Metadata Privacy Beyond Tunneling for Instant Messaging

Boel Nelson
Aarhus University
boel@cs.au.dk

Elena Pagnin
Chalmers University of Technology
elenap@chalmers.se

Aslan Askarov
Aarhus University
aslan@cs.au.dk

***Abstract*—Transport layer data leaks metadata unintentionally – such as who communicates with whom. While tools for strong transport layer privacy exist, they have adoption obstacles, including performance overheads incompatible with mobile devices. We posit that by changing the objective of metadata privacy for *all traffic*, we can open up a new design space for pragmatic approaches to transport layer privacy. As a first step in this direction, we propose using techniques from information flow control and present a principled approach to constructing formal models of systems with metadata privacy for *some*, deniable, traffic. We prove that deniable traffic achieves metadata privacy against strong adversaries– this constitutes the first bridging of information flow control and anonymous communication to our knowledge. Additionally, we show that existing state-of-the-art protocols can be extended to support metadata privacy, by designing a novel protocol for *deniable instant messaging* (DenIM), which is a variant of the Signal protocol. To show the efficacy of our approach, we implement and evaluate a proof-of-concept instant messaging system running DenIM on top of unmodified Signal. We empirically show that the DenIM on Signal can maintain low-latency for unmodified Signal traffic without breaking existing features, while at the same time supporting deniable Signal traffic.**

1. Introduction

Modern instant messaging (IM) services strive for strong end-to-end security. Services such as Signal, WhatsApp [1], Wire [2], and Facebook Messenger [3], all use the Signal protocol that is formally secure [4] and achieves ambitious security goals, such as post-compromise security, backward secrecy, confidentiality, and integrity. Still, these IM services lack strong metadata privacy, making them vulnerable to traffic analysis attacks. This is a serious deficiency, because traffic analysis remains an effective mechanism [5], [6] for surveillance and censorship used by governments, organizations, and internet service providers in over 100 countries [7]. For example, China's "great firewall" actively probes and censors privacy tools [8]. Although collecting metadata may seem non-intrusive, metadata is used to make critical decisions – "we kill people based on metadata", as former US government official general Hayden [9] put it. "Harvest today, analyze tomorrow" is a viable adversarial strategy for many such actors.

While the general problem of metadata privacy has been extensively studied, there are both social and technical barriers that prevent adoption of existing privacy tools to IM services. On a social level, people are either unaware of privacy tools [10] or have diverse misconceptions about them [11]. Adding to the existing problem, people also find these tools too complicated to use, or lack the knowledge of how to use them [12]. For example, Norcie et al. [13] investigated the Tor Browser Bundle and found that users experienced usability issues such as the browser's launch time and difficulties downloading and installing it. Beyond the challenges of usability, there are risks of being scrutinized for having a particular app installed [14], [15].

On a technical level, available tools are far from perfect. The Tor project [16] although relatively popular with 2M active users [17], is vulnerable to de-anonymization [18], denial of service (DoS) [19], and traffic analysis [20]. Because Tor can be automatically fingerprinted [5], [6], it is also easy to block (ironically, the authors of this paper themselves were blocked from accessing the Tor project's website on their university network). Metadata private focused IM tools that run on Tor [21], [22], [23], [24] suffer from the same issues. Other tools that hide traffic by imitating well-known apps do not produce credible traffic [25].

The strongest guarantees for metadata privacy are provided by dedicated protocols. In particular, round-based, DC-nets like, protocols [26], [27], [28], [29], where predetermined rounds make traffic patterns indistinguishable, are able to resist traffic analysis. However, round-based protocols are both resource exhaustive and inflexible. The rounds themselves require constant overhead, which results in poor performance [30], making them especially infeasible for resource constrained devices such as phones or wearables. Moreover, a major obstacle with round-based protocols is that they depend on fixed sets of individuals participating. That is, participants cannot join or leave without changing the privacy guarantees. Finally, round-based protocols are also easy to fingerprint and block.

Existing approaches to metadata privacy all have in common that they focus on the strong objective of *metadata privacy for all users all the time*. However, such a strong objective significantly delimits the design space of possible solutions. We propose a different, pragmatic objective: **rather than offering privacy to all users all the time, let us offer privacy to all users some of the time**. This shift in objective expands the design space for metadata privacy to new solutions.

Our new approach is to incorporate metadata privacy into an existing store-and-forward IM protocol. To that extent, we present *Deniable Instant Messaging* (DenIM)– an IM protocol that provides both message confidentiality

and metadata privacy. DenIM distinguishes two kinds of messages: (i) *regular* messages that do not require metadata privacy, and (ii) *deniable* messages that do require it. Regular and deniable communication is combined in one system, and users decide which messages to send privately. To withstand traffic analysis, deniable messages are not communicated immediately, instead they are piggybacked on top of the regular messages, which in turn requires that all messages are extended by a small known amount of bytes. The store-and-forward server breaks the link between the sender and receiver of a private message by buffering the message until there is an opportunity to piggyback it on some other regular message to the receiver. It is vital that the messages are extended even if the communicating parties have nothing to say, in which case a dummy payload is sent instead. To minimize overhead, the size of the payload must be small in proportion to the overall communication.

The importance of incorporating metadata privacy into an existing IM protocol aligns with earlier observations in the literature. As EFF put it, "An app with great security features is worthless if none of your friends and contacts use it" [31]. In their paper "*Practical Traffic Analysis Attacks on Secure Messaging Applications*", Bahramali et al. [32] recommend that metadata privacy for IM should be adopted by IM services to be effective. An encouraging development in this direction already is WhatsApp's use of the Noise Protocol Framework (NPF) [33] to protect certain metadata [1]. Finally, Zuckerman's [34] *Cute Cat Theory of Censorship* posits that platforms that combine entertainment with political activism are more resilient to censorship than dedicated political platforms.

In our case, we pair DenIM with the (unmodified) Signal protocol. We call the resulting system *DenIM on Signal*. We chose Signal because it provides the state-of-the art security guarantees in instant messaging, including: forward secrecy, backward secrecy (post-compromise security), data confidentiality, and integrity (see [4] for detail).

DenIM's piggybacking of deniable messages is a form of tunneling (e.g., [35], [36], [37]). Yet tunneling alone is insufficient. The reason is that in settings where adversaries are legitimate users in the system, information may be leaked inadvertently through parts of the protocol state that are shared between all users. For example, in Signal, adversaries can gain information about other users through the state of the key distribution center because the protocol allow users to run out of keys – this allows adversaries to count the number of keys a user has and in extension allows the adversary to deduce how many conversations a user is part of.

