation population varying. Here it becomes apart that after a larger percentage of the population engages with the deniable communication the correlation of bursts and users muddies, resulting in low beliefs. The results show that 60% of the population can participate in deniable communication while doing bursts for the attack to be believable at $q = 0.12$.

Population	**Belief**	**Hit/Miss**
100%	2.18%	Miss
80%	8.34%	Miss
60%	71.81%	Hit
40%	61.17%	Hit
20%	94.46%	Hit

Table 6.5: Correlation of client 0 and client 1 by varying the amount of clients who forwards their deniable traffic through bursts with the experiment running over a duration of 8 minutes with 10 clients, $dt = 0.2$ and $q = 0.12$.

7. Discussion

The data that we have generated are generated on the same machine, which can potentially yield som issues, as discussed in both Section 3.1 and Chapter 6. So the obvious choice to make it more realistic is to run each client on separate machines, that way they cannot in any way be dependent of each other, which would yield better results. But we doubt that the results would change that much, since the effect from the OS is minimal.

According to our research and the study [11] has conducted, DenIM does indeed provide impeccable privacy given a certain user behavior. However, if the users does not conform to this behavior, but lets their deniable behavior influence their regular behavior, they may loose privacy or the system would have to sustain a large overhead to keep the privacy. In order for DenIM where users follow the new user behavior assumption to work in practice, the value of q must be large enough so it does not compromise their privacy. According to our study, the value of q should at least be larger than 0.36 to not compromise the users privacy and can be set higher to increase the privacy. However, this should still be studied on a larger scale to be able to say anything more conclusive. To study this on a larger scale a more scalable implementation of DenIM is required, we know of two implementations which is currently in the midst of being developed, one trying to implement a production DenIM server which facilitates many client connections at a time [15], and the other is a simulation tool which supports at least 500 clients [16].

In Chapter 6, we discussed the design of the attack on DenIM and briefly mentioned our choice to use the last timestamp of a burst for executing the attack. We asserted that this choice would not impact the results. In the following, we provide a more detailed explanation of the differences involved, thereby supporting the validity of our claim.

Firstly, the difference of using the first or last timestamp is clearly what happens while the burst is being sent. Therefore, if some messages in a burst has a significant delay some receive events could have happened within the time of the burst. Going off the notation used in Chapter 6 to describe the pattern that we are looking for, we need additional notation to identify each individual message within each burst. We therefore denote a message for a burst from A as $A_{x,y}$, where x is a counter of bursts sent by A, and y is the counter of messages within the burst x. Thus, we can construct an example of where using the first timestamp is critical as: $[A_{1,1},\ A_{1,2},\ b,\ A_{1,3},\ b,\ b,\ B,\ a,\ a,\ A]$. In the example we can see that user Bob is expecting to receive 3 messages in order to have received Alice's deniable message, since it took 3 messages for Alice to send out the deniable message (still working under the assumption that messages are of equal length). Consequently, using the last timestamp Bob could not have received the message according to our attack, where in fact he could since the first receive event for Bob could receive either $A_{1,1}$ or $A_{1,2}$.

When using the first timestamp there is a catch, the attacker now also have to keep track of how many receive events happened intra burst. An example of where an issue is apparent, when changing the current algorithm to use the first timestamp, could be denoted as: $[A_{1,1},\ b,\ b,\ A_{1,2},\ A_{1,3},\ b,\ B,\ a,\ a,\ A]$. In the given example when the attacker expects Bob to receive three times, they would therefore mark Bob as a possible recipient of Alice's burst. However, since two of Bob's receive events happened between $A_{1,1}$ and $A_{1,2}$ one of the two events could only have forwarded $A_{1,1}$, and none of them could have transferred $A_{1,2}$ or $A_{1,3}$. Therefore in no instance could Bob have received all three messages from Alice and it must have been another user who Alice was conversing with in this scenario. Therefore, a possible improvement is using the first timestamp and adding more logic to the attack to circumvent this issue.

8. Future Work

Currently the analysis tool uses the last timestamp of a burst to determine when the burst occurred, but as discussed in Chapter 7, this may impose some limitations on who we can correlate. Therefore we suggest as future work to change the analysis tool to use the first timestamp of a burst for a more robust algorithm.

