This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

I Still Know What You Did Last Summer: Inferring Sensitive User Activities on Messaging Applications Through Traffic Analysis

Ardavan Bozorgi, Alireza Bahramali, Fateme Rezaei, Amirhossein Ghafari, Amir Houmansadr, Ramin Soltani, Dennis Goeckel, Don Towsley *Fellow, IEEE*

Abstract—Instant Messaging (IM) applications such as Signal, Telegram, and WhatsApp have become tremendously popular in recent years. Unfortunately, such IM services have been targets of governmental surveillance and censorship, as these services are home to public and private communications on socially and politically sensitive topics. To protect their clients, popular IM services deploy state-of-the-art encryption. Despite the use of advanced encryption, we show that popular IM applications leak sensitive information about their clients to adversaries merely monitoring their encrypted IM traffic, with no need for leveraging any software vulnerabilities of IM applications. Specifically, we devise traffic analysis attacks enabling an adversary to identify participants of target IM communications (e.g., forums) with high accuracies. We believe that our study demonstrates a significant, real-world threat to the users of such services.

We demonstrate the practicality of our attacks through extensive experiments on real-world IM communications. We show that standard countermeasure techniques can degrade the effectiveness of these attacks. We hope our study will encourage IM providers to integrate effective traffic obfuscation into their software. In the meantime, we have designed a countermeasure system, called IMProxy that can be used by IM clients with no need for any support from IM providers. We demonstrate the effectiveness of IMProxy through simulation and experiments.

Index Terms—Traffic Analysis, Secure Messaging Applications, Flow Correlation

1 Introduction

Instant Messaging (IM) applications such as Signal [86], Telegram [95], and WhatsApp [107] have become enormously popular in recent years. Recent studies estimate that over 3 billion people use mobile IM applications across the world [43]. IM services enable users to form private and public social groups and exchange messages of various types, including text messages, images, videos, and audio files. In particular, IM applications are used extensively to exchange politically and socially sensitive content. As a result, governments and corporations increasingly monitor the communications made through popular IM services [2], [3], [79], [97].

A notable example of oppressed IM services is Telegram with over 500 million users globally [67], where a large fraction of its users come from countries with strict media regulations like Iran and Russia. In particular, Telegram is so popular in Iran that it has been estimated to consume more than 60 percent of Iran's Internet bandwidth [11]. Consequently, Iranian officials have taken various measures to monitor and block Telegram: from requesting Telegram to host some of its servers inside Iran to enable surveillance [97], to requesting Telegram to remove controversial political and non-political channels [97]. Eventually, Iran blocked Telegram entirely in April 2018 due to Telegram's non-compliance. Despite this, statistics suggest only a small decrease in Telegram's Iranian users who connect to it through various kinds of VPNs [44]. Telegram has also been blocked in Russia as Telegram operators refrained from handing over their encryption keys to Russian officials for surveillance [79]. Finally, in the light of Telegram's crucial role in recent Hong Kong protests, there are unconfirmed reports [24], [81] that mainland Chinese and Hong Kong authorities may have attempted to discover Hong Kong protesters by misusing a Telegram feature that enabled them to map phone numbers to Telegram IDs.

Signal is another example of IM applications with over 40 million monthly active users. Known for its privacy and security considerations, Signal has been the communication method of choice for activists, people in the hacker community, and others concerned about privacy. [87] In January 2021, due the concerns over a notification of updated terms of service from WhatsApp application (which is the most popular IM in the world), Signal started to gain popularity as people are increasingly concerned with safeguarding their private information [89]. What makes Signal one of the leading SIMs is its efforts to minimize the amount of metadata each message leaves behind in addition to hiding the content of the messages. In this regard, Signal recently has deployed a feature called *Sealed Sender* [90] that conceals the identity of the sender of messages. In January 2021, the Iranian government started blocking all Signal traffic. In response, Signal added support for a simple TLS proxy in its Android version in order to let user bypass the network block. [45]

A Fundamental Vulnerability: Popular IM applications such as Signal, Telegram, and WhatsApp, deploy encryption (either end-to-end or end-to-middle) to secure user communications. We refer to such services as *secure IM (SIM)* applications. In this paper, we demonstrate that *despite their*

use of advanced encryption, popular IM applications leak sensitive information about their clients' activities to surveillance parties. Specifically, we demonstrate that surveillance parties are capable of identifying participants of target IM communications (e.g., politically sensitive IM channels) with very high accuracies, and by only using low-cost traffic analysis techniques. Note that our attacks are *not* due to security flaws or buggy software implementations such as those discovered previously [33], [51], [80], [113]; while important, such security flaws are scarce, and are immediately fixed by IM providers once discovered. Instead, our attacks enable surveillance by *merely watching encrypted IM traffic* of IM users, and assuming that the underlying IM software is entirely secure. The key enabler of our attacks is the fact that major IM operators *do not* deploy effective mechanisms to obfuscate traffic characteristics (e.g., packet timing and sizes), due to the impact of obfuscation on the usability and performance of such services. We therefore argue that *our attacks demonstrate a fundamental vulnerability in major in-the-wild IM services*, and, as we will demonstrate, they work against all major IM services.

We believe that our attacks present *significant real-world threats* to the users of believed-to-be-secure IM services, specially given escalating attempts by oppressive regimes to crack down on such services, e.g., the recent attempts [2], [3], [24], [81] to identify and seize the participants of controversial IM communications.

Our Contributions: We design traffic analysis attack algorithms for SIM communications; the objective of our attack is to *identify the participants* of target SIM communications. What enables our attack is that, *widely-used SIM services do not employ enough mechanisms to obfuscate statistical characteristics of their communications*.

We start by establishing a statistical model for IM traffic characteristics. Such a model is essential in our search for effective traffic analysis attacks on SIM services. To model IM communications, we join over 1,000 public Telegram channels and record their communications, based on which we derive a statistical model for IM traffic features.

Based on our statistical model for IM communications, we use hypothesis testing [74] to *systematically* design effective traffic analysis attack algorithms. Specifically, we design two traffic analysis attack algorithms; our first algorithm, which we call the *event-based* algorithm, relies on the statistical model that we derive for SIM communications to offer an optimal matching of users to communications. Our second algorithm, which we call the *shape-based* algorithm, correlates the shapes of SIM traffic flows in order to match users to target communications. Our shape-based algorithm is slower but offers more accurate detection performance than the event-based algorithm for smaller values of false positive rates. In practice, the adversary can cascade the two algorithms to optimize computation cost (and scalability) versus detection performance. Note that, as demonstrated through experiments, our statistical detectors outperform deep learning based detectors trained on IM traffic when the IM service has not deployed effective obfuscation. This is because, as also demonstrated in recent work [63], deep learning traffic classifiers outperform statistical classifiers *only* in network applications with non-stationary noise conditions (e.g., Tor), where statistical models becomes unreliable. These attacks are closest in nature to flow correlation attacks.

We perform extensive experiments on live traffic of 5 popular SIMs to evaluate the performance of our attacks. Signal, Telegram, WhatsApp, Wickr [108], and Wire [110] are our target SIMs. We demonstrate that our algorithms offer extremely high accuracies in disclosing the participants of target SIM communications. In particular, we show that *only 15 minutes* of Telegram traffic suffices for our shape-based detector to identify the participant of a target SIM communication with a 94% true positive (TP) and a 10^{-3} false positive (FP) rate—the adversary can reduce the FP rate to 5×10^{-5} by observing an hour of traffic (the adversary can do this hierarchically, e.g., by monitoring the users flagged when using 15 minutes of traffic for longer traffic intervals). Using our event-based detector on 15 minutes of captured Signal traffic, we reach a 93% TP rate and a 6×10^{-3} FP rate. Similarly, we reach a 94% TP rate and a 6×10^{-3} FP rate on 15 minutes of captured Wire traffic.

We also study the use of standard traffic analysis countermeasures against our attacks. In particular, we investigate tunneling SIM traffic through VPNs, mixing it with background traffic, adding cover IM traffic, and delaying IM packets. As expected, our experiments show that such countermeasures reduce the effectiveness of the attacks at the cost of additional communication overhead as well as increased latency for SIM communications. For instance, we find that tunneling Telegram traffic through VPN and mixing it with background web-browsing traffic reduces the accuracy of our attack from 93% to 70%, and adding cover traffic with a 17% overhead drops the accuracy to 62%. We argue that since many SIM users do not deploy such third-party countermeasures due to usability reasons, SIM providers should integrate standard traffic obfuscation techniques into their software to protect their users against the introduced traffic analysis attacks. In the meantime, *we have designed and deployed an open-source, publicly available countermeasure system, called IMProxy, that can be used by IM clients with no need to any support from IM providers*. We have demonstrated the effectiveness of IMProxy through simulations and experiments.

In summary, we make the following contributions:

- We introduce traffic analysis attacks that reliably identify users involved in sensitive communications through secure IM services. To launch our attacks, the adversary does not need to cooperate with IM providers, nor does she need to leverage any security flaws of the target IM services.
- We establish a statistical model for regular IM communications by analyzing IM traffic from a large number of real-world IM channels.
- We perform extensive experiments on the popular SIM services of Signal, Telegram, WhatsApp, Wickr, and Wire to demonstrate the in-the-wild effectiveness of our attacks.
- We study potential countermeasures against our attacks. In particular, we design and evaluate IMProxy, which is a proxy-based countermeasure system. IMProxy works for all major IM services, with no need for support from IM providers.

- Our code and other artifacts are available online.[1]

2 Background: Secure Instant Messaging (SIM) Applications

We define a **secure IM (SIM)** service to be an instant messaging service that satisfies two properties: (1) it deploys strong encryption on its user communications (either end-to-end or end-to-middle), and (2) it is not controlled or operated by an adversary, e.g., a government. While our attacks also apply to non-secure IM applications, an adversary can use other trivial techniques to compromise privacy of non-secure IM services. For instance, if the operator of an IM service fully cooperates with a surveillant government, e.g., the *WeChat* IM service in China, the IM provider can let the adversary identify target users with no need for traffic analysis mechanisms. Similarly, an IM service with weak encryption can be trivially eavesdropped with no need for sophisticated traffic analysis attacks. Table 1 overviews some of the most popular SIM services.

2.1 How SIM Services Operate

Architecture: All major IM services are *centralized*, as shown in Table 1. Therefore, all user communications in such services are exchanged through servers hosted by the IM provider companies, e.g., Telegram Messenger LLP (note that some less popular services use a peer-to-peer architecture, e.g., FireChat [92], Ring [78], and Briar [17]). Each IM service has a server for authentication and key exchange. A database server stores message contents and other user information (possibly encrypted with client keys). Some IMs use Content Delivery Networks (CDNs) to run their databases to improve quality of service and resist attacks. Existing IM services use various messaging protocols for user communications, including Signal [32], Matrix [10], MTProto [60], and Off-the-Record [16].

Popular IM services intermediate all user communications by having user traffic go through their servers. Such a centralized architecture allows IM providers to offer high quality of services and solves critical issues like reaching to offline clients and clients behind NAT/firewalls. However, this presents different privacy threats to the users, as IM servers are involved in all user communications. Some IM services deploy end-to-end encryption to alleviate this, as presented below.

Security Features: IM services use standard authentication mechanisms like authorization keys and public key certificates to *authenticate* IM servers and peers [13], [14]. Also, they use standard techniques to ensure the *integrity* of messages. All major IM services encrypt user communications to protect *confidentiality* [35]. Some IM providers additionally deploy *end-to-end encryption* on user communications. This prevents IM operators from seeing the content of communications; however, they can still see communication meta-data, e.g., who is talking to whom and when. WhatsApp, Skype, Line, as well as Telegram and Facebook Messenger offer end-to-end encryption, while WeChat, Snapchat, and the BlackBerry Messenger do not. Please refer to Johansen et al. [46] for further discussion of other IM security features.

1. https://github.com/SPIN-UMass/IMProxy.

2.2 Prior Security Studies of IM Services

Metadata leakage: Coull and Dyer [25] are the first to apply traffic analysis on messaging applications. They demonstrate traffic analysis attacks that can infer various meta-data of a target Apple iMessage user, specifically, the operating system version, type of the IM action, and, to some degree, the language of conversations. More recently, Park and Kim [73] perform traffic analysis on the Korean KakaoTalk IM service, to identify users' online activities using basic classification algorithms. Afzal et al. [4] identify the activities associated with the Signal app such as receiving or initiating calls, typing patterns, and media messages by analyzing a user's traffic patterns. Our work differs from these works in that the design of our detectors rely on theoretical foundations and meticulous modeling of IM communications. Also, we believe that our attacks are able to reveal IM meta-data that is more sensitive than what was identified by prior works. We demonstrate the applicability of our attacks on several IM services, and design and evaluate tailored countermeasures.