To ensure that DenIM guarantees metadata privacy for deniable messages, including unknown attacks, we use techniques from secure information flow. Our insight is to model user deniable behavior as *user strategies* [38] – a technical device that is traditionally used for specifying semantic security of interactive and nondeterministic programs. In DenIM, a user strategy is a function that given a history of the user's communication determines their next deniable action, e.g., send a deniable message, request key material from the server to initiate new deniable communication, or block a user from receiving deniable messages. We recast the notion of metadata privacy as strategy-based noninterference: *user strategies must not leak through the protocol*. The significance of this insight is that because noninterference is an end-to-end characterization, proving noninterference requires that there is no way in which the sensitive information may leak anywhere in the protocol, not just on the transport layer. In essence, this guides the features and non-features of DenIM. For example, DenIM restricts the notification of user blocking, because notifying a user that they have been blocked leaks information about the blocking user's deniable behavior.

The contributions of this paper are as follows:

- It presents a deniable variant of the Signal protocol (Section 2.2) called DenIM (Section 4.1), that supports both the original strong cryptographic guarantees of Signal, and metadata privacy.
- It presents a system design that layers deniable Signal messages on top of the unmodified Signal protocol, which we call DenIM on Signal (Section 4).
- It presents a formal privacy analysis (Section 5) that constitutes a principled approach of using information flow techniques to guarantee privacy by proving noninterference.
- It presents a proof-of-concept implementation (Section 6.1) of an instant messaging system with DenIM.
- It presents an empirical evaluation (Sections 6.2 and 6.3) of the performance of DenIM.

2. Background

This section provides an overview of instant messaging (IM), and the main machinery in the Signal protocol which we design a deniable version of in Section 4.1.

2.1. Instant messaging

In 2019, instant messaging (IM) services had seven billion registered accounts worldwide [39]. The most popular IM services include WhatsApp (2B users), Facebook messenger (1.3B users), iMessage (estimated to 1B users), Telegram (550M users), and Snapchat (538M users) [40], [41]. While IM appears deceptively simple, the sheer amount of users and traffic (69M messages/min in 2021 [42]) present several engineering challenges. Keeping up with the demands requires deploying and maintaining robust systems. As an example, WhatsApp's architecture handles over one million connections per server [43].

All major IM services, including WhatsApp, Facebook messenger, Telegram and Snapchat, use centralized servers to forward messages [32]. Many IM apps also come with end-to-end encryption, in addition to server-client encryption (through TLS). Telegram uses their own protocol, MTProto [44], iMessage uses RSAES-OAEP [45], and Snapchat uses an unnamed encryption scheme for some of its content [46]. The most popular protocol is Signal [47], [48], which also has the strongest security guarantees of the mentioned protocols, and is used by WhatsApp [1], Facebook Messenger [3], Wire [2], ChatSecure, Conversations, Pond, the Signal app, and Silent Circle [4]. The Signal protocol is formally secure [4], and is based on Off-the-Record Messaging (OTR) [49] and Silent

Circle Instant Messaging Protocol (SCIMP) [50]. Despite strong cryptographic guarantees, none of the centralized IM services support transport layer privacy for IM.

2.2. The Signal protocol

At a high level the Signal protocol [51] realizes an end-to-end secure communication channel between two parties that exchange instant messages in a possibly asynchronous way, i.e., users may not be online at the same time. Signal distinguished itself among the landscape of messaging protocols in that it achieves ambitious security goals including: forward secrecy, backward secrecy (post-compromise security), data confidentiality, and integrity (see [4] for detail). This is obtained by managing several different cryptographic keys (Table 1), relying on semi-trusted centralized servers (to store and forward messages, and implement a key distribution center), and combining three cryptographic primitives: a key derivation function (KDF), a non-interactive key-exchange protocol (namely DH for Diffie-Hellman) for initiating new sessions, and an authenticated encryption scheme with associated data (AEAD).

2.2.1. Keys used in Signal. In Signal, each user U holds a set of keys that identify the user, and are used to initiate new sessions (chats) and to AEAD-encrypt messages. Table 1 provides a categorization of the cryptographic key material of Signal that is relevant to this work. Keys employed only to set up new sessions are highlighted with the symbol $\bigstar$.

Name	Key(s)	Usage
Identity key-pair$\bigstar$	$\{idpk_U, idsk_U\}$	Long-term
Mid-term key-pair$\bigstar$	$\{prepk_U, presk_U\}$	Mid-term
Ephemeral key-pair$^{(\bigstar)}$	$\{epk_U, esk_U\}$	One-time
Master secret	ms	One-time
Message key	$mk_{x,y}$	One-time

TABLE 1: List of Signal's keys that are relevant to this work. Ephemeral keys are used in various parts of the Signal protocol, when employed in session initialization they are commonly called one-time keys.

2.2.2. Overview of the Signal Protocol. The Signal protocol is made of three main steps:

User registration. Run once in the lifetime of a user in the system. This step entails storing a user's public key material in the Signal server, namely $idpk_U, prepk_U$, and a set of (one-time) ephemeral public keys $\{epk_U^{(1)}, \ldots, epk_U^{(n)}\}$.

New-session initialization. Run once per new session initiated by a user. This step is used to start a new chat. The requesting user A interacts with the server to obtain the handle of B, another user, consisting of $idpk_B, prepk_B$ and a single one-time public key $epk_B^{(i)}$. A uses B's keys together with their identity secret key, long term secret key and an ephemeral secret key to run a non-interactive key-exchange and generate a master secret key ms_{AB}, that is computable only by A and B.

The double ratchet mechanism for messaging. Run every time the user receives or sends a new message. In Signal every message is AEAD-encrypted under a different message key $mk_{x,y}$. We index the message keys by two non-negative integers x, y that operate as coordinates. The value x identifies the current sender, y the number of messages sent by the current sender since the last change of speaker. Thus even values of x correspond to events where the current speaker is the initiator of the chat, while y denotes how many messages the sender of level x has sent so far. In order to securely derive new keys from previous ones, the double ratchet mechanism ingeniously combines two KDFs.

3. System design

This section presents the scope and goals of our deniable messaging system, *DenIM on Signal*. We start by defining the threat model, which will dictate the necessary design goals and trust assumptions.

3.1. Threat model

We consider a global active adversary who participates in the deniable protocol. The adversary can:

- Observe the entire network, including messages to and from the server, and to and from the users.
- Insert or modify traffic.
- Participate in the protocol. This gives to the adversary access to the parts of the protocol state that are accessible to all protocol participants, including requesting other users' keys from the key distribution center (KDC), and sending messages.