As discussed in Section 6.2, we had some issues with running simulations with many clients due to the memory consumption. Therefore we suggest that future work could focus on finding a way to conduct these experiments with more clients. As mentioned in Chapter 7, we know of two other implementations under development which could allow for this, namely [15] and [16]

The attacks within our analysis tool that we have created, both the one we applied on Signal and DenIM, expects a CSV file representing the network trace to be analyzed, i.e. it is designed to work offline. Future work could focus on finding a way to run these analysis tools online, so whilst the network trace is being captured. This could either be done by using TShark directly in the analysis tool, for example with pyshark[1].

The experiments we conducted on Signal was with a group size of 2 to 5 clients, meaning that a client only has 1 to 4 other friends (not including himself) he can have a conversation with. These numbers where gotten from a research focusing on inter and intra company communication [14]. Private communication may differ, but at the time of writing we where unable to find any research in that regard. In the experiments we conducted on DenIM we created the groups as described in Section 4.2. The issue with this approach is that the regular groups would grow with the amount of clients and eventually become very large, but the deniable groups will always consist of two clients. Therefore we propose as future work to find a better way to create these groups/contact lists.

Currently our analysis tool only works if the IP of a client does not change during the network capture. This also means that the analysis tool is not suitable for roaming mobile devices, which may change IP frequently depending on which telecommunication mast is closest. As future work we suggest to change this analysis tool to make it work on roaming mobile devices. One way to accomplish this is with finger printing.

[1] `https://github.com/KimiNewt/pyshark/`

9. Related Work

Vuvuzela [8] is a privacy solution solving a similar problem as DenIM using a round based protocol. The way in which Vuvuzela provides its privacy is through a large amount of cover traffic and a constant participation in the protocol. Vuvuzela therefore provides a service for private communication, but due to the nature of a round based protocol it cannot be integrate on top of existing IM services, unlike DenIM.

Signal's sealed sender[1] is a privacy solution made by Signal which tries to hide the sender of a message from the server, but not the receiver. Specifically it hides the sending client's UUID from the server, but not the IP address. Thus, sealed sender is designed to protect users from a potentially malicious service provider, yet does not provide any additional privacy against external attackers.

Tor [17] is a privacy focused onion browser. An onion browser is a browser connected to an onion network, which is a network with multiple relay nodes, which will relay your request forward in the network to a possible other relay node or the destination. This also means there can be many hops from source to destination which can affect performance.

A method for attacking signal's sealed sender was proposed by [12] to correlate senders with receivers. The authors utilize a modified version of the SDA which they show can reveal sender and receiver relationships over time. The attack they propose shows an approach where they modify an existing attack on round based protocols such that it works for continuous networks.

The Perfect matching disclosure attack [5] proposed alongside the NSDA, is a variant of the statistical disclosure attack [4]. It works similar to NSDA, but instead of normalizing the matrix it computes an NP-hard problem to associate every single sender and receiver. This attack can achieve better results that NSDA, but compared to the heavy computation it is properly not worth it.

[1]Read more about sealed sender on Signal's blog: `https://signal.org/blog/sealed-sender/`

10. Conclusion

This study introduces a tool containing different disclosure attacks, which is publicly available on our GitHub[1], see Chapter 2 for a in-depth explanation of the tool. We have evaluated this tool throughout this report by disclosing different instant messaging systems. In Chapter 3 we propose two methods to attack normal communication channels such as Signal, detailed in Section 3.1, which are available in the tool. From the test conducted it is clear that Signal is not private, as the two different attacks disclosed communicating users. In Chapter 4 we have tested the privacy of DenIM by attacking it with the same attacks as used on Signal. This showed that DenIM is resistant to traffic analysis attacks by keeping the deniable traffic private. In Chapter 6 we introduced a novel method, available in the tool, identifying deniable conversation links for users using the DenIM protocol under a new assumption, proposed in Chapter 5. Under the new assumptions it is evident that the strong privacy guarantees stated by DenIM starts to crack, since we are able to disclose deniable traffic. In Chapter 6 we successfully disclose the deniable communications with the new assumption, but the privacy of DenIM is not completely lost under this new user behavior assumption. The tests shows that the value of q not just related to bandwidth and latency but also has an impact on the systems privacy. Furthermore, given a large value of q users should still be private whilst following this new user behavior model, our tests shows that the value of q should be at least greater than 0.36 and for a more conservative one could choose a value above 0.96.