Security vulnerabilities: Johansen et al. [46] surveyed different implementations of SIM protocols such as Signal, WhatsApp, and Threema, and evaluated their security and usability; they conclude that none of the studied applications are infallible. Unger et al. [100] performed a comprehensive study of instant messaging protocols focused on their security properties around trust establishment, conversation security, and transport privacy. Also, Aggarwal et al. [5] study the implementation of encryption in widely-used messaging applications.

Furthermore, there have been various identity enumeration attacks on messaging applications. In particular, as some IM services use SMS text message to activate new devices, an adversarial phone company can initiate and intercept such authorization codes to either identify users or access their accounts. Alternatively, unconfirmed reports [24] suggest that mainland Chinese and Hong Kong authorities may have attempted to discover Hong Kong protesters by misusing a Telegram feature that allowed one to discover the Telegram IDs of phone contacts (therefore, mapping phone numbers to their Telegram IDs); Telegram has promised to fix this issue through an update that will allow users to cloak their phone numbers [81].

Alternatively, Schliep et al. [82] evaluate the security of the Signal protocol against Signal servers. They identify vulnerabilities that allow the Signal server to learn the contents of attachments, re-order and drop messages, and add/drop participants from group conversations. Note that their study targets an entirely different adversary than ours, i.e., their adversary is a compromised/malicious Signal server, whereas in our case the adversary is any third-party who is able to wiretap encrypted IM traffic. Also, their attacks only work against Signal, whereas our attacks apply to all major IM services as they rely on fundamental communication behavior of IM services.

Communication privacy: The centralized nature of popular SIM services makes them susceptible to various privacy issues. First, all user communications, including group communications and one-on-one communications, are established with the help of the servers run by the SIM providers;

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

TABLE 1: Popular IM services [91]

IM Service	Monthly Users	Based in	Owned by	End-to-End Encryption	Centralized
WhatsApp	2000 M	United States	Facebook	✓	✓
Facebook Messenger	1300 M	United States	Facebook	✓(Secret Communications)	✓
WeChat	1251 M	China	Tencent	✗	✓
Telegram	550 M	UAE	Telegram Messenger LLP	✓(Secret Chats)	✓
Snapchat	538 M	United States	Snap Inc	✗	✓
Signal	40 M	United States	Open Whisper Systems	✓	✓
Wickr	<1 M	United States	Amazon	✓	✓
Wire	<1 M	European Union	Wire Swiss GmbH	✓	✓

therefore, SIM providers have access to the metadata of all communications, i.e., who is talking to whom, and channel ownership and membership relationships. Recent works suggest using various cryptographic techniques, such as private set intersection, to protect privacy against the central operators, e.g., for contact discovery [49], [58]. Second, even if an IM service provider is not malicious, its servers may be compromised by malicious adversaries [31] or subpoenaed by governments, therefore putting client communication metadata at risk.

In traditional SIM services, user communications are encrypted end-to-middle, i.e., between clients and SIM servers. In such services, the SIM providers can see not only the users' communication metadata but also their communication contents. Recently, major SIM providers such as WhatsApp have started to support end-to-end encryption, therefore protecting communication content from SIM providers [1]. Poor/buggy implementations of some SIM services have resulted in various security flaws and meta-data leakage threats despite their use of end-to-end encryption [33], [51], [65], [80], [113], e.g., through on/off notifications in Telegram [33] and the recent WhatsApp vulnerability giving remote access to the hackers [113].

Censorship: The centralized architecture of popular SIM services makes their censorship trivial: censors can easily blacklist a handful of IP addresses or DNS records to block all communications to a target SIM service. A straightforward countermeasure to unblock censored SIM services is to use standard circumvention systems like VPNs [101], Tor [28], and information-centric networks [59]. Alternatively, major SIM services allow the use of circumvention proxies to evade blocking, e.g., as built into the recent versions of the Telegram software after censorship attempts by Iranian and Russian authorities.

3 Attack and Threat Model

In this work, we demonstrate **a fundamental attack** on IM services: *our attacks are applicable to all major IM services, and are not due to buggy software implementations that can be fixed through software updates*, as overviewed in Section 2.2.

Our attacks are performed by an adversary who merely performs *traffic analysis*. In this setting, the attacker does not need to compromise or coerce the SIM provider, nor does she need to block the target IM service entirely. Instead, the adversary performs traffic analysis to identify the participants of target IM communications in order to either punish the identified IM participants or *selectively block* the target communications. In particular, the adversary can use traffic analysis to identify the administrators of controversial political or social IM channels and force them to shut down their channels (as seen in recent incidents [2], [3]). Alternatively, the adversary can use our traffic analysis attacks to identify the members of controversial IM channels, and thereby selectively disrupt the access to the target channels.

3.1 Introducing the Players

The **adversary** is a surveillance organization, e.g., an intelligence agency run by a government. The **goal of the adversary** is to *identify (the IP addresses of) the members or administrators (owners) of target IM communications*.

A *target IM communication* can be a public IM channel (e.g., a chat room) on politically or socially sensitive topics, or a private IM communication between target users, e.g., dissidents and journalists.

For the adversary to be able to conduct the attack, she needs to be *intercepting* the (encrypted) network traffic of the **monitored IM users**, e.g., by wiretapping the ISPs of the monitored users. Therefore, considering the Great Firewall of China as the adversary, it can only perform the attack on the IM users residing inside China.

3.2 Threat Model

We assume that the hosting IM service is a secure IM (SIM) service, as defined in Section 2. Therefore, the adversary does not leverage any security vulnerabilities of the target SIM service in performing the attack. For instance, the SIM system does not leak the IP addresses (or other sensitive meta-data) of its clients to the adversary. Also, we assume all traffic between IM clients and the IM servers to be *encrypted* with strong encryption. Finally, the operators of the SIM service *do not cooperate with the adversary* in identifying target members.

3.3 How the Attack Is Performed

Figure 1 illustrates the setup of the attack. Suppose that the adversary aims at identifying the participants of a specific IM communication, C.

Adversary's ground truth: For any target communication C, the attacker needs to obtain some *ground truth* about the content of the communication. This can be done in three ways:

(1) If C is an open (public) communication e.g., a public group or channel, the adversary joins C (as a member) and records the messages sent on C along with their metadata (e.g., time and size of the messages).

(2) The adversary has joined C and is capable of posting messages to C. This can happen if C a closed group that gives every member the ability to post messages, or this could be because the adversary has gained an admin role for C (e.g., the surveillance adversary has created a channel on a politically sensitive topic to identify target journalists, or the adversary has arrested the admin of a sensitive channel and is misusing her account). In this setting, not only the adversary can record the messages

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

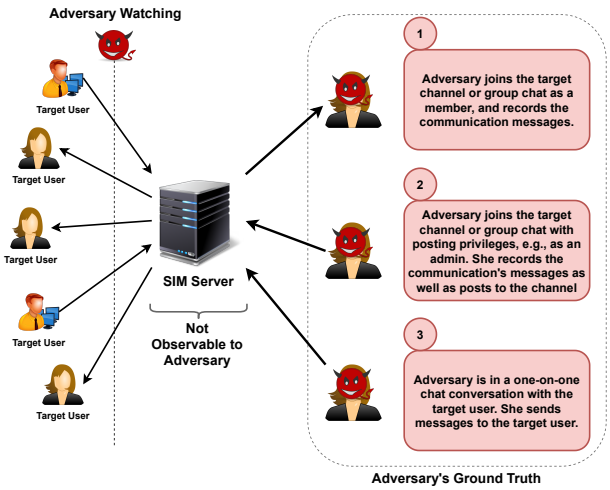


Fig. 1: Alternative attack scenario

posted to C, but also she can post her own messages to C with her desired (distinct) traffic patterns.

(3) The adversary is messaging the target user (e.g. an anonymous political account) in the one-on-one chat C, in which she can send her desired distinct traffic patterns.

Adversary's wiretap: The adversary monitors the (encrypted) network traffic of IM users to identify (the IP addresses of) the members/admins of the target IM communcation C. This can be performed by the adversary wiretapping the network traffic of the ISPs or IXPs he is controlling, e.g., by the Great Firewall of China. Alternatively, the adversary can wiretap the network traffic of specific individuals (e.g., suspected activists), perhaps after obtaining a wiretapping warrant.

Adversary makes decisions: The adversary uses a detection algorithm (as introduced in Section 5) to match the traffic patterns of the wiretapped users to the ground truth traffic patterns of the target communication C.

3.4 Related Traffic Analysis Attacks

Prior work has studied various kinds of traffic analysis attacks in different contexts.

Flow correlation In this setting, the adversary tries to link obfuscated network flows by correlating their traffic characteristics, i.e., packet timings and sizes [26], [29], [38], [53], [64], [85], [106], [117]. Flow correlation has particularly been studied as an attack on anonymity systems like Tor [9], [62], [76], [104], [118]: the adversary can link the ingress and egress segments of a Tor connection (say, observed by malicious Tor guard and exit relays) by correlating the traffic characteristics of the ingress and egress segments. Recently, Nasr et al. [63] introduce a deep learning based technique called DeepCorr which learns a correlation function to match Tor flows, and outperforms the previous statistical techniques in flow correlation.

Flow watermarking This is the active version of flow correlation attacks described above. In flow watermarking, the adversary encodes an imperceptible signal into traffic patterns by applying slight perturbations to traffic features, e.g., by delaying packets [40], [41], [75], [105], [116]. Compared to regular (passive) flow correlation techniques, flow watermarks offer higher resistance to noise, but require real-time modification of network traffic, and are subject to detection attacks.

Website fingerprinting In Website Fingerprinting (WF), the adversary intercepts network connections of some monitored users and tries to match the patterns of the intercepted connections to a set of target webpages. WF has particularly been studied as an attack on Tor. Existing WF techniques leverage various machine learning algorithms, such as k-NN, SVM, and deep neural networks to design classifiers that match monitored connections to target web pages [20], [36], [37], [39], [47], [55], [70], [71], [77], [102].

Intersection Attacks These attacks [6], [27], [29], [50] try to compromise anonymous communications by matching users' activity/inactivity time periods. For instance, Kesdogan et al. [50] model an anonymity system as an abstract threshold mix and propose the disclosure attack whose goal is to learn the potential recipients for any target sender.

Side channel attacks Another class of traffic analysis attacks aims at leaking sensitive information from encrypted network traffic of Internet services [8], [12], [15], [21], [34], [83], [94], [114]. For instance, Chang et al. [21] infer speech activity from encrypted Skype traffic, Chen et al. [22] demonstrate how online services leak sensitive client activities, and Schuster et al. [83] identify encrypted video streams.

Our Traffic Analysis Direction: Our attacks presented in this paper are *closest in nature to the scenario of flow correlation techniques*. Similar to the flow correlation setting, our adversary intercepts a live target flow (e.g., by joining a controversial IM channel), and tries to match it to the traffic patterns of flows monitored in other parts of the network (to be able to identify the IP addresses of the members or admins of the target channel). However, we can not trivially apply existing flow correlation techniques to the IM scenario, since the traffic models and communication noise are entirely different in the IM scenario. We, therefore, design flow correlation algorithms tailored to the specific scenario of IM applications. To do so, we first model traffic and noise behavior in IM services, based on which we design tailored flow correlation algorithms for our specific scenario.

Note that one could alternatively use techniques from the intersection attacks literature to design traffic analysis attacks for IM services. However, flow correlation is significantly more powerful than intersection attacks, as flow correlation leverages not just the online/offline behavior of the users, but also the patterns of their communications when they are online. Also, typical IM clients tend to remain online for very long time intervals. Therefore, we expect attacks based on intersection to be significantly less reliable (or require very long observations to achieve comparable reliability) when compared to our flow correlation-based attacks.

4 Characterizing IM Communications

We start by characterizing IM traffic and deriving a statistical model for it. We will use our model to design attack algorithms that are able to identify the participants of SIM communications.