We assume that the adversary cannot compromise the internal state of honest parties, including servers. The adversary may control multiple nodes in the network, and can collude with other adversaries.

Under this threat model, the adversary could for example be an internet service provider, or a nation-state. Given these capabilities, the goal of the adversary is:

- To learn or to alter the payload of deniable traffic between honest parties.
- To learn whether a given network message contains deniable payload or not.
- To learn whether two parties have an ongoing exchange of deniable traffic or not.

Note that our system makes traffic 'deniable' on the transport layer, which is different from e.g., deniable encryption [52] where the goal is to give deniability for the message content (plaintext) rather than hiding fact of communication. Our goal is that deniable communication should be unobservable to an adversary – we prove (Theorem 1) that this is the case by guaranteeing that network traces with and without deniable communication appear indistinguishable to the adversary.

3.2. Design goals

The high-level goal of our system is to be resilient against adversaries with the goals in Section 3.1. Additionally, tunneling deniable traffic inside instant messaging systems

requires making decisions regarding performance trade-offs between the deniable traffic and the regular traffic. We derive the design goals for security and privacy (Section 3.2.1) based on the threat model and instant messaging use case, and design goals for performance (Section 3.2.2) from the use case.

3.2.1. Security and privacy goals.

Confidentiality of users' deniable behavior. A consequence of our threat model is that an adversary could try to infer users' deniable behavior both by observing the network, or by observing shared protocol states. Adequate protection measures therefore depend on data the users generate by interacting with the deniable protocol not leaking into channels the adversary can observe (the network and the shared state). That is, a successful implementation depends on proving noninterference between the deniable protocol and the protocol it piggybacks on – noninterference ensures all of the users' input to the deniable protocol is kept confidential, not just that the network traffic is protected.

Privacy guarantees independent of the number of online users. To achieve strength-in-numbers, we aim for the design where the privacy guarantees do not depend on the dynamic behavior of the system, i.e., users may join or leave the system without significantly affecting the privacy of others. This means that the system should tunnel the deniable traffic using an observable protocol that does not achieve transport layer privacy on its own.

Strong security guarantees for deniable messages. Message content should be protected using state-of-the-art techniques, which for IM can be achieved via the Signal protocol. Signal is more than just a mere key exchange protocol; it is designed to deliver not only confidentiality and integrity, but also more advanced security features such as key healing. We aim to maintain the same security benefits provided by Signal by carefully building our deniable messaging machinery around the Signal protocol in a way that does not impact Signal's security functionalities.

3.2.2. Performance goals.

Parameterizable bandwidth overhead for deniable traffic. To control the privacy-performance trade-offs in the system, the deniable payload overhead should be a global tuneable parameter that is set on a case-by-case basis to match a user population's demand for deniable traffic. There should be no limitation on the length of the regular traffic.

Prioritize low latency for regular traffic. We prioritize the performance of regular traffic – it is important that users continue to use the regular IM system – above the performance of the deniable traffic. This creates an asymmetry in the latency of regular and deniable communication. When using the protocol for regular Signal, traffic is forwarded immediately resulting in low latency overhead. For DenIM, the latency depends on when traffic can be safely piggybacked. While a system with different privacy guarantees for different messages like this has not yet been studied from the usability perspective, we assume that a higher latency overhead for deniable communication is tolerable as the privacy guarantees are stronger.

3.3. Trust assumptions

The previously stated design goals, combined with the threat model, leads to the following trust assumptions:

- The adversary cannot access the internal state of honest parties.
- Users trust receivers of their deniable traffic, i.e. users are by design not able to deny having sent traffic to their intended receiver.
- Users' deniable behavior does not influence their regular behavior, e.g., a user does not send more regular traffic than they normally would to piggyback their deniable traffic.
- The forwarding servers are trusted.
- The KDC is trusted, and can generate ephemeral keys on behalf of a user in case the user's deniable ephemeral keys have been depleted. All user-generated keys are signed by the issuing user, just like in Signal. This means that the KDC cannot impersonate users, but may have access to one third of the key material to initialize the master secret in certain cases.
- Users do not issue deniable key requests for adversaries' keys, and do not respond to deniable Signal sessions initiated by adversaries.

Note that our trust assumptions to a large extent are inherited from the use case, IM, and from the Signal protocol. For example, centralized, trusted servers is the natural setting for IM. Moreover, Signal assumes that a user is able to verify that the receiver of messages is a trusted party using an out of bounds channel – in their deployment they support this by providing a QR code that both parties are supposed to verify in person.

Our formal model (Section 5) incorporates the trust assumptions at a technical level.

4. DenIM on Signal

This section presents *DenIM on Signal*, an instant messaging system that supports two different protocols: regular Signal, and our deniable variant of Signal, *Deniable Instant Messaging* (DenIM). DenIM is a centralized IM protocol with both the cryptographic guarantees of the Signal protocol, and transport layer privacy for messages. In DenIM on Signal the deniable protocol, DenIM, piggybacks on an unmodified version of Signal.

At a high level, DenIM on Signal provides users with two communication abstractions: sending 'regular' Signal traffic that is not resilient to traffic analysis, and sending 'deniable' Signal messages that come with transport layer privacy. To prevent an adversary from trivially inferring which users are communicating, DenIM on Signal uses a simple centralized architecture where traffic is routed through a trusted server. The server forwards the regular Signal traffic immediately, and stores the DenIM traffic until there is regular traffic for the intended recipient to piggyback on. To prevent an adversary from tracing traffic by fingerprinting it as it is forwarded by the server, the traffic between clients and server is sent over TLS.

We model and prove the security of our implementation in Section 5, and empirically evaluate how bandwidth overhead affects system performance both for deniable and regular traffic in Section 6.

4.1. Protocol details

In this section we elaborate on the technical details of DenIM. We explain how the deniable part of a network message is created (Section 4.1.1), how and where deniable parts get buffered (Section 4.1.2), and which content can be carried in the deniable part i.e., what deniable actions are supported by DenIM (Section 4.1.4). DenIM is a variant of Signal – we make a minor change to the Signal protocol (Section 4.1.3) to ensure that DenIM is a deniable variant of Signal (the standard Signal protocol would otherwise leak information about a user's deniable sessions), but otherwise encapsulates Signal. We stress that this modification does not impact the cryptographic security.