[1]https://github.com/cs-25-ds-10-08/Traffic-Analysis-Tools

Reference

[1] Meta. *Messenger End-to-End Encryption Overview.* [Accessed 2025-06-03]. 2023. URL: https://engineering.fb.com/wp-content/uploads/2023/12/MessengerEnd-to-EndEncryptionOverview_12-6-2023.pdf.

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

[3] Moxie Marlinspike. *The Double Ratchet Algorithm.* Ed. by Trevor Perrin. [Accessed 2025-01-04]. Nov. 2016. URL: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf.

[4] George Danezis. "Statistical disclosure attacks: Traffic confirmation in open environments". In: *Security and Privacy in the Age of Uncertainty: IFIP TC11 18 th International Conference on Information Security (SEC2003) May 26–28, 2003, Athens, Greece 18.* Springer. 2003, pp. 421–426.

[5] Carmela Troncoso, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. "Perfect matching disclosure attacks". In: *Privacy Enhancing Technologies: 8th International Symposium, PETS 2008 Leuven, Belgium, July 23-25, 2008 Proceedings 8.* Springer. 2008, pp. 2–23.

[6] Johns Hopkins University. *The Johns Hopkins Foreign Affairs Symposium Presents: The Price of Privacy: Re-Evaluating the NSA.* https://www.youtube.com/watch?v=kV2HDM86XgI. [Accessed 03-06-2025]. 2014.

[7] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency-choose two". In: *2018 IEEE Symposium on Security and Privacy (SP).* IEEE. 2018, pp. 108–126.

[8] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. "Vuvuzela: scalable private messaging resistant to traffic analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles.* SOSP '15. Monterey, California: Association for Computing Machinery, 2015, pp. 137–152. ISBN: 9781450338349. DOI: 10.1145/2815400.2815417. URL: https://doi.org/10.1145/2815400.2815417.

[9] David Lazar, Yossi Gilad, and Nickolai Zeldovich. "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18).* [Accessed 2025-01-04]. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 711–725. ISBN: 978-1-939133-08-3. URL: https://www.usenix.org/conference/osdi18/presentation/lazar.

[10] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. "Stadium: A Distributed Metadata-Private Messaging System". In: *Proceedings of the 26th Symposium on Operating Systems Principles.* SOSP '17. [Accessed 2025-01-04]. Shanghai, China: Association for Computing Machinery, 2017, pp. 423–440. ISBN: 9781450350853. DOI: 10.1145/3132747.3132783. URL: https://doi.org/10.1145/3132747.3132783.

[11] Boel Nelson, Elena Pagnin, and Aslan Askarov. "Metadata Privacy Beyond Tunneling for Instant Messaging". In: *9th IEEE European Symposium on Security and Privacy, EuroS&P 2024, Vienna, Austria, July 8-12, 2024.* [Accessed 2025-01-04]. IEEE, 2024, pp. 697–723. DOI: 10.1109/EUROSP60621.2024.00044. URL: https://doi.org/10.1109/EuroSP60621.2024.00044.

[12] Ian Martiny, Gabriel Kaptchuk, Adam J. Aviv, Daniel S. Roche, and Eric Wustrow. “Improving Signal’s Sealed Sender”. In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021.* [Accessed 2025-01-04]. The Internet Society, 2021. URL: https://www.ndss-symposium.org/ndss-paper/improving-signals-sealed-sender/.

[13] Richard Sinkhorn. “A relationship between arbitrary positive matrices and doubly stochastic matrices”. In: *The annals of mathematical statistics* 35.2 (1964), pp. 876–879.

[14] Zhen Xiao, Lei Guo, and John Tracey. “Understanding Instant Messaging Traffic Characteristics”. In: July 2007, pp. 51–51. ISBN: 0-7695-2837-3. DOI: 10.1109/ICDCS.2007.149.

[15] Magnus Jørgsensen Harder Christensen Alex Skytt Steffensen and Simon Deleuran Laursen. *DenIM on SAM*. 2025.

[16] Arthur Osnes Gottlieb Andreas Knudsen Alstrup and Martin de Fries Justinussen. *Yet Another Privacy Protocol End-user Reveal*. 2025.

[17] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. “Tor: The second-generation onion router.” In: *USENIX security symposium*. Vol. 4. 2004, pp. 303–320.