4.1 Main IM Messages

IM services allow their users to send different types of messages; most commonly, text, image, video, file, and audio messages. IM messages are communicated between users through one of the following major communication forms:

- **Direct messages** are one-on-one communications between IM users. As mentioned earlier, popular IM services are centralized, therefore all direct messages are relayed through the servers of the IM providers, and unless end-to-end encryption is deployed, the servers can see communication contents.
- **Private (Closed) Group Communications** are communications that happen between multiple users. In groups, every member can post messages and read the messages posted by others. Each group has an administrator member who created the group and has the ability to manage the users and messages. An invitation is needed for a user to join a closed group.
- **Public (Open) Group Communications** which are also called *channels*, are a broadcast form of communication in which one or multiple administrators can post messages, and the members can only read or make limited reactions to these posts. Users can join public channels with no need for an invitation.

Note that some IM services offer other forms of communications, like status messages, that are not relevant to the attacks discussed in our work.

4.2 Data Collection

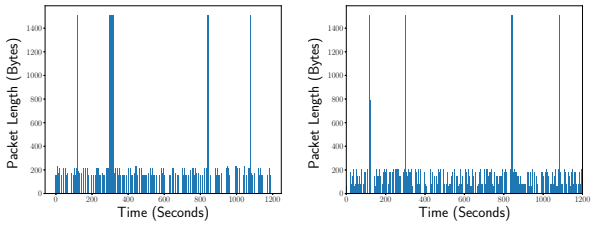
Since among the services studied, Telegram is the only one with publicly available communications in the form of public channels, we used it to collect the content of real-world public channels.

We use Telegram's API to collect the communications of 1,000 random channels with different message rates, each for a 24-hour span. For every collected Telegram message, we extract the channel ID it was sent over, its timestamp, the type of message (text, photo, video, audio or file), and the message size. Telegram has a limit of 50 on the number of new channels a user can join every day. Therefore, we use multiple Telegram accounts over several days to perform our data collection (also note that each Telegram account needs to be tied to an actual mobile phone number, limiting the number of accounts one can create). In Section 6.1, we describe how we use the communication patterns we collected for Telegram to generate traffic flows for the other SIMs.

4.3 Modeling IM Communications

We use Telegram's data to derive a model for IM traffic for two reasons; first, Telegram hosts a very large number of *public* channels that we can join to collect actual IM traffic. This is unlike other popular IM services where most group communications are closed/private. The second reason for choosing Telegram for data collection is that Telegram has been at the center of recent censorship and governmental surveillance attempts [2], [3], [96], [97], as it is home to a multitude of politically and socially sensitive channels.

Although we choose Telegram to obtain a statistical model for IM traffic, we show that our techniques perform similarly on other SIMs like WhatsApp and Signal. This is because these services implement limited traffic obfuscation, and therefore the shape of the traffic is similar across different IMs. We have illustrated this in Figure 2, where the same stream of messages are sent over two different SIM services, resulting in similar traffic patterns.



(a) Telegram (b) Signal

Fig. 2: Comparing the shape of traffic on two major SIM services; by sending the same sequence of IM messages, we observe similar traffic bursts regardless of the service provider.

TABLE 2: Distribution of various message types

Type	Count	Volume (MB)	Size range	Avg. size
Text	12539 (29.4%)	3.85 (0.016%)	1B-4095B	306.61B
Photo	20471 (48%)	1869.57 (0.765%)	2.40KB-378.68KB	91.33KB
Video	6564 (15.4%)	232955.19 (95.3%)	10.16KB-1.56GB	35.49MB
File	903 (2.1%)	47.46 (0.019%)	2.54KB-1.88MB	52.56KB
Audio	2161 (5.1%)	9587.36 (3.92%)	2.83KB-98.07MB	4.44MB

We model two key features of IM traffic: inter-message delays (IMDs) and message sizes. We also model the communication latency of IM traffic. We use *Maximum Likelihood Estimation (MLE)* [69] to fit the best probability distribution for each of these features.

Inter-Message Delays (IMDs): The IMD feature is the time delay between consecutive IM messages in an IM communication. In our model, we merge sent messages separated by less than a threshold, t_e seconds. We do this because extremely close messages create a combined traffic burst in the encrypted IM traffic that cannot be separated by the traffic analysis adversary. Such close messages can appear (infrequently) when an administrator forwards a batch of IM messages from another group. We also filter out the very long IMDs that can correspond to long late-night inactivity periods.

We show that the probability density function of IMDs can be closely fitted to an exponential distribution using our MLE algorithm; Figure 3 shows the probability density function of IMDs for 200 IM channels with a message rate of 130 messages per day. We interpret the exponential behavior of the IMDs to be due to the fact that messages (or message batches) are sent independently in the channels (note that this will be different for interactive one-on-one chats which will be discussed in Section 6.1).

Also, we consider IMDs to be independent of the type and size of messages, since in practice there is no correlation between the time a message is sent and its type or size.

Messages Sizes: Table 2 shows the size statistics and frequencies of the five main message types in our collected IM messages. We use these empirical statistics to create a five-state Markov chain, shown in Figure 6, to model the sizes of the messages sent in an IM communication stream.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

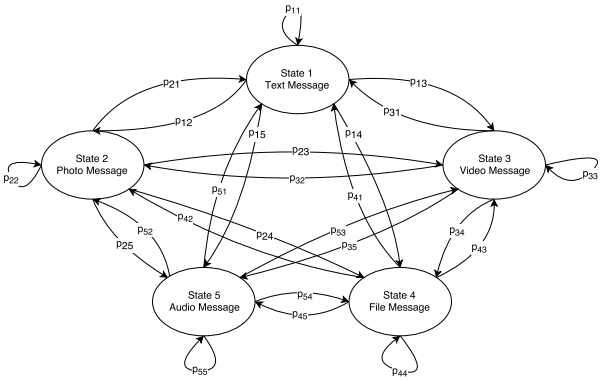


Fig. 6: Markov chain of IM message sizes

We obtain the empirical transition probability matrix of this Markov model for the aggregation of all channels, as well as for groups of channels with similar rates.

Finally, Figure 4 shows the Complementary Cumulative Density Function (CCDF) of the normalized message sizes for different message types (the sizes are normalized by the maximum message size of each category). We observe that different message types are characterized by different message size distributions.

Communication Latency: IM messages are delayed in transit due to two reasons: network latency and the IM servers' processing latency. To measure such latencies, we collect IM traffic from 500 channels, each for one hour (therefore, 500 hours worth of IM traffic) using Telegram's API. We then set up two IM clients, and send the collected IM traffic between the two clients to measure the incurred communication latencies. Using MLE, we find that transition latencies fit best to a Laplacian distribution $f_{\mu,b}(x)$, where μ is the average and $2b^2$ is the variance of the delay. Since network delay cannot be negative, we consider only the positive parts of the Laplace distribution. Figure 5 shows a Quantile-Quantile (Q-Q) plot of the packet latencies against the best Laplace distribution.

5 Details of Attack Algorithms

We design two algorithms for performing our attack (i.e., to map monitored IM users to their communication). As discussed in Section 3.4, our attack scenario is closest in nature to the scenario of flow correlation attacks. Therefore, the design of our attacks is inspired by existing work on flow correlation. Prior flow correlation techniques use standard statistical metrics, such as mutual information [23], [119], Pearson correlation [52], [85], Cosine Similarity [42], [64], and the Spearman Correlation [93], to link network flows by correlating their vectors of packet timing and sizes. We use *hypothesis testing* [74],[2] similar to state-of-the-art flow correlation works [41], [42], we design optimal traffic analysis algorithms for the particular setting of IM communications. In contrast to flow correlation studies which use the features of network packets, we use the features (timing and sizes) of *IM messages* for detection.

Note that the recent work of DeepCorr [63] uses a deep learning classifier to perform flow correlation attacks on Tor. They demonstrate that their deep learning classifier outperforms statistical correlation techniques in linking Tor connections. In Section 6.5, we compare our statistical classifiers with a DeepCorr-based classifier tailored to IM traffic. As we will show, when a SIM service has not deployed strong traffic obfuscation, our statistical classifiers *outperform* such deep learning based classifiers, especially for shorter flow observations. Intuitively, this is due to the sparsity of events in typical IM communications, as well as the stationary nature of noise in IM communications in contrast to the scenario of Tor. Note that this fully complies with Nasr et al. [63]'s observation that DeepCorr only outperforms statistical classifiers in non-stationary noisy conditions, where statistical traffic models become inaccurate.

Our hypothesis testing: Consider C to be a target SIM communication (e.g., a public group on a politically sensitive topic). For each IM user, U, the attacker aims at deciding which of the following hypotheses is true:

- H_0: User U is *not* associated with the target communication C, i.e., she is participant of communication C.
- H_1: User U *is* associated with the target communication C, i.e., she is posting messages to that communication as an admin, or is a member of that communication and therefore receives the communication's messages.

2. Our approach is "threshold testing" by some of the more strict definitions, however, we will use the term "hypothesis testing" in this paper as threshold testing falls into the general class of statistical hypothesis tests [74].

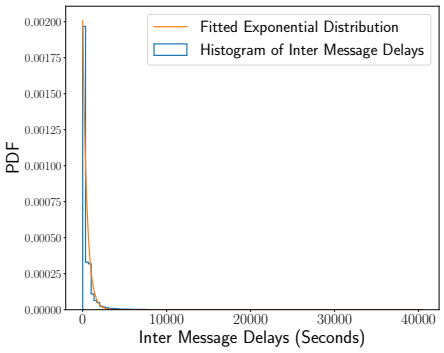


Fig. 3: The PDF of inter-message delays and its fitted exponential distribution

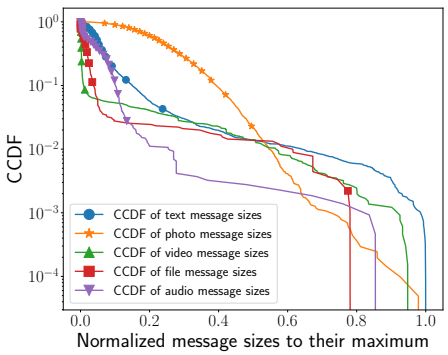


Fig. 4: Complementary CDF (CCDF) of IM Size distributions for different types of messages

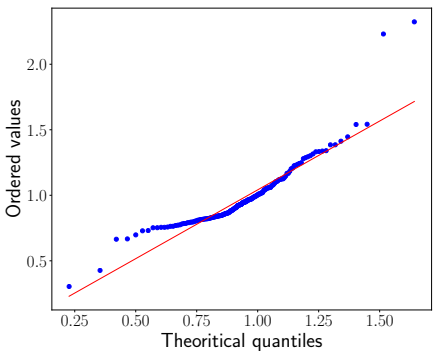


Fig. 5: The Quantile-Quantile plot of transition delay and its fitted Laplacian distribution

As described in our threat model (Section 3), the adversary can only observe encrypted SIM communications between users and SIM servers. Therefore, we design detectors that use traffic features, i.e., IMDs and message sizes. In the following, we describe two detector algorithms.

5.1 Event-Based Detector

Our first detector, the *Event-Based Detector*, aims at matching SIM events in a target user's traffic to those of the target communication C. An *event* $e = (t, s)$ is a single SIM message or a batch of SIM messages sent with IMDs less than a threshold t_e (as introduced in Section 4.3). Each single SIM message can be one of the five types of image, video, file, text, or audio. t is the time that e appeared on the SIM communication (e.g., sent to the public channel), and s is the size of e. Note that an SIM communication can include SIM protocol messages as well (handshakes, notifications, updates, etc.); however, such messages are comparatively very small as shown in Figure 7, and thus the detector ignores them in the correlation process. It also ignores messages with type text since they usually only generate not more than a couple of MTU-sized packets making them hard to detect. Recall that the adversary is not able to see plaintext events in the user's traffic due to encryption. Therefore, the first stage of our event-based detector is to *extract* events based on the user's encrypted SIM traffic shape. Figure 8 depicts the components of our event-based detector.

Event Extraction: Each SIM event, e.g., a sent image, produces a burst of MTU-sized packets in the encrypted traffic, i.e., packets with very small inter-packet delays. This is illustrated in Figure 7: SIM events such as images appear as traffic bursts, and scattered packets of small size are SIM protocol messages like notifications, handshakes, updates, etc. Therefore, the adversary can extract SIM events by looking for bursts of MTU-sized packets, even though she cannot see packet contents due to encryption. We use the IMD threshold t_e to identify bursts. Any two packets with distance less than t_e are considered to be part of the same burst. Note that t_e is a hyper-parameter of our model and we discuss its choice in Section 4.3. For each burst, the adversary extracts a SIM event, where the arrival time of the last packet in the burst gives the arrival time of the event, and the sum of all packet sizes in the burst gives the size of the event. Two SIM messages sent with an IMD less than t_e are extracted as one event. Similarly, the adversary combines events closer than t_e when capturing them from the target communication.