Communication flow by example. Figure 1 presents an example of communication flow in DenIM on Signal. Alice (upper left) has queued a deniable message (D) waiting to be sent to Dorothy (lower right). As Alice sends a regular message R_1 to Bob (upper right), part of Alice's deniable message for Dorothy, D_1 is added to the deniable padding. The server immediately forwards the regular message to Bob, and as there are no deniable messages queued for Bob, the message is padded with dummy padding. Next, Charlie (lower left) sends a regular message R_2 to Dorothy. The server forwards the regular message to Dorothy, and adds part of Alice's deniable message that has been queued on the server to the padding of the message for Dorothy. Where relevant, the internal steps are ordered to avoid timing leaks; for example, queuing of the deniable message (2c), takes place after forwarding (2b).

4.1.1. Deniable padding. The size of the deniable part of a network message is $l \cdot q$, where l is the length of the regular part, and q is a system-wide padding parameter set by the server. Both l and q are publicly known. Because of the strict size limit on the deniable part, deniable communication is chunked to fit the deniable part. If there is no deniable communication (or its length is less than $l \cdot q$), the deniable part is padded to always reach length $l \cdot q$.

4.1.2. Deniable buffers. Each client keeps a deniable buffer to allow the user to queue new deniable messages at any time. Any time the user sends a regular message, $l \cdot q$ bytes of the oldest deniable message in the buffer are added as padding to the regular message.

The server keeps one deniable buffer per user. When receiving a message, the server extracts the regular part of the message, and creates a new message for the receiver and adds a deniable part – either from the recipient's deniable buffer or dummy padding. Note that depending on the implementation strategy, there may be subtle timing channels here. In particular, it may be desirable to handle the deniable parts of the incoming message only after the response has been processed. This is because differences in timing could leak information about the deniable part – such as how many deniable actions were piggybacked. We elaborate more on potential side-channels and mitigation strategies in Section 7.

4.1.3. Changes to Signal. A known (and often overseen) weakness of Signal is that if a user runs out of ephemeral keys, new sessions are initialized with less randomness by reusing the mid-term key instead of a one-time use key. Standard Signal mitigates this issue by letting users refill ephemeral keys at any point in time to avoid running out of keys.

However, if the key distribution center (KDC) were to fail to return a deniable ephemeral key for a user because they have run out of keys, it would leak information about the number of deniable sessions a user has. Therefore, the number of deniable ephemeral keys each user stores in the KDC must be secret to the adversary. To limit traffic between the KDC and the user, and still maintain the randomness for the generation of the master secret, DenIM lets the KDC generate new deniable ephemeral keys on the behalf of the user. To keep the KDC and client in sync, the client provides a seed for the KDC to be used as input for a deterministic key generator upon user registration. The KDC keeps a counter for how many times the key generator has been used, and the value of the counter is sent to the corresponding client with each deniable message. We stress that this change to Signal still means that the deniable messages will be end-to-end encrypted between sender and recipient – the server will at most have access to one of the three keys (the ephemeral one) used to generate the master secret. This change means that the KDC in our protocol requires more trust than in standard Signal, but it still cannot impersonate users.

4.1.4. Supported deniable actions. In DenIM we support all actions needed to implement the Signal protocol's session initialization and double ratchet mechanism. However, we do not support all functionality that the Signal IM app supports. For example, we do not support group chats and video calls. Additionally, we have intentionally chosen not to support specific functionality to prevent adversaries from learning about users' deniable behavior. First, we do not support read receipts for messages, since they leak to an adversary if or when a user receives a deniable message. Second, we support users blocking other users, but with the twist that blocked users are not informed that they have been blocked. An adversary that could learn that they have been blocked can for example flood a user with messages to provoke the user into blocking them, and the adversary can then use the time of being blocked to infer that a user has received their deniable messages, which also leaks that the user's deniable buffer has been drained.

In order to use DenIM, each user needs to upload Signal keys and a seed for the key generator to the KDC through registering. We support the following *deniable* Signal actions:

Key exchanges. Users can send key requests to initiate new Signal sessions, and the server responds with a key response containing the user's public identity key, mid-term key, and crucially, always an ephemeral key unlike in standard Signal where the KDC may run out of ephemeral keys. While the KDC always provides ephemeral keys

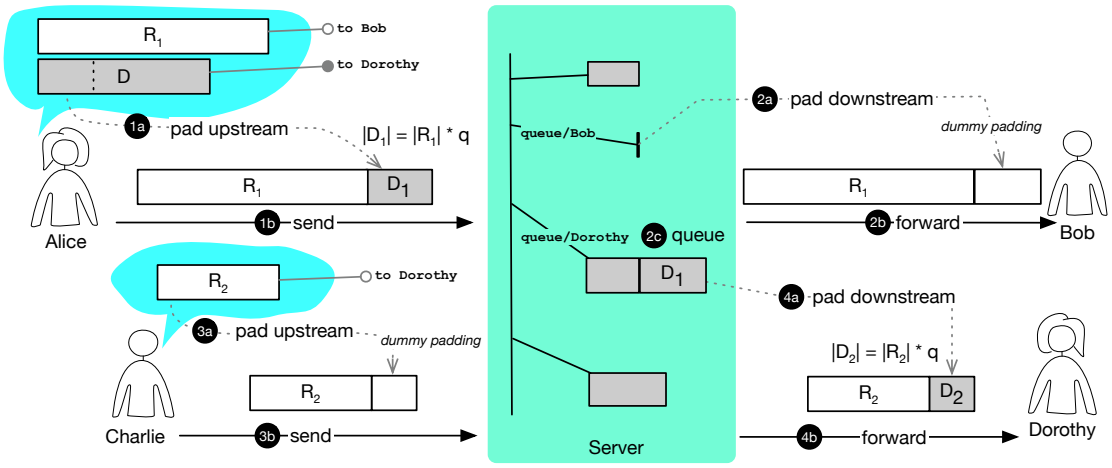


Figure 1: Diagram representation of the communication flow in DenIM. R and D denote regular and deniable communication, respectively. Odd steps (1 and 3) are performed by clients, even steps (2 and 4) are performed by the server. All communication to/from the server is TLS encrypted.

as part of the key material, users can also issue key refills.

User message. Signal messages containing ciphertext and Signal headers to support ratcheting.

Block request. Enforced by the server by silently dropping messages by blocked users instead of buffering them.

5. Formal analysis

This section presents the security and privacy analysis of DenIM on Signal. In this formal privacy analysis we abstract away the details of the internal state of the unmodified version of Signal, which allows for a cleaner modeling using an abstract centralized protocol. Due to space, the full formal model and accompanying proofs are available in Appendices A, B and C.