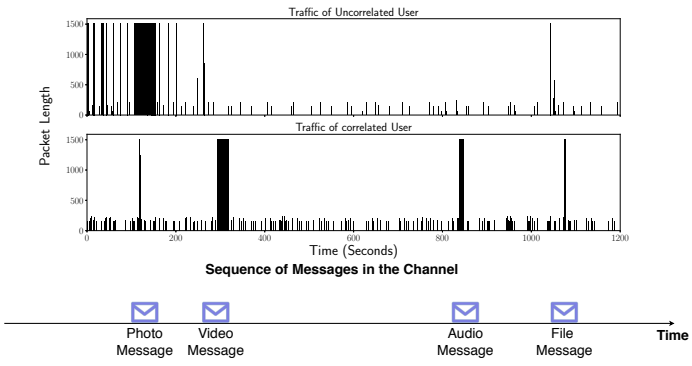


Fig. 7: Event extraction: IM Messages sent/received by a target user create bursts of (encrypted) packets; the adversary can extract events from packet bursts.

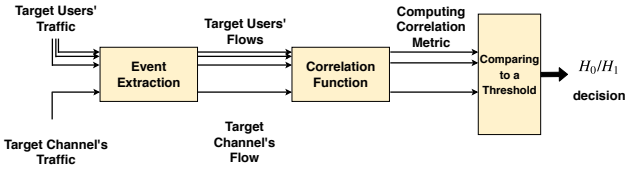


Fig. 8: Event-based detector

Forming Hypotheses: We call a one-sided SIM communication an SIM *flow*. Therefore, a flow either consists of the packets *sent* by a user to an SIM server, or the packets *received* by the user from the SIM server. We represent a flow with n events as $f = \{e_1, e_2, \ldots, e_n\}$, where $e_i = (t_i, s_i)$ is the ith event.

Consider a user U and a target communication C. Suppose that the adversary has extracted flow $f^{(U)} = \{e_1^{(U)}, e_2^{(U)}, \ldots, e_n^{(U)}\}$ for user U (through wiretapping), and flow $f^{(C)} = \{e_1^{(C)}, e_2^{(C)}, \ldots, e_n^{(C)}\}$ for the target communication C (using her ground truth). The detector aims at deciding whether user U is participant of C. We can re-state the adversary's hypotheses presented earlier in this section as follows:

- H_0: User U is not a participant of the target communication; hence, $f^{(C)}$ and $f^{(U)}$ are independent.
- H_1: User U is a participant of the target communication C; therefore, the user flow $f^{(U)}$ is a noisy version of the communication flow $f^{(C)}$.

Therefore, we have

$$\begin{cases} H_0 : t_i^{(C)} = t_i^{(*)} + d_i^{(*)}, s_i^{(C)} = s_i^{(*)}, 1 \le i \le n \\ H_1 : t_i^{(C)} = t_i^{(U)} + d_i^{(U)}, s_i^{(C)} = s_i^{(U)}, 1 \le i \le n \end{cases}$$

where $f^{(*)} = \{e_1^{(*)}, e_2^{(*)}, \ldots, e_n^{(*)}\}$ is the flow of a user $U' \neq U$ who is *not* a participant of communication C. Also, $d_i^{(\cdot)}$ is the latency applied to the timing of the ith event. Note that IM message sizes do not change drastically in transit, and the order of messages remains the same after transmission.

Detection Algorithm: The adversary counts the number of event matches between the user flow $f^{(U)}$ and the communication flow $f^{(C)}$. We say that the ith communication event $e_i^{(C)}$ matches *some* event $e_j^{(U)}$ in $f^{(U)}$ if:

- $e_i^{(C)}$ and $e_j^{(U)}$ have close timing: $|t_i^{(C)} - t_j^{(U)}| < \Delta$; and
- $e_i^{(C)}$ and $e_j^{(U)}$ have close sizes: $|s_i^{(C)} - s_j^{(U)}| < \Gamma \times s_i^{(C)}$.

where Δ and Γ are thresholds values for the timing and sizes of events. Note that even though the sizes of SIM messages do not change in transmission, the event extraction algorithm introduced earlier may impose size modifications, as network jitter is able to divide/merge event bursts (i.e., a burst can be divided into two bursts due to network jitter or two bursts can be combined due to the small bandwidth of the user). It should also be noted that defining closeness of event sizes as a ratio of the size of the event rather than using a fixed-size threshold improves the performance of the detector.

Finally, the adversary calculates the ratio of the matched events within a flow as $r = k/n$, where k is number of

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

TABLE 3: The empirical value of p_1 measured for different client bandwidths

Client Bandwidth (Mbps)	p_1
0.1	0.824
0.5	0.902
1	0.921
10	0.974
100	0.983

matched events and n is the total number of events in the flow of target communication. The detector decides the hypothesis by comparing to a threshold: $r = \frac{k}{n} \underset{H_0}{\overset{H_1}{\gtrless}} \eta$ where η is the detection threshold.

Analytical Bounds: We first derive an upper-bound on the probability of false positive ($\mathbb{P}_{\text{FP}}$), i.e., the probability that H_1 is detected when H_0 is true (Type I error). Let p_0 be the probability that a message with size $s_i^{(C)}$ and time $t_i^{(C)}$ matches an event in $f^{(U)}$ when H_0 is true, i.e., there exists only one message whose time $t_j^{(*)}$ satisfies $t_i^{(C)} \leq t_j^{(*)} \leq t_i^{(C)} + \Delta$ and has the same size label as $s_i^{(C)}$. From our observations, $p_0 = 0.002$. This Type I error occurs if more than $\eta \cdot n$ events in $f^{(C)}$ match $f^{(U)}$, when H_0 is true. This is equivalent to the case that less than $n - \eta \cdot n$ events in $f^{(C)}$ do not match $f^{(U)}$ when H_0 is true. Consequently,

$$\begin{aligned}\mathbb{P}_{\text{FP}} = \mathbb{P}(k \geq \eta n \mid H_0) &= \mathbb{P}(n-k \leq n - \eta n \mid H_0), \\ &= F(n - \eta n; n, 1-p_0), \\ &\leq \left(\frac{1-\eta}{p_0}\right)^{-n+n\eta\eta} \left(\frac{\eta}{1-p_0}\right)^{-n\eta},\end{aligned} \tag{1}$$

where $F(r; m, p) = \mathbb{P}(X \leq r)$ is the cumulative density function of a Binomial distribution with parameters m, p, and the last step follows from the following inequality which is tight when p is close to zero [7]:

$$F(r; m, p) \leq \left(\frac{r/p}{p}\right)^{-k} \left(\frac{1 - r/m}{1-p}\right)^{k-m} \tag{2}$$

Next, we upper-bound the probability of false negatives ($\mathbb{P}_{FN}$), i.e., the probability that H_0 is detected when H_1 is true, which occurs when less than k messages of $f^{(C)}$ match $f^{(U)}$. Let p_1 be the probability of the case that an event of $f^{(C)}$ matches $f^{(U)}$ when H_1 is true (Type II error).

Even though we mentioned earlier in this section that when H_1 is true, a delayed version of each event of $f^{(U)}$ appears in $f^{(C)}$, the bandwidth of the target user can affect the burst extraction process. As explained earlier in this section, we merge bursts of packets for messages whose IMD is less than t_e. Hence, suppose that the time it takes for the user to send a message is large enough to make the IMD between the current message and the next one less than t_e. Therefore, these two consecutive messages are combined in one burst. Table 3 shows the value of p_1 observed from our data for different bandwidths. Since the bandwidth of our experiments is 1Mbps, $p_1 = 0.921$.

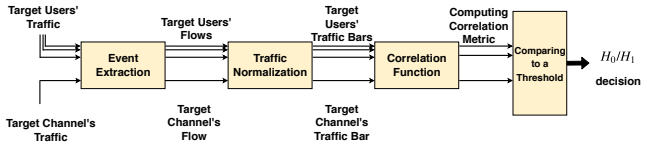


Fig. 9: Shape-based detector

Note that Type II error occurs when less than $\eta \cdot n$ messages of $f^{(C)}$ match $f^{(U)}$ when H_1 is true. Therefore,

$$\begin{aligned}\mathbb{P}_{\text{FN}} = \mathbb{P}\left(k \leq \eta n | H_1\right) &= F(\eta n; n, p_1) \\ &\leq \left(\frac{\eta}{p_1}\right)^{-n\eta} \left(\frac{1-\eta}{1-p_1}\right)^{\eta n - n},\end{aligned} \tag{3}$$

where the last step follows from (2).

5.2 Shape-Based Detector

We design a second detector called the *shape-based* detector. This detector links users to SIM communications by correlating the shape of their network traffic, where traffic shape refers to the vector of packet lengths over time. Figure 9 illustrates the four stages of the shape-based detector.

Event Extraction: The first stage of the shape-based detector is to extract SIM events from network traffic, which is performed similar to what was described earlier for the event-based detector. As described in the following, we do this in a way that accounts for the different bandwidths of the users being correlated.

Normalizing Traffic Shapes: The shape-based detector converts the extracted events into normalized traffic shapes by replacing each event with a traffic bar. The reason for doing so is that the shape of an IM event (e.g., the corresponding packet burst) is a function of user network bandwidths; our traffic normalization removes the impact of user bandwidth, and therefore the adversary can correlate traffic shapes with no knowledge of the underlying users' bandwidths.

To perform this normalization, we replace each event (i.e., each burst) with a traffic bar whose width is $2 \times t_e$, where t_e is the threshold used during event extraction as discussed in section 5.1. We choose this value to reduce the chances of overlaps between consecutive bars. To capture the sizes of events in traffic normalization, the height of each bar is chosen such that the area under the bar is equal to the size of the event. Our shape normalization also reduces correlation noise by removing small traffic packets that are not part of any SIM events.

To form the new normalized shape of traffic, we divide each bar into smaller bins of width t_s, the value of which is discussed in Section 6.1, and with a height equal to the height of the corresponding bar. Therefore each bar consists of a number of bins of equal width and height. Furthermore, we put bins with the same width t_s and height 0 between these bars. By doing so, after the traffic normalization, the new shape of traffic will be a vector of heights of bins over time.

Correlating Normalized Traffic Shapes: Our shape-based detector correlates the normalized shapes of two traffic streams of target communication C and user U to decide if they are associated. Suppose that $b^{(C)} =$

$\{b_1^{(C)}, b_2^{(C)}, \ldots, b_{n_C}^{(C)}\}$ and $b^{(U)} = \{b_1^{(U)}, b_2^{(U)}, \ldots, b_{n_U}^{(U)}\}$ are the respective vectors of heights of bins associated with the target communication and user being tested, where n_C and n_U are the number of events in target communication and user flows, respectively. We use the following normalized correlation metric:

$$corr = 2 \times \frac{\sum_{i=1}^{n} b_i^{(C)} b_i^{(U)}}{\sum_{i=1}^{n} (b_i^{(C)})^2 + \sum_{i=1}^{n} (b_i^{(U)})^2} \quad (4)$$

where $n = \min(n_C, n_U)$. Note that $corr$ returns a value between 0 and 1, which shows the similarity of the two traffic shapes (1 shows the highest similarity). Finally, the detector makes its decision by comparing $corr$ to a threshold, $corr \underset{H_0}{\overset{H_1}{\gtrless}} \eta$, where η is the detection threshold.

6 Attack Experiments

6.1 General Setup

We design our experimental setup to perform our attacks in the setting of Figure 1, and based on the threat model of Section 3. We use two SIM clients using different SIM accounts (e.g., Telegram accounts) that are running IM software on two separate machines. One of these IM clients is run by the adversary, and the other one represents the target client.

For Telegram, due to the large number of public channels available, we use the first type of ground truth in Figure 1 (adversary joins the target channel as a reading-only member). The adversary client joins target channel C, (e.g., a public political Telegram channel) and records the metadata of all the SIM communications of C, i.e., the timing and sizes of all messages sent on that channel. The target client may or may not be a member/admin of the target channel C.

Other than Telegram, the other SIM services do not have public communications (group chats or channels). For those SIM services, we focus on one-on-one communication and use the third type of ground truth in Figure 1 where adversary sends messages to the target in a one-on-one chat. In this scenario, the adversary chooses the content of the messages she sends to the target client, including the timing and sizes. She can also record the timing and sizes of the messages she receives from the target client.