5.1. Cryptographic guarantees

From the cryptographic point of view, DenIM is an instance of Signal. Since our protocol does not alter the mechanisms employed to derive key material, encrypt, and decrypt messages, it trivially satisfies the cryptographic guarantees offered by the Signal protocol. More formally, since the security analysis of Cohn-Gordon et al. [4] rules out ephemeral key refill issues, it applies to DenIM as well. This is straightforward for regular traffic, since DenIM ephemeral keys are refilled as in Signal for regular messages. For deniable ones we fall back to use a secret seed shared between a user and the KDC. Employing a secure pseudo-random generator (PRG) to derive deniable ephemeral keys from the seed brings us to the setting of Cohn-Gordon et al.'s security analysis, since we assume the server to be honest.

5.2. Protecting deniable behavior

To ensure privacy against the network level adversary described in Section 3.1, we need to secure the two channels: the network, and the shared state. Ultimately, an adversary will examine network messages to try to distinguish dummy padding from deniable content, and observe the traffic patterns as well as trying to access shared states to infer something about a target user's deniable behavior.

Making deniable traffic indistinguishable. Part of securing the network relies on guaranteeing that the adversary cannot fingerprint the deniable part of network messages – for this we need a TLS-like connection between each client and the server. TLS, however, is not sufficient, since the adversary can participate in the protocol themselves, and it is important they do not infer anything about deniable behavior of other users through such participation.

Protecting traffic patterns and shared states. We need to prove that a user's deniable behavior, which is reflected by the deniable protocol's internal state both client-side and server-side, does not leak into the regular protocol's state and becomes visible on the network or in the server's state.

The key technical insight of our approach is that by developing a fine-grained formal model of the deniable protocol and the tunnel, we can formulate the privacy problem as a form of noninterference [53] property, where the confidential input to the system is users' deniable behavior. This allows us to leverage state-of-the-art formal machinery for information flow to precisely characterize the guarantees of the deniable protocol. The main result of our analysis is crystallized at the end of this section as Theorem 1.

The model formalizes both client and server behavior in DenIM, but abstracts away the details of the tunnel that constitutes the regular part of a network message. For simplicity, we model the message forwarding server and the KDC as one 'server' network node. To model Signal's double ratchet mechanism as part of our deniable protocol, we introduce a notion of *abstract ratchets*, drawing on the techniques from the literature on symbolic cryptography [54]. Abstract ratchets turn out to be a particularly fitting technical gadget to reason about important privacy

guarantees of DenIM, without diving into probabilistic modeling. The rest of the section highlights some of the aspects of the formal model.

5.2.1. Communication model. Our communication model uses the notion of upstream and downstream messages. Intuitively, upstream events are the events that flow in the direction of the server, e.g., sending a message, or registering a user. Downstream events are the events that flow from the server to a client. Each message consists of the client node designation n, the host protocol payload ρ that we abstract away from, and a DenIM event ξ. Additionally, downstream messages include the counter c for synchronizing clients key generation with server-generated keys introduced in Section 4.1.3. We use meta variable α for messages.

$$\alpha ::= \mathsf{denimup}(n, \rho, \xi) \mid \mathsf{denimdn}(n, \rho, \xi, c)$$

DenIM events correspond to the supported deniable actions (cf Section 4.1) and the corresponding server responses. The events contain information about the originator and the potential destination, as well as associated information, such as symbolic keys k and symbolic ratchet tokens tok. Here, we assume that keys are elements of the abstract key space $\mathbf{K}$.

Ratchet tokens are sets of the form:

$$\{(n_1, k_1), (n_2, k_2), (x, y)\}$$

and correspond to an active session between nodes n_1 and n_2, using k_1 and k_2 that are used for initiating the session. Coordinates (x, y) correspond to the Signal message coordinates as described in Section 2.2.2.

Without loss of generality, we assume that all deniable events, including dummy padding, are of the same length. This allow us to omit the actual payload from the events grammar, and instead rely on payload confidentiality that we get from Signal. The following grammar describes DenIM events:

$$\begin{aligned}\xi ::= {} & \mathsf{send}(n \to n, tok) \mid \mathsf{fwd}(n \to n, tok) \\ & \mid \mathsf{refill}(n, k) \mid \mathsf{kreq}(n \text{ for } n) \mid \mathsf{kresp}((n, k) \text{ for } n) \\ & \mid \mathsf{block}(n \text{ by } n) \mid \bullet\end{aligned}$$

Here, the event $\mathsf{send}(n_1 \to n_2, tok)$ corresponds to the upstream event of sending a deniable message from n_1 to n_2. Once received by the server, the server forwards it to the destination as $\mathsf{fwd}(n_1 \to n_2, tok)$.

Based on the event grammar, we define a few auxiliary functions.

Definition 1 (Upstream and downstream events; event sender and receiver)**.** *Given an event ξ we define* event direction*, denoted as* $\mathrm{dir}(\xi)$*, and* event sender and receiver*, denoted as* $\mathrm{snd}(\xi)$ *and* $\mathrm{rcv}(\xi)$*, respectively, as per Table 2.*

We lift these functions to messages, so we write, e.g., $\mathrm{dir}(\alpha)$ to get the direction of message α.

Next, we introduce the notion of traces, and their local projections.

Definition 2 (DenIM trace)**.** *A* DenIM trace*, denoted as τ, is a sequence of DenIM messages $\alpha_1, \ldots, \alpha_n$. Empty traces are denoted as ϵ.*

We use $\cdot$ to denote trace concatenation.

Definition 3 (Local trace projection)**.** *Given a trace τ, and a node n, define* local trace projection*, written $\lfloor \tau \rfloor_n$, to be the subtrace of τ consisting only of messages that are local to n. It is defined inductively as $\lfloor \epsilon \rfloor_n = \epsilon$, and*

$$\lfloor \alpha \cdot \tau \rfloor_n = \begin{cases} \alpha \cdot \lfloor \tau \rfloor_n & \text{if } \mathrm{dir}(\alpha) = \uparrow \wedge \mathrm{snd}(\alpha) = n \\ & \quad \vee \mathrm{dir}(\alpha) = \downarrow \wedge \mathrm{rcv}(\alpha) = n \\ \lfloor \tau \rfloor_n & \text{otherwise} \end{cases}$$

To model users' deniable behavior, we introduce the notion of strategies [55]. A strategy ω is a function that takes a user-local view of the trace and decides the next upstream event. These decisions eschew low-level details of ratcheting, instead only providing information about the kind of the event κ given by the following grammar.

$$\kappa ::= \mathsf{SEND}\ n \mid \mathsf{REFILL} \mid \mathsf{KREQ}\ n \mid \mathsf{BLOCK}\ n \mid \bullet$$

Event kinds and events are related by function $\mathrm{kind}(\xi)$, defined in Table 2. Next, we define a notion of strategy validity, which allows us to qualify user behavior.