Unlike public communications (e.g. channels in Telegram), one-on-one communications in these SIM services are private. Therefore, we had to generate the content of communications, and have the adversary's client send them to her target. To generate the content of the messages, we generate random content that matches the size and frequency statistics of the five main message types collected from Telegram, shown in Table 2. As stated in [54], to generate the IMDs, we used the Pareto Type I distribution with scale and shape parameters of 5000 millisecond and 0.93 respectively. The result was a total of 267 hours of one-on-one communication traces which included a total of 25367 messages, 17948 of which are media (non-text) messages. When sending the messages, we record the timestamp and at the same time, we capture the network traffic of the target's machine using tcpdump. It is not necessary to send messages in the other direction (from target's client to the adversary) since as a result of symmetry, we could assume that message was sent from the adversary to the target and it would have the same timestamp, size, and network traffic pattern.

Generating Traffic: With all of the studied SIM services, the adversary is not able to see the contents of the target client's communications by intercepting her traffic (due to encryption), however she can capture the encrypted traffic of the target client. The adversary then uses the detection algorithms introduced in Section 5 to decide if the target user is associated with the target communication C. In a real-world setting, the adversary will possibly have multiple target communications, and will monitor a large number of suspected clients.

WhatsApp The two clients were running on separate virtual machines running Ubuntu 18.04. We use Selenium [84] to connect to WhatsApp's web application and send 100 hours of generated one-on-one traces. This results in a more realistic packet capture as the target's client would have to click to download media messages upon arrival while subsequent messages are arriving.

Signal The two clients run on separate virtual machines running Ubuntu 20.04. We use temporary phone numbers to create Signal accounts. Then on both machines, we use version 0.8.4.1 of signal-cli [88], an open source command line and dbus interface for the Signal messenger to write a Python program for the adversary's machine to send all of the generated one-on-one traces and a separate Python program on the target's machine to receive those messages. Our target's client starts downloading media messages while subsequent messages are arriving.

Wire Wire provides the following messaging solutions: Wire Personal, Wire for Free, Wire for Enterprise, and Wire for Governments. In this work we use Wire for Free to perform our experiments. In the rest of the paper we refer to Wire Personal as Wire. Wire offers two types of messaging: one-on-one and group communications. We use different email addresses to create Wire accounts for these experiments. We use Selenium to connect to Wire's web application [111] to automate sending all of the generated one-on-one traces. When sending photos on Wire's web application, instead of using Wire's option to send compressed photos, we send them as files. Because otherwise, the order by which the media is downloaded on the target's client would be different from the order of the arrived messages, which would be as a result of Wire automatically downloading photos (not files) as they arrive. At the time of the experiments, there was no option to turn off this function.

Wickr Based on different customer needs, Wickr has developed several secure messaging apps: Wickr Me, Wickr Pro, and Wickr Enterprise. In this work we use Wickr Pro to perform the experiments. Note that in the rest of the paper we refer to Wickr Pro as just Wickr. Wickr offers two types of group communications: regular group communication and secure room communication. A room is a private group. The main difference between group communication and a room is that in a room there is a subset of participants called the *administrators* who can add new participants to the room or remove any participants from the room. In a group communication, only participants may remove themselves. We use the Wickr IO Integration Gateway to automate the

process of the experiments. We use the Wickr IO Docker Container [109] to set up the Web Interface REST API integration enabling the communication with Wickr client using Python.

Parameter Selection. We choose burst detection threshold as $t_e = 0.5s$ based on the empirical distribution of network jitter. Also, we set t_s of the shape-based detector to $0.01s$, as it leaves enough separation between two consecutive IM messages. Note that the optimum values of Δ and Γ in the event-based algorithm are different for each SIM experiment.

Ethics. We performed our inference attacks only over public IM channels or one-to-one communications between our own accounts; therefore, we did not capture any private IM communications. Also, we performed our attacks only on our own IM clients, but no real-world IM clients. Therefore, our experiments did not compromise the privacy of any real-world IM members or admins.

Synchronization As the adversary's clock may be skewed across her vantage points, our adversary uses a simple sliding window to mitigate this: for the first 10 seconds of traffic, the adversary slides the two flows being compared with 0.5 second steps, and uses the maximum correlation value.

6.2 Experiments in Normal Network Conditions

We experiment our attacks for each SIM (fully complying with the ethical considerations of Section 6.1).

Event-Based Detector Figure 11 shows the ROC curve of the event-based algorithm using Signal traffic data for 4 different observation lengths. We can see that, as expected, *longer traffic observations improve the accuracy of the detector*. For instance, the event-based detector offers a TP $= 0.58$ and FP $= 5.4 \times 10^{-3}$ with 3 min observation, while 30 mins of observation increases the TP to close to TP $= 0.98$. To calculate FP, we pair each flow of messages with traces of every other flow and feed it to the algorithm to see if it detects any matches. In practice, *an adversary can deploy the attack with hierarchical observation intervals to optimize accuracy and computation*. For instance, the adversary can monitor a mass of IM users for 15 mins of observation; then the adversary will monitor only the clients detected with 3 mins observations for longer time periods, e.g., 30 mins, to improve the overall FP performance while keeping computations low.

Furthermore, Figure 12 compares the performance of the event-based detector on our target SIMs. As can be seen, with Signal and Wire, the detector has a worse performance compared to other SIMs. It appears that they apply obfuscation algorithms on their traffic flows. However, we could not find any official documentation about their obfuscation techniques.

Shape-Based Detector We also experiment our shape-based detector on each SIM. Figure 13 compares the performance of the event-based and shape-based detectors on Wickr traffic for 3 mins and 15 mins of observed traffic.

As can be seen, *the shape-based detector outperforms the event-based detector for smaller values of false positive rates*. For instance, for a target true positive rate of 0.8, the shape-based detector offers a false positive of 1.5×10^{-3} compared to 4×10^{-3} of the event-based detector (with 15 mins of observation). The reason for this performance gap is the impact of event extraction noise on the event-based detector. Such noise has smaller impact on the shape-based detector as it correlates the shape of traffic flows. For higher false positive rates, the performance is opposite and the event-based detector has more true positive rate than the shape-based detector.

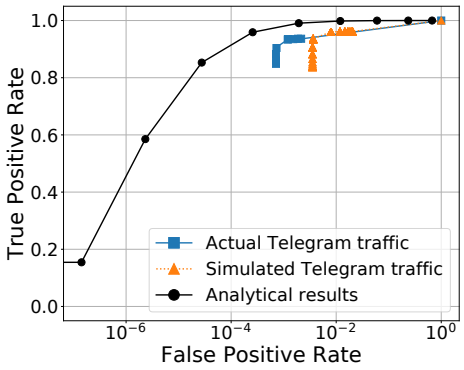


Fig. 10: Comparing the analytical upper bounds of the event-based detector with empirical results (for 15 minutes of Telegram traffic).

TABLE 4: The attack performances for one correlation.

Method	# of CPU cycles	One correlation time
Shape-based	34 431 709	34431.70 ms
Event-based	123 962	123.96 ms

Note that for our event-based detector in Figure 13, for short traffic observations (e.g., 3 mins) we cannot observe small FPs in our ROC curve. This is because the event-based correlation uses the number of matched events, which is very coarse-grained due to the limited number of events in short (e.g., 3 minutes) intervals. We use our analytical upper-bounds (derived in (1) and (3)) to estimate the performance trend for smaller false positive values for Telegram traffic in Figure 10.

In terms of the performance time, *our event-based detector is two orders of magnitude faster than the shape-based detector*. Table 4 compares the number of CPU cycles each of the two detectors take to calculate the correlation of a pair of 900 second Signal flows, as well as the correlation times of the two detectors. The main reason for this difference is that the event-based correlator uses the discrete time-series of event metadata for its correlation, while the shape-based detector uses traffic histograms over time.

6.3 Experiments in Poor Network Conditions

To evaluate the effect of the bandwidth of the target's device on the performance of the event-based detector, we collected traffic for each SIM with the target device's bandwidth limited to 1Mbps, 5Mbps, and in some cases 10Mbps. We used Wondershaper [112] to limit the bandwidth on target's VM. We then tuned the parameters of the event-based detector to better detect events of the traffic of the client with limited bandwidth.

Figure 14 shows the performance of the event-based detector on Signal and Wire when the bandwidth of the target user is limited to 10 Mbps and 5 Mbps compared

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

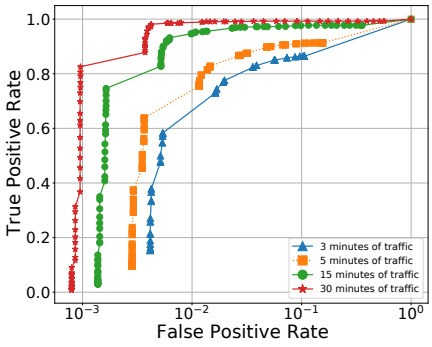


Fig. 11: The performance of the event-based detector on Signal traffic for different traffic lengths.

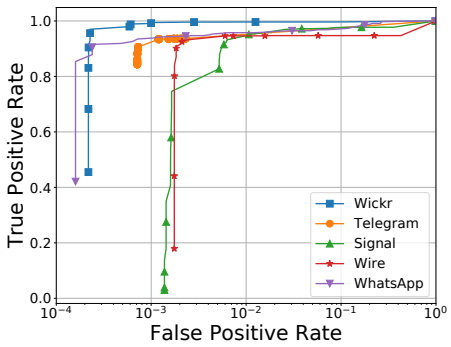


Fig. 12: Performance of the event-based detector on different SIMs (15 mins of observed traffic).

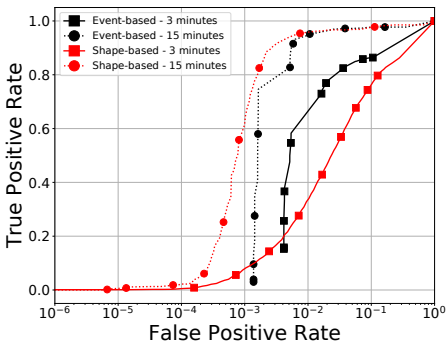


Fig. 13: Comparing event-based and shape-based detectors on Signal traffic.

to when there is no imposed limits. As expected, lower bandwidth corresponds with lower performance of the detector on both SIMs. With smaller bandwidths, there are more overlaps between events which causes more errors in the event extraction process. A similar pattern existed in our experiments for Wickr, Telegram, and WhatsApp. When we limit the bandwidth to 1Mbps, the performance drops significantly making the detector ineffective. This is expected as the the SIMs become almost unusable with a 1Mbps bandwidth when sending media messages.

6.4 Experiments to evaluate the effect of adversary's location

We also evaluate the performance of our event-based algorithm while tunneling adversary's traffic through VPNs in different locations. This is to evaluate the effect of adversary's location with respect to its target. We tunnel the traffic through VPNs in three locations: Japan, South Africa, and Turkey. In setups where VPN is used, either a TorGurad VPN client [99] or a NordVPN client [66] is installed on the sender (adversary) virtual machine and the VPN has been connected prior to sending messages. Figure 15 shows the performance of the event-based detector while observing 15 minutes of Signal's traffic as adversary's traffic is tunneled through a VPN server in different locations. We believe the poor performance of the event-based detector when traffic was tunneled through the VPN server in South Africa is due to the very low bandwidth of the connection through that VPN server (close to 1Mbps). As can be seen, tunneling the traffic through VPN affects the performance of the detector to some degree but does not make it ineffective as VPNs do not obfuscate the traffic patterns of SIMs. VPNs however, add a delay to adversary's traffic. This indicates that when directly sending messages to its target, the location of the adversary has some effect on the performance of the event-based detector. Figure 16 shows the performance of the event-based detector while observing the traffic of different SIM applications for 15 minutes as adversary's traffic is tunneled through a VPN server located in Japan. Comparing this figure with Figure 12 shows how the location of the adversary has some effect on the performance of the event-based algorithm while the algorithm still has its lowest performance on Signal.

6.5 Comparison with Deep Learning Techniques

As mentioned earlier in Section 3.4, the recent work of DeepCorr [63] uses deep learning classifiers to perform flow correlation attacks on Tor. They demonstrate that deep learning classifiers outperform statistical correlation techniques, like the ones we used in our work, in correlating Tor connections. In this section, we compare our IM classifiers with deep learning classifiers. As we show in the following, when the SIM service has not deployed effective obfuscation, *our statistical classifiers outperform deep-learning-based classifiers*, especially for shorter flow observations. Intuitively, this is due to the sparsity of events in typical IM communications, as well as the stationary nature of noise in unobfuscated IM communications in contrast to the scenario of Tor. Note that this fully complies with Nasr et al. [63]'s observation that DeepCorr only outperforms statistical classifiers in non-stationary noisy conditions, where statistical traffic models become inaccurate.