Definition 4 (Strategy send validity)**.** *Given a set of adversary nodes $\mathbf{N}$, and a strategy function ω that runs on node n, say that this strategy is* send-valid w.r.t. $\mathbf{N}$*, if the following conditions hold for all traces τ:*

Send well-formedness *if $\omega(\tau) = \mathsf{SEND}\ n_{dest}$ then τ must contain message α such that either*

- $\alpha = \mathsf{denimdn}(n, \rho, \mathsf{kresp}((n_{dest}, k) \text{ for } n), c)$*, or*
- $\alpha = \mathsf{denimdn}(n, \rho, \mathsf{fwd}(n_{dest} \to n, tok), c)$

DenIM send validity *if $\omega(\tau) = \mathsf{SEND}\ n_{dest}$ or $\omega(\tau) = \mathsf{KREQ}\ n_{dest}$ for some node n_{dest}, then $n_{dest} \notin \mathbf{N}$.*

Send well-formedness is a technical requirement that ensures that the strategy follows the basic mechanics of the Signal protocol, since DenIM is a variant of Signal. In order to send a message to n_{dest}, we either need to have obtained the key material from the server to initiate the session, or n_{dest} needs to have initiated the session already. We note that there is no loss of generality in restricting the adversary to Signal messages in our model. All malformed deniable messages to the server will be dropped (in that way, they are morally equivalent to dummy messages); all malformed messages to the client will be dropped following the Signal rules.

DenIM send validity states that non-malicious nodes do not communicate with the adversary. This is a critical constraint that is important for the DenIM privacy theorem. The following example demonstrates an attack that is possible if DenIM send validity does not hold.

Example attack. Consider a system with three users: Alice, Bob, and the adversary, Eve. We assume that Eve observes all the network traffic, including messages from/to Alice/Bob via the server. Consider the trace consisting of the following messages.

ξ	dir	snd	rcv	kind	Description
$\mathsf{send}(n_1 \to n_2, tok)$	$\uparrow$	n_1	n_2	SEND n_2	Send a message from n_1 to n_2 with the ratchet token tok
$\mathsf{fwd}(n_1 \to n_2, tok)$	$\downarrow$	n_1	n_2		Forward a message from n_1 to n_2 with the ratchet token tok
$\mathsf{refill}(n, k)$	$\uparrow$	n		REFILL	Upload keys k belonging to n
$\mathsf{kreq}(n_1 \text{ for } n_2)$	$\uparrow$	n_2		KREQ n_1	Request key belonging to n_1 to be sent to n_2
$\mathsf{kresp}((n_1, k) \text{ for } n_2)$	$\downarrow$		n_2		Key response with key k belonging to n_1 sent to n_2
$\mathsf{block}(n_1 \text{ by } n_2)$	$\uparrow$	n_2		BLOCK n_1	Block n_1 from messaging n_2
$\bullet$				$\bullet$	Dummy event, used when there is no deniable communication to send or forward

TABLE 2: Auxiliary functions on unobservable events.

1) Alice sends a message to the server $\alpha_1 = \mathsf{denimup}(\text{Alice}, \rho_1, \xi_1)$ with some host payload ρ_1. Eve observes α_1, but does not know whether ξ_1 is a deniable message for Bob or not. If ξ_1 is for Bob, it would mean that ξ_1 must be queued on the server.
2) Eve requests Bob's key material from the server: $\alpha_2 = \mathsf{denimup}(\text{Eve}, \rho_2, \mathsf{kreq}(\text{Bob for Eve}))$.
3) The server responds to Eve with Bob's key k_1 via message $\alpha_3 = \mathsf{denimdn}(\text{Eve}, \rho_3, \mathsf{kresp}((\text{Bob}, k_1) \text{ for Eve}), c_1)$.
4) Eve uses k_1 to initiate a deniable session with Bob. Eve generates their own key k_2 and a symbolic ratchet token $tok_1 = \{(\text{Eve}, k_2), (\text{Bob}, k_1), (0, 0)\}$. Here, $(0, 0)$ are the Signal coordinates of the message. Eve constructs a deniable message $\xi_2 = \mathsf{send}(\text{Eve} \to \text{Bob}, tok_1)$, and sends message $\alpha_4 = \mathsf{denimup}(\text{Eve}, \rho_4, \xi_2)$. Upon receiving this message, the server must queue ξ_2.
5) The server sends a message $\alpha_4 = \mathsf{denimdn}(\text{Bob}, \rho_5, \xi_3, c_2)$ to Bob. Here ξ_3 is either ξ_1, in case it was not dummy, or ξ_2.
6) There are no other downstream messages to Bob up until this point.

Suppose that ξ_1 is dummy, and Bob receives Eve's message ξ_2. If Bob replies to it, they must construct a token $tok_2 = \{(\text{Eve}, k_2), (\text{Bob}, k_1), (1, 0)\}$, Note the change in coordinate 1 here. When Eve receives this message they know that Bob has received ξ_2, because of the change in the ratchet coordinates. In particular, these ratchet coordinates cannot correspond to Bob reaching out to Eve on their own. Because there are no other downstream messages to Bob, Eve knows that the only way ξ_2 can be forwarded to Bob is in step 5, which means that Alice's ξ_1 must be dummy!

The attack relies only on the ordering of the events to up to the Bob's reply. The actual trace may otherwise be infinite.

This example shows the importance of the DenIM send validity requirement. A similar example can be constructed for allowing users to request the adversary's keys. Note that blocking an adversary is allowed by Definition 4. The reason is that unlike sending a message to the adversary or requesting their key, the server does not propagate these events to the blocked users – adversaries do not learn that they have been blocked.

Note that our provable methodology means that DenIM is secure against *all* attacks that fall within the threat model, not just the example above. In fact, the example above has been discovered during the proof process.

A final technical aspect of strategy validity is that we require strategies to be *deterministic w.r.t. formal randomness*. Intuitively, strategies may differentiate among keys, but cannot depend on the actual bit representation of the keys. This reflects the probabilistic nature of generated keys, and is a natural requirement in symbolic cryptographic models.

5.2.2. System state. At the top level, our system is represented as network configurations of the form $\langle \mathcal{S}, \mathcal{U}, \tau \rangle$, where $\mathcal{S}$ is the server state, $\mathcal{U}$ is a collection of user states of the form $\omega; \sigma$, where ω is the user deniable strategy, and σ is the user's local Signal state that contains the necessary information for session bookkeeping. Finally, τ is the global trace produced so far.

We formally define user and server state, respectively, as follows.

Definition 5 (User local Signal state)**.** *A user state σ is a tuple $\langle n, L, M, R, s, c, g \rangle$, where n is the identity of the user, L and M are sets containing the user's own keys or keys of other users paired with a state of the form (k, fresh) or (k, used), R is the abstraction of a ratchets mapped by receiving users to $(w, \{k_1, k_2\}, I)$ containing the starting index assigned to the user, the initiating keys, and the observed message indices I, s is a seed for the deterministic random number generator, c is a formal randomness counter for the server-side key generation, and g is the formal randomness counter for the client-side key generation.*

Definition 6 (Server state)**.** *A server state is represented by a tuple $\langle H, rq, D \rangle$ where H represents the state of an arbitrary host protocol, rq is a list of outgoing regular messages, and D represents the deniable state. The deniable state is a mapping of a user to a tuple of the form (s, c, K, B, dq), where s represents a seed for a deterministic random number generator, c a counter, K a set of keys, B a set of blocked users, and dq a list of DenIM payloads.*

The notion of validity is lifted to user configurations: strategies of non-adversarial nodes must be valid, written $\mathrm{valid}(\mathcal{U} \mid \mathbf{N})$ which means means Definition 4 holds for all users $\mathcal{U}$ and adversaries $\mathbf{N}$. We assume that all users are initially registered on the server with their respective secret seeds.

5.2.3. System transitions. Figure 2 captures the top-level interaction between users and the server, where each unique arrow type indicates a state transition. It shows how the system is updated when a user sends a message, α. Figure 3 presents selected rules for the upstream user state transitions $\omega; \sigma \overset{\tau, \alpha}{\rightarrowtail} \omega; \sigma'$. Here, τ is the trace of the

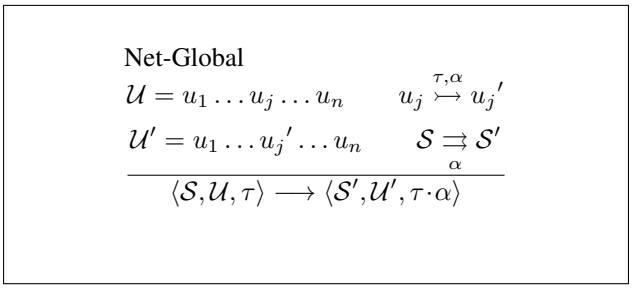


Figure 2: Network transitions, showing how the server, user, and trace state is updated when a message α is sent

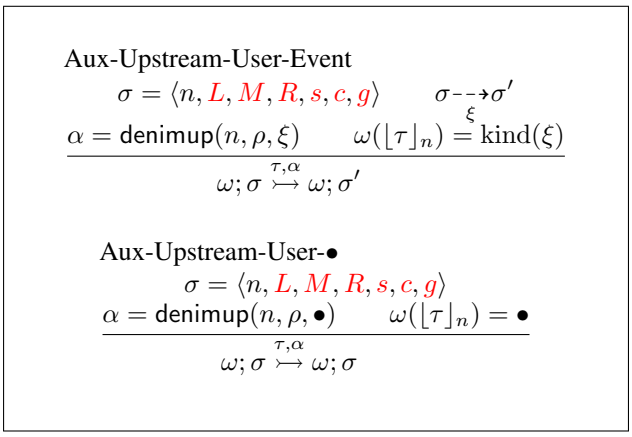


Figure 3: Auxiliary user transitions: selected rules showing how the state of the user strategy ω and local Signal state σ is updated when sending new events or dummy traffic in the message α

system so-far, and α is the new upstream message. The host protocol payload ρ is chosen non-deterministically, while the DenIM event is constrained by the strategy function ω. The subtlety of these rules is that the strategy determines only the kind of the next deniable message, but not their exact content, because the latter depends on the ratchet coordinates. There are two advantages of having the strategies define only the kind of the event and not its full content. First, this keeps the notion of strategy well-formedness simple; it would otherwise have to rule out nonsensical sequences of ratchet coordinates. Second, as a modeling device, it is also conceptually more truthful to capturing the user intent, e.g., the one of sending a message rather than sending a message with particular ratchet indices. Finally, note that we use two rules for the upstream messages. When the user strategy returns dummy, $\bullet$, we do not update the user local Signal state. Note the clause $\sigma \overset{\xi}{\dashrightarrow} \sigma'$ in the rule (Aux-Upstream-User-Event) and its lack in the rule (Aux-Upstream-User-$\bullet$).

Our main theorem states that all valid DenIM strategies are possible. It is formulated as a noninterference theorem on strategies. We write $\simeq_{\mathbf{N}}$ when two configurations or traces are indistinguishable by the adversary.

Theorem 1 (Unobservability of deniable messages in DenIM)**.** *Consider a set of adversary nodes* $\mathbf{N}$*, and two initial indistinguishable configurations* $\langle \mathcal{S}_1, \mathcal{U}_1, \epsilon \rangle \simeq_{\mathbf{N}}^{init} \langle \mathcal{S}_2, \mathcal{U}_2, \epsilon \rangle$*, with valid user strategies, that is* $\mathrm{valid}(\mathcal{U}_i \mid \mathbf{N}), i = 1, 2$*. If* $\langle \mathcal{S}_1, \mathcal{U}_1, \epsilon \rangle \longrightarrow^* \langle \mathcal{S}_1', \mathcal{U}_1', \tau_1 \rangle$ *then* $\langle \mathcal{S}_2, \mathcal{U}_2, \epsilon \rangle \longrightarrow^* \langle \mathcal{S}_2', \mathcal{U}_2', \tau_2 \rangle$*, such that* $\tau_1 \simeq_{\mathbf{N}} \tau_2$*.*

Proof. By induction on the length of τ_1 using Lemma 1 (see Appendix A). □

Note that the guarantee of Theorem 1 allows arbitrary collusion of the adversarial nodes within the set $\mathbf{N}$.

6. Empirical evaluation

To evaluate the feasibility and performance of DenIM, we have created a proof-of-concept implementation of DenIM on Signal and designed experiments to measure the system's behavior when running on a real network.

6.1. Implementation

Our DenIM implementation runs on NodeJS. It amounts to 3838 lines of TypeScript code[1], using the Signal app's open-source implementation [2] of the Signal protocol and their TypeScript bindings (developed for the official Signal Desktop Client). We implement three types of network entities: a dispatcher server that orchestrates the experiments and injects code in the clients to simulate different user behaviors, a DenIM server which handles messages and also acts as a KDC, and DenIM clients.