For fair comparisons, we obtain the original code of DeepCorr [63], and adjust it to the specific setting of IM traffic. Specifically, we divide the timing of each flow to equal periods of length 1 second, and in each period we assign values of $\{0, 1\}$ to that period. We set the value of a period 1 if there is a burst of packets in that period, and 0 if there is no burst of packets. As an example, if we use 15 minutes of traffic flows for correlation, our feature dimension is a 900-length vector with values of $0, 1$.

We design a DeepCorr model for each SIM using its collected data. Figures 17 and 18 show the ROC curves of our event-based detector compared with our deep-learning-based detector, using 3 and 15 minutes of Wickr and Signal traffic, respectively. As we can see, in case of Wickr, our event-based technique outperforms the deep-learning-based classifier for smaller false positive rates. For instance, for a false positive rate of 10^{-3} when using 15 minutes of traffic, our event-based detector achieves a 98% accuracy compared to 95% of the DeepCorr-based technique. We see that the performance advantage of our event-based detector significantly increases for shorter flow observations, e.g., when 3 minutes of traffic is used for detection, our classifier provides 93% accuracy compared to 62% of the DeepCorr-based classifier (for the a false positive rate of 10^{-3}).

On the other hand, for Signal, when 3 minutes of traffic is

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

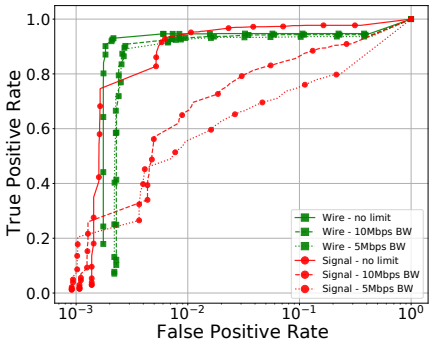


Fig. 14: Performance of the event-based detector on Signal and Wire SIMs with different bandwidth limits (15 mins of observed traffic).

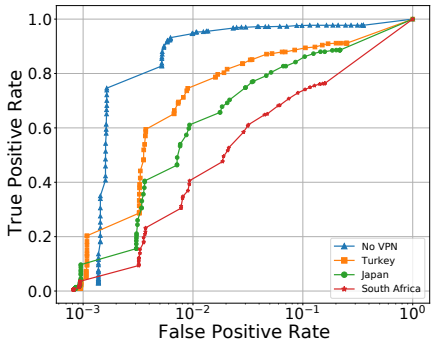


Fig. 15: Performance of the event-based detector on Signal when tunneling traffic through different VPN server locations (15 mins of observed Signal traffic).

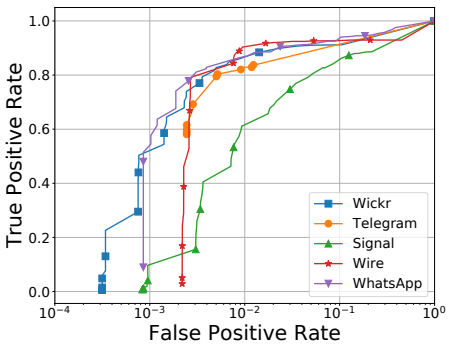


Fig. 16: Performance of the event-based detector on different SIMs when tunneling traffic through the VPN located in Japan (15 mins of observed traffic).

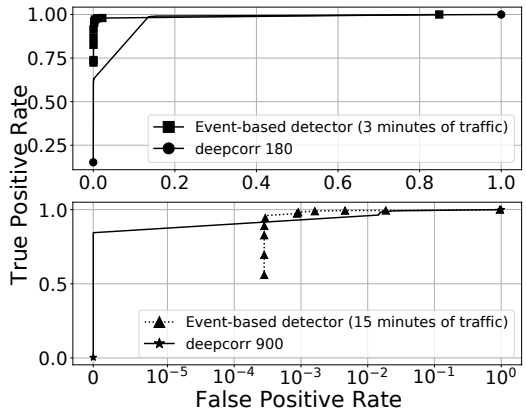


Fig. 17: Comparing our event-based detector with a DeepCorr-based classifier, for 3 and 15 mins of observed Wickr traffic.

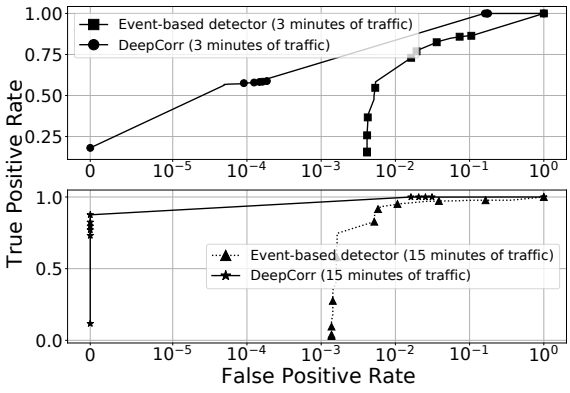


Fig. 18: Comparing our event-based detector with a DeepCorr-based classifier, for 3 and 15 mins of observed Signal traffic.

used for detection, DeepCorr provides a FP rate of 1.9×10^{-4} compared to 5.4×10^{-3} of our classifier (for the TP rates of 60% and 58% respectively). With 15 minutes of Signal traffic, DeepCorr provides a TP rate of 99% compared to 95% of our classifier (for a FP rate of 1.4×10^{-2}). In contrast with Wickr, on Signal, the deep-learning-based technique is performing better than the event-based detector. We believe this is because it can capture the noise in the traffic caused by the Signal obfuscation mechanisms as mentioned in Section 6.2.

Furthermore, we train a DeepCorr model on the aggregated data of all SIMs. We then test this aggregated model on the test data of each SIM. Figure 22 compares the performance of the aggregated model with individual models trained on Wickr, Wire, and Signal traffics. We see that the aggregated model has a similar performance compared to the models trained on each dataset separately.

Temporal Constraints: According to [63], DeepCorr learns the generic features of noise in Tor, regardless of the specific circuits and end-hosts during the training process. Therefore, there could be a need to re-train the DeepCorr model trained on SIM traffic if the generic features of noise of a SIM application change. Examples of such a change can be if a SIM starts to use a new encoding or a new compression algorithm for their text or media message. We believe this type of change to be infrequent.

7 Countermeasures

We deploy and evaluate possible countermeasures against our presented attacks. Intuitively, our attacks work because in-the-wild SIM services do not obfuscate traffic patterns enough. Therefore, we investigate various traffic obfuscation mechanisms as countermeasures against our traffic analysis-based attacks.

Note that obfuscation-based countermeasures have been studied against other kinds of traffic analysis attacks overviewed in Section 3.4. There are several key ideas used in existing countermeasures: (1) tunneling traffic through an overlay system that perturbs its patterns [57], [68], e.g., Tor, (2) adding background traffic (also called decoy) that is mixed with the target traffic [30], [56], [72], [103], [115], (3) padding traffic events (e.g., packets) [18], [19], [30], [48], [102], and (4) delaying traffic events [18], [19], [30], [102], [103]. In the following, we investigate various countermeasure techniques inspired by these standard approaches.

7.1 Tunneling Through Circumvention Systems With/Without Background Traffic

As the first countermeasure, we tunnel SIM traffic through standard circumvention systems, in particular VPN

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

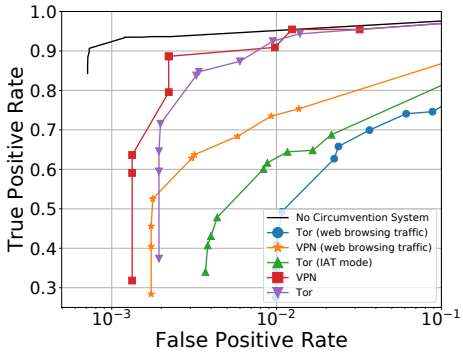


Fig. 19: The impact of various countermeasures on the performance of the event-based detector using different circumvention systems (15 minutes of observed Telegram traffic)

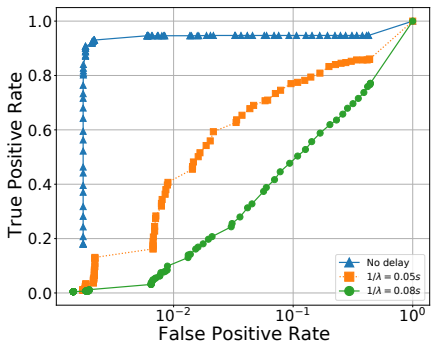


Fig. 20: Randomly delaying events by an SIM server acts as an effective countermeasure to our attacks. $\frac{1}{\lambda}$ is the mean of the added delay in seconds (15 minutes of observed Wire traffic)

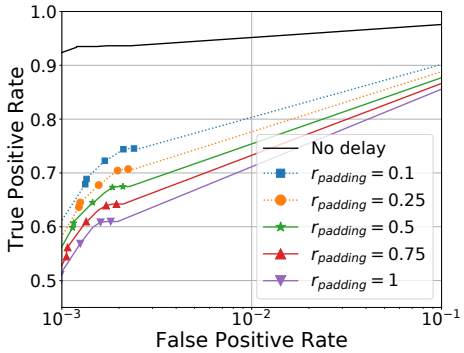


Fig. 21: Padding IM events by the SIM server (or client) can act as an effective countermeasure against our attacks. (15 minutes of observed Telegram traffic)

and Tor pluggable transports [98]. We use the same experimental setup as before and connect to 300 Telegram channels. For each circumvention system, we perform the experiments with and without any background traffic. In the experiments with background traffic, the VM running the SIM software also makes HTTP connections using Selenium. The background HTTP webpages are picked randomly from the top 50,000 Alexa websites. To amplify the impact of the background traffic, the time between every two consecutive HTTP GETs is taken from the empirical distribution of Telegram IMDs, therefore producing a noise pattern similar to actual SIM channels.

We observe that our event-based attack performs stronger against our countermeasures. Therefore, we only present the countermeasure results against the event-based detector. Figure 19 shows the ROC curve of the event-based detector using various circumvention systems and in different settings. Our Tor experiments are done once with regular Tor, and once using the obfs4 [68] transport with the IAT mode of 1, which obfuscates traffic patterns.

We see that *using regular Tor (with no additional obfuscation) as well as using VPN does not significantly counter our attacks*, e.g., we get a TP of 85% and a FP of 5×10^{-3} when tunneling through these services (using 15 mins of traffic). However, adding *background traffic* when tunneled through Tor and VPN reduces the accuracy of the attack, but *we get the best countermeasure performance using Tor's obfs4 obfuscator*.

Note that tunneling through a generic circumvention system like Tor is not the most attractive countermeasure to the users due to the poor connection performance of such systems.

7.2 IMProxy: An Obfuscation Proxy Designed for IM Services

We design a proxy-based obfuscation system, called IMProxy, built specifically for IM communications. IMProxy combines two obfuscation techniques: changing the timing of events (by adding delays), and changing the sizes of events through adding dummy traffic. An IM client has the ability to enable each of these countermeasures, and specify the amplitude of obfuscation to make her desired

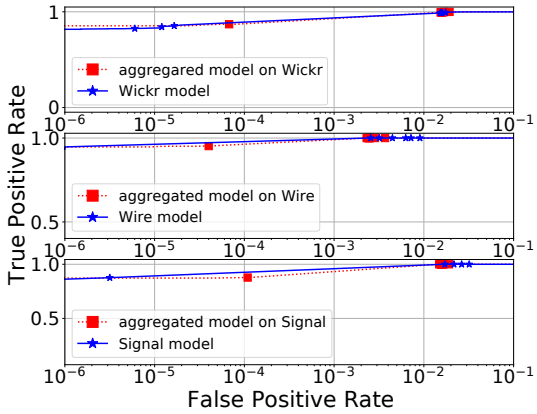


Fig. 22: Comparing the aggregated model trained on all the data with models trained on each SIM's dataset

tradeoff between performance and resilience. *IMProxy does not require any cooperation from IM providers*, and can be used to obfuscate *any IM service*.

Components of IMProxy: Figure 23 shows the design of IMProxy. For a client to use IMProxy, she needs to install a local proxy software. local proxy runs a SOCKS5 proxy listening on a local port. The client will need to change the setting of her IM software (e.g., Telegram software) to use this local port for proxying or use a proxy that can filter out IM software packets.

A second component of IMProxy is remote proxy, which is a SOCKS5 proxy residing outside of the surveillance area. The client needs to enter the (IP, port) information of this remote proxy in the settings of her local proxy software. Note that, in practice, remote proxy can be either run by the client herself (e.g., as an AWS instance), or can be run by the IM provider or trusted entities (similar to the MTProto proxies run for Telegram users [61]).

How IMProxy works: Once an IM client sets up her system to use IMProxy as above, her IM traffic to/from the IM servers will go through proxy servers of IMProxy, as shown in Figure 23. The IM traffic of the client will be handled by local proxy and remote proxy, which obfuscate traffic through padding and delaying.

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3218191

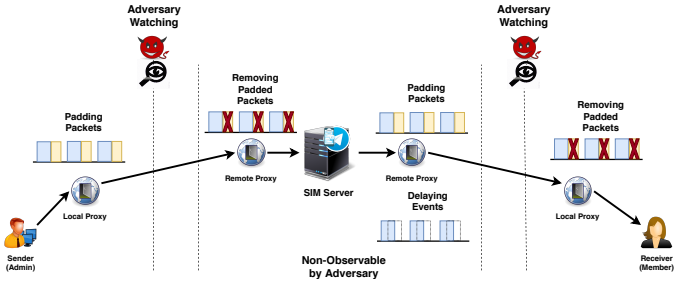


Fig. 23: Design of our IMProxy countermeasure.

As shown in the figure, IMProxy acts differently on upstream and downstream IM traffic. For upstream SIM communications (e.g., messages sent by an admin), local proxy adds padding to the traffic by injecting dummy packets and events at certain times. First, some dummy packets are injected close to the events in order to change their sizes. The size of padding for each event is chosen randomly, following a uniform distribution in [0, $r_{padding}$], where $r_{padding}$ is a parameter adjusted by each user. Second, some dummy events (burst of packets) are injected during the silence intervals; this is done randomly: during each 1 second silence interval, an event is injected with a probability $p_{padding}$, where $p_{padding}$ is also adjusted by each individual user. The size of dummy events is drawn from the empirical distribution of the sizes of image messages, as presented earlier. Finally, the dummy packets are removed by remote proxy before getting forwarded to the IM server. Note that all traffic between local proxy and remote proxy is encrypted so the adversary can not identify the dummy packets.

For downstream SIM communications (e.g., messages received by a member), remote proxy adds dummy packets, as above, which are then dropped by local proxy before being released to the client's IM software. In addition to padding, remote proxy delays the packets in the downstream traffic. In our implementation, remote proxy uses an Exponential Distribution with rate λ to generate random delays (which is based on our delay model in Figure 5). Note that no delay is applied on upstream traffic, as the delay will transit to the corresponding downstream traffic. Also, note that each client can control the intensity of padding by adjusting the $p_{padding}$ and $r_{padding}$ parameters, and control the amplitude of delays by adjusting λ.

To implement the delaying of packets, we used the *NetFilterQueue* and *scapy* modules in Python. To evaluate the effects of addition and removal of dummy packets and padding, we run a simulation using traffic collected without the use of IMProxy.

Evaluation against oblivious adversary: We first evaluate our IMProxy implementation against an adversary who is not aware of how IMProxy works (or its existence). To do so, we evaluate IMProxy against our event-based detector.

Figure 20 shows the ROC curve of the event-based detector for different values of λ. Note that $\frac{1}{\lambda}$ defines the average amount of delay added to the packets. As we can see, *increasing the added delay (by reducing λ) reduces the performance of our attack*, as it causes to missalign events across the monitored flows. For instance, a $\frac{1}{\lambda} = 0.05s$ reduces the adversary's TP from 90% to 2% (for a constant 18×10^{-4} false positive).

Figure 21 shows the ROC curves for the simulation of the event-based detector with different $r_{padding}$ and $p_{padding} = 10^{-4}$. Note that a $p_{padding} = 10^{-4}$ causes a 7% average traffic overhead. As expected, *increasing $r_{padding}$ reduces the performance of our attack*; even a $r_{padding}$ as small of 10% and 7% of dummy events can have a noticeable impact on countering the traffic analysis attacks, i.e., for a 10^{-3} false positive rate, the detection accuracy is reduced from 93% to 62%. Increasing $r_{padding}$ to 50% will further reduce detection accuracy to 56%.

Evaluation against IMProxy-aware adversary: Next, we evaluate IMProxy against an adversary who is aware that target users are deploying IMProxy and also knows the details of IMProxy. Our adversary trains a DeepCorr-based classifier on IM traffic obfuscated using IMProxy (note that our statistical detectors will suffer for such an adversary due to the randomness of IMProxy's obfuscation).

Figure 24 shows the performance of this DeepCorr-based classifier against IMProxy-obfuscated connections on Telegram (each flow is 15 mins). We use $r_{padding} = 0.1$ and evaluate the performance for different values of $p_{padding}$. As can be seen, *IMProxy is highly effective even against an IMProxy-aware classifier*, demonstrating IMProxy's efficiency in manipulating IM traffic patterns. For instance, for a false positive rate of 10^{-3}, the IMProxy-aware classifier provides true positive rates of 25% and 15% (for average obfuscation delays of 0.5 and 1), which is significantly weaker compared to 93% of the event-based detector when IMProxy is not deployed. As we can see, delaying provides better protection than padding; however, we expect that most users will prefer padding over delays due to the latency-sensitive nature of IM communications.

Note that each user tradeoffs between privacy protection and overhead by adjusting the countermeasure parameters. Ideally, the countermeasure software can ask the user her tolerable padding/delay overhead (or her target FP/FN for the adversary), and then will choose the best countermeasure parameters for the user. For instance, based on Figure 24, assuming that a real-world adversary can tolerate a FP of 10^{-3}, if the user states that she intends to keep the adversary's TP below 0.3, the countermeasure software delays packets with an average of 1s.

8 Conclusions

In this paper, we showed how popular IM applications leak sensitive information about their clients to adversaries who merely monitor encrypted traffic. Specifically, we devised traffic analysis attacks that enable an adversary to identify the administrators and members of target IM channels with practically high accuracies. We demonstrated the practicality of our attacks through extensive experiments on 5 real-world IM systems. We believe that our study presents a significant, real-world threat to the users of such services given the escalating attempts by oppressive governments in cracking down on social media.

We also investigated the use of standard countermeasures against our attacks and demonstrated their practical feasibility at the cost of communication overhead and increased IM latency. We designed and implemented an open-

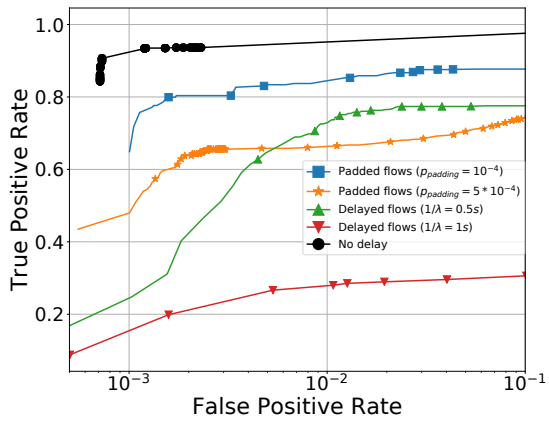


Fig. 24: Evaluating IMProxy against an IMProxy-aware classifier (trained using DeepCorr).

source, publicly available countermeasure system, IMProxy, which works for major IM services with no need to support from the IM providers. While IMProxy may be used as an ad hoc, short-term countermeasure by IM users, we believe that to achieve the best usability and user adoption, effective countermeasures should be deployed by IM providers (i.e., through integrating traffic obfuscation techniques into their software). We hope that our study will urge IM providers to take action.

ACKNOWLEDGMENTS

This work has been supported by the NSF grants 1953786 and 1564067, and by DARPA and NIWC under contract N66001-15-C-4067. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] "About end-to-end encryption," https://faq.whatsapp.com/791574747982248/, 2022.
[2] "Continued Arrest of Telegram Channels Admin in Orumiyeh," http://kurdistanhumanrights.net/en/continued-arrest-of-telegram-channels-admin-in-orumiyeh/, 2018.
[3] "Admins of 12 Reformist Telegram Channels Arrested in Iran Ahead of May 2017 Election," https://www.iranhumanrights.org/2017/03/12-reformist-telegram-channel-admins-arrested/, 2017.
[4] A. Afzal, M. Hussain, S. Saleem, M. K. Shahzad, A. T. S. Ho, and K.-H. Jung, "Encrypted network traffic analysis of secure instant messaging application: A case study of signal messenger app," *Applied Sciences*, vol. 11, no. 17, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/17/7789
[5] P. K. Aggarwal, P. Grover, and L. Ahuja, "Security Aspect in Instant Mobile Messaging Applications," in *RAETCS*, 2018.
[6] D. Agrawal, D. Kesdogan, and S. Penz, "Probabilistic treatment of MIXes to hamper traffic analysis," in *IEEE S&P*, 2003.
[7] R. Arratia and L. Gordon, "Tutorial on large deviations for the binomial distribution," *Bull. Math. Biol.*, 1989.
[8] J. Atkinson, M. Rio, J. Mitchell, and G. Matich, "Your WiFi Is Leaking: Ignoring Encryption, Using Histograms to Remotely Detect Skype Traffic," in *MILCOM*, 2014.
[9] A. Back, U. Moller, and A. Stiglic, "Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems," in *Information Hiding*, 2001.
[10] A. Balducci and J. Meredith, "Olm cryptogrpahic review." NCC Group PLC, Tech. Rep., 2016.
[11] "At least 60 percent of Iran Internet bandwidth usage accounts for Telegram," https://www.isna.ir/news/96062715757, 2017.
[12] D. Barradas, N. Santos, and L. Rodrigues, "Effective detection of multimedia protocol tunneling using machine learning," in *USENIX Security*, 2018.
[13] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication," in *Crypto*, 1999.
[14] J. Black and P. Rogaway, "CBC MACs for arbitrary-length messages: The three-key constructions," in *Crypto*, 2000.
[15] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: when randomness plays with you," in *SIGCOMM CCR*, 2007.
[16] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record Communication, or, Why Not to Use PGP," in *WPES*, 2004.
[17] "Secure P2P Messenger Releases First Version, Receives New Funding," https://briarproject.org/news/2018-1.0-released-new-funding.html, 2018.
[18] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense," in *WPES*, 2014.
[19] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses," in *CSS*, 2014.
[20] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *CCS*, 2012.
[21] Y.-C. Chang, K.-T. Chen, C.-C. Wu, and C.-L. Lei, "Inferring speech activity from encrypted Skype traffic," in *GLOBECOM*, 2008.
[22] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *IEEE S&P*, 2010.
[23] T. Chothia and A. Guha, "A statistical test for information leaks using continuous mutual information," in *CSF*, 2011.
[24] C. Cimpanu, "Hong Kong protesters warn of Telegram feature that can disclose their identities," https://www.zdnet.com/article/hong-kong-protesters-warn-of-telegram-feature-that-can-disclose-their-identities/, 2019.
[25] S. E. Coull and K. P. Dyer, "Traffic Analysis of Encrypted Messaging Services: Apple iMessage and Beyond," *SIGCOMM CCR*, 2014.
[26] G. Danezis, "The traffic analysis of continuous-time mixes," in *PETS*, 2004.
[27] G. Danezis and A. Serjantov, "Statistical disclosure or intersection attacks on anonymity systems," in *Information Hiding*, 2004.
[28] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-generation Onion Router," in *USENIX Security*, 2004.
[29] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay," in *RAID*, 2002.
[30] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail," in *IEEE S&P*, 2012.
[31] J. Engler and C. Marie, "Secure messaging for normal people," NCC Group, Tech. Rep., 2015. [Online]. Available: https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/secure-messaging-for-normal-people-whitepaper.pdf
[32] K. Ermoshina, F. Musiani, and H. Halpin, "End-to-End Encrypted Messaging Protocols: An Overview," in *INSCI*, 2016.
[33] O. Flisback, "Stalking anyone on Telegram," https://oflisback.github.io/telegram-stalking/, 2015.
[34] J. Gu, J. Wang, Z. Yu, and K. Shen, "Walls have ears: Traffic-based side-channel attack in video streaming," in *INFOCOM*, 2018.
[35] S. Harris, *CISSP All-in-One Exam Guide*, 6th ed. McGraw-Hill Osborne Media, 2012.
[36] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *USENIX Security*, 2016.
[37] G. He, M. Yang, X. Gu, J. Luo, and Y. Ma, "A novel active website fingerprinting attack against Tor anonymous system," in *CSCWD*, 2014.
[38] T. He and L. Tong, "Detecting Encrypted Stepping-Stone Connections," *TSP*, 2007.

[39] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naive-bayes classifier," in *CCSW*, 2009.

[40] A. Houmansadr and N. Borisov, "The need for flow fingerprints to link correlated network flows," in *PETS*, 2013.

[41] A. Houmansadr, N. Kiyavash, and N. Borisov, "RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows," in *NDSS*, 2009.

[42] ——, "Non-blind watermarking of network flows," *IEEE TON*, 2014.

[43] "Number of mobile phone messaging app users worldwide from 2016 to 2021," https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide, 2018.

[44] "Getting around Iran's Telegram ban," https://observers.france24.com/en/20180502-getting-around-iran'-telegram-ban-"i-installed-vpn-old-lady-next-door", 2018.

[45] " Help users in Iran reconnect to Signal," https://signal.org/blog/help-iran-reconnect/, 2021.

[46] C. Johansen, A. Mujaj, H. Arshad, and J. Noll, "Comparing Implementations of Secure Messaging Protocols (long version)," 2017.

[47] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *CCS*, 2014.

[48] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an Efficient Website Fingerprinting Defense," in *ESORICS*, 2016.

[49] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert, "Mobile private contact discovery at scale," in *USENIX Security*, 2019.

[50] D. Kedogan, D. Agrawal, and S. Penz, "Limits of anonymity in open environments," in *Information Hiding*, 2002.

[51] "Information leak from chat group. How do we find out which user is sharing information? ," https://security.stackexchange.com/questions/178435/information-leak-from-chat-group-how-do-we-find-out-which-user-is-sharing\-infor, 2018.

[52] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *FC*, 2004.

[53] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell-counting-based attack against Tor," *IEEE TON*, 2012.

[54] Y. Liu, D. Ghosal, F. Armknecht, A.-R. Sadeghi, S. Schulz, and S. Katzenbeisser, "Robust and undetectable steganographic timing channels for i.i.d. traffic," in *Information Hiding*, R. Böhme, P. W. L. Fong, and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 193–207.

[55] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *ESORICS*, 2010.

[56] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *NDSS*, 2011.

[57] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph: Protocol Obfuscation for Tor Bridges," in *CCS*, 2012.

[58] H. Mozaffari and A. Houmansadr, "Heterogeneous Private Information Retrieval," in *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[59] H. Mozaffari, A. Houmansadr, and A. Venkataramani, "Blocking-Resilient Communications in Information-Centric Networks Using Router Redirection," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019.

[60] "MTProto Mobile Protocol," https://core.telegram.org/mtproto.

[61] "MTProto proxy server for Telegram," https://mtproto.co/.

[62] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE S&P*, 2005.

[63] M. Nasr, A. Bahramali, and A. Houmansadr, "DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning," in *CCS*, 2018.

[64] M. Nasr, A. Houmansadr, and A. Mazumdar, "Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis," in *CCS*, 2017.

[65] L. H. Newman, "ENCRYPTED MESSAGING ISN'T MAGIC," https://www.wired.com/story/encrypted-messaging-isnt-magic/, 2018.

[66] "Nordvpn," https://nordvpn.com/, 2021.

[67] "How Many People Use Telegram in 2021? 55 Telegram Stats," https://backlinko.com/telegram-users#telegram-statistics, 2018.

[68] "[tor-project] Turning on timing obfuscation (iat-mode=1) for some default bridges," https://lists.torproject.org/pipermail/tor-project/2016-November/000776.html.

[69] J.-X. Pan and K.-T. Fang, *Maximum Likelihood Estimation*. New York, NY: Springer New York, 2002, pp. 77–158. [Online]. Available: https://doi.org/10.1007/978-0-387-21812-0_3

[70] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website Fingerprinting at Internet Scale," in *NDSS*, 2016.

[71] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *WPES*, 2011.

[72] ——, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *WPES*, 2011.

[73] K. Park and H. Kim, "Encryption is Not Enough: Inferring User Activities on KakaoTalk with Traffic Analysis," in *WISA*, 2015.

[74] H. V. Poor, *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.

[75] Y. J. Pyun, Y. H. Park, X. Wang, D. S. Reeves, and P. Ning, "Tracing traffic through intermediate hosts that repacketize flows," in *INFOCOM*, 2007.

[76] D. Ramsbrock, X. Wang, and X. Jiang, "A first step towards live botmaster traceback," in *RAID*, 2008.

[77] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated Website Fingerprinting through Deep Learning," in *NDSS*, 2018.

[78] D. Robertson, "The Licensing and Compliance Lab interviews Guillaume Roguez, Ring Project Director," *Free Software Foundation*, 2016, https://www.fsf.org/blogs/licensing/the-licensing-and-compliance-lab-interviews-guillaume-roguez-ring-project-director.

[79] "Russia orders Telegram to hand over users' encryption keys," https://www.theverge.com/2018/3/20/17142482/russia-orders-telegram-hand-over-user-encryption-keys, 2018.

[80] H. Saribeykan and A. Margvelashvili, "Security Analysis of Telegram," https://courses.csail.mit.edu/6.857/2017/project/19.pdf, 2017.

[81] J. Schectman, "Exclusive: Messaging app Telegram moves to protect identity of Hong Kong protesters," https://www.reuters.com/article/us-hongkong-telegram-exclusive/exclusive-messaging-app-telegram-moves-to-protect-identity-of-hong-kong-protesters-idUSKCN1VK2NI, 2019.

[82] M. Schliep, I. Kariniemi, and N. Hopper, "Is Bob Sending Mixed Signals?" in *WPES*, 2017.

[83] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the Burst: Remote Identification of Encrypted Video Streams," in *USENIX Security*, 2017.

[84] "Python Language Bindings for Selenium WebDriver," https://pypi.org/project/selenium/, 2020.

[85] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *ESORICS*, 2006.

[86] "Signal," https://signal.org/en/, 2021.

[87] "The Inside Story of How Signal Became the Private Messaging App for an Age of Fear and Distrust," https://time.com/5893114/signal-app-privacy/, 2020.

[88] "Signal-cli, an unofficial commandline and dbus interface for signalapp/libsignal-service-java," https://github.com/AsamK/signal-cli, 2021.

[89] "Why messaging app Signal is surging in popularity right now," https://www.cnn.com/2021/01/12/tech/signal-growth-whatsapp-confusion/index.html, 2021.

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

[91] "Most popular mobile messaging apps worldwide as of January 2018, based on number of monthly active users," https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps, 2018.

[92] T. Simonite, "FireChat Could Be the First in a Wave of Mesh Networking Apps," *MIT Technology Review*, 2014. [Online]. Available: https://www.technologyreview.com/s/525921/the-latest-chat-app-for-iphone-needs-no-internet-connection/

[93] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: routing attacks on privacy in tor," in *USENIX Security*, 2015.

[94] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *TIFS*, 2017.

[95] "What is Telegram? What do I do here?" https://telegram.org/faq, 2013.

[96] "Where is Telegram based?" https://telegram.org/faq, 2013.

[97] "Telegram and Instagram being restricted in Iran," https://techcrunch.com/2018/01/02/telegram-and-instagram-being-restricted-in-iran, 2018.

[98] "Tor: Pluggable Transports," https://www.torproject.org/docs/pluggable-transports.html.en.

[99] "Torgaurd," https://torguard.net/, 2020.

[100] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "SoK: Secure Messaging," in *IEEE S&P*, 2015.

[101] "Virtual Private Networking: An Overview," Microsoft Technet, Tech. Rep., 2011. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/bb742566(v=technet.10)

[102] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security*, 2014.

[103] T. Wang and I. Goldberg, "Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks," in *USENIX Security*, 2017.

[104] X. Wang, S. Chen, and S. Jajodia, "Tracking Anonymous Peer-to-peer VoIP Calls on the Internet," in *CCS*, 2005.

[105] ——, "Network flow watermarking attack on low-latency anonymous communication systems," in *IEEE S&P*, 2007.

[106] X. Wang, D. S. Reeves, and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," in *ESORICS*, 2002.

[107] "WhatsApp," https://www.whatsapp.com/, 2020.

[108] "Wickr," https://wickr.com/, 2021.

[109] "Wickr IO," https://wickrinc.github.io/wickrio-docs/#wickr-io, 2020.

[110] "Wire," https://wire.com/en/, 2020, 2021.

[111] "Wire Web Application," https://app.wire.com, 2020, 2021.

[112] "Wondershaper," https://github.com/magnific0/wondershaper.

[113] "WhatsApp reveals major security flaw that could let hackers access phones," https://www.cnn.com/2019/05/14/tech/whatsapp-attack/index.html, 2019.

[114] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Uncovering Spoken Phrases in Encrypted Voice over IP Conversations," *TISSEC*, 2010.

[115] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis," in *NDSS*, 2009.

[116] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *IEEE S&P*, 2007.

[117] Y. Zhang and V. Paxson, "Detecting Stepping Stones." in *USENIX Security*, 2000.

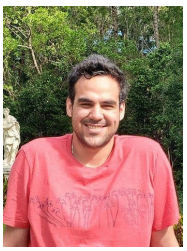
[118] Y. Zhu and R. Bettati, "Unmixing Mix Traffic," in *PETS*, 2005.

[119] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On flow correlation attacks and countermeasures in mix networks," in *PETS*, 2004.

Ardavan Bozorgi received the B.S. degree in Computer Engineering from the University of Tehran in 2018. He is currently working towards the Ph.D. degree with the College of Information and Computer Sciences at the University of Massachusetts Amherst, MA, USA. His research interests include privacy, security, and traffic analysis.

Alireza Bahramali received the B.S. degree in Electrical Engineering from the University of Tehran in 2017, and the M.S. degree in Computer Science from the University of Massachusetts Amherst in 2020. He is currently a Ph.D. student at the University of Massachusetts Amherst, MA, USA, studying computer science. His research interests include security and privacy, traffic analysis, machine learning, and wireless communication systems.

Fateme Rezaei received B.S. degree in Computer Engineering from Sharif University of Technology in 2015, M.S. and Ph.D. degrees in Computer Science from University of Massachusetts Amherst in 2021. Her research interests include Privacy and Security, and Traffic Analysis. She is currently working at Snap Inc. as a Software Engineer.

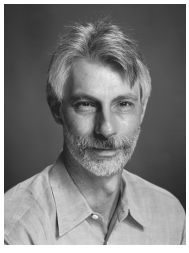


Amirhossein Ghafari received a B.S degree in Software Engineering from the University of Tehran in 2018 and an M.S. degree in Computer Science from the University of Massachusetts Amherst in 2022. He is currently a Software Engineer at NVIDIA, Santa Clara, CA. His research interests are network security, traffic analysis, and censorship.

Amir Houmansadr (Member, IEEE) received the Ph.D. degree from the University of Illinois at Urbana–Champaign in 2012. He is currently an Associate Professor with the Manning College of Information and Computer Sciences, University of Massachusetts Amherst. His broad area of research is network security and privacy. He has received several awards, including the Best Practical Paper Award at the IEEE Symposium on Security & Privacy in 2013, the Google Faculty Research Award in 2015, and the NSF CAREER Award in 2016.

Ramin Soltani received the B.S. degree in Electrical Engineering from the University of Tehran in 2009, M.S. degree in Electrical Engineering from the Sharif University of Technology in 2012, and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Massachusetts Amherst, in 2019. He is currently a Senior Engineer at Samsung Semiconductor Inc., San Diego, CA. His research interests include wireless communication, 5G NR, security and privacy, machine learning, and networking.

Dennis Goeckel (S'89–M'92–SM'04–F'11) received the Ph.D. degree from the University of Michigan in 1996. He is currently a Professor at the Electrical and Computer Engineering Department, University of Massachusetts Amherst, His research interests are in physical layer communications and wireless network theory. He received the NSF CAREER Award in 1999 for coded modulation for high-speed wireless communications and the University of Massachusetts Distinguished Teaching Award in 2007.



Don Towsley (Fellow, IEEE and ACM) holds a Ph.D. in Computer Science (1975) from University of Texas. He is currently a Distinguished Professor at the Manning College of Information & Computer Sciences,. His research interests include performance modeling and analysis, and quantum networking. He has received several achievement awards including the 2007 IEEE Koji Kobayashi Award and the 2011 INFOCOM Achievement Award.