6.1.1. Network messages. Crucial to achieving DenIM's unobservability of deniable traffic is producing network traffic that does two things: 1) ensures that dummy padding is indistinguishable from encrypted deniable payloads, and 2) ensures that any given padding size can be precisely achieved. To achieve 1), we use TLS sockets for client-server communication, and to achieve 2) we have designed own message formats to ensure that padding serializes to the right size.

To represent and serialize objects, we use Google's protocol buffers [56] – a language and platform-agnostic mechanism for serializing structured data that is also used by the official Signal implementation. All communication is packaged within a `DenimMessage` structure (see Listing 1) before being serialized and sent over an encrypted TLS connection. Regular communication is stored within `RegularPayload` structure, the ratio of deniable padding to use is communicated by the server using `q`, and `c` is used by the server to keep the client's and server's deniable ephemeral key generator in sync. The deniable communication is chopped up to fit within the allotted deniable padding, and stored in the field `chunks` on the `DenimChunk` structure (Listing 2). Since the deniable padding needs to be a fixed size and Google's protocol buffers use varint encoding, we use two additional ballast fields to be able to vary the length of the serialized object by one byte.

```
1 message DenimMessage {
2   required RegularPayload payload
3   optional double q  //server -> client
4   optional int32 c //server -> client
5   repeated DenimChunk chunks
```

1. https://github.com/Niteo/denim-on-signal
2. https://github.com/signalapp/libsignal

```
6   required int32 ballast;
7   optional int32 extra_ballast;
8  }
```

Listing 1: Network message serialization format.

```
1  message DenimChunk {
2      required bytes chunk
3      required int32 flags
4  }
```

Listing 2: Chunk serialization format.

6.1.2. Randomized key labels. In the implementation of Signal, each ephemeral key is labeled with an identifier to allow for quick matching of keys. The number of sessions a user has is not secret in regular Signal, so this identifier may be assigned sequentially. In DenIM, however, assigning identifiers sequentially would leak to anyone requesting ephemeral keys from the KDC how many ephemeral keys the client has generated. To mitigate this leakage in DenIM, we assign random identifiers to deniable ephemeral keys. Since the KDC can generate deniable ephemeral keys as well, the client and KDC stays in sync by providing a seed to generate a deterministic sequence of identifiers when registering with the KDC. Whenever the KDC sends a counter value higher than the client's current counter value, the client generates the corresponding amount of deniable ephemeral keys and identifiers using the two deterministic generators. To avoid identifier collisions, the output space is segmented into identifiers that can be used solely by the client, and solely by the server.

6.2. Experimental setup and design

Each experiment run consists of one DenIM server running on a dedicated machine, multiple clients evenly spread across four machines, and one dispatcher server. The DenIM server is a remote machine with a dedicated 8 core 2.50GHz CPU and 64GB RAM running Ubuntu, and the clients are virtual machines running Ubuntu on 4 core 2.49GHz CPUs with 8GB RAM. The dispatcher server is a virtual machine running Ubuntu on 4 core 2.29GHz CPUs with 8GB RAM.

We have designed our experiments to evaluate the DenIM's performance at high CPU loads, in the range of 80%-90% CPU utilization. Using the CPU utilization goal, we have tuned the number of clients to 20, and the client events to occur every tick of 20 ms. Every experiment run presented here is 60 seconds – as we have seen no empirical changes in the system behavior in longer runs, up to 30 minutes. Each client establishes a TCP connection with the DenIM server and performs the following steps:

1) Registers the user.
2) Every 20 ms (tick), the clients does the following:
   a) sends a given amount (See Tables 3 and 4) of regular messages, to randomly chosen recipients.
   b) sends a given amount (See Tables 3 and 4) of deniable messages, to randomly chosen recipients.

Each message consists of a 52 characters long delimited timestamp to measure latency, and a randomly chosen quote of mean length 57 characters, making messages 109 characters long on average.

To evaluate the performance impact of deniable traffic, we vary the variable q (experiment settings in Table 3) that determines the amount of deniable traffic carried by regular traffic, and the proportion of deniable messages generated in relation to regular messages (experiment settings in Table 4). We empirically establish that the size of a deniable message is approximately 1.2 the size of a regular message with the same plaintext, so when when increasing the ratio with 0.1 we vary q using steps of 0.12.

Id	q	Regular msgs/tick	Deniable msgs/tick
A1	0	10	0
A2	0.12	10	1
A3	0.24	10	2
A4	0.36	10	3
A5	0.48	10	4
A6	0.6	10	5
A7	0.72	10	6
A8	0.84	10	7
A9	0.96	10	8
A10	1.08	10	9
A11	1.2	10	10

TABLE 3: Experiment settings with q set in proportion to the regular and deniable messages ratio.

Id	q	Regular msgs/s	Deniable msgs/s
A6X1	0.6	10	1
A6	0.6	10	5
A6X2	0.6	10	10

TABLE 4: Experiment settings to capture behavior when q is not set in proportion to the regular and deniable messages ratio.

During the experiments, we measure CPU utilization, regular message throughput per second, and the length of the deniable buffers server-side. Client-side, we measure end-to-end latency for regular and deniable messages.

6.3. Results

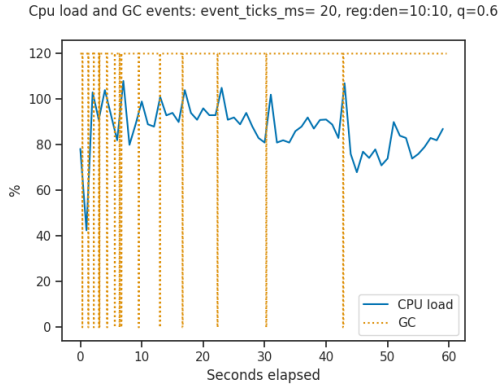
The results for all settings from Table 3 (for all graphs see Appendix C) show, as expected, that regular message throughput is higher for lower values of q, which means lower q gives lower regular message latency. We show the mean throughput for regular and deniable messages for each experiment setting in Table 5. For comparison, CoverDrop [57] empirically has a throughput of 833 (analogous to our deniable) messages per second. Notice that at $q = 1.2$ in setting A11, the throughput for regular and deniable traffic is similar which we expect due to the additional overhead in deniable messages compared to regular messages.

In Figure 4 we show the server's CPU utilization and the regular message throughput over time for setting A6 where $q = 0.6$ and the regular to deniable message ratio is 10:5. The shape of the regular message throughput follows the CPU utilization, with an average of 6544 messages/s which is approximately 392k messages/min. As we can see, the mean CPU utilization is 87%, and the spikes in CPU utilization are correlated with the NodeJS major garbage collection events. Note that NodeJS's garbage collection is concurrent, whereas NodeJS's runtime is single-threaded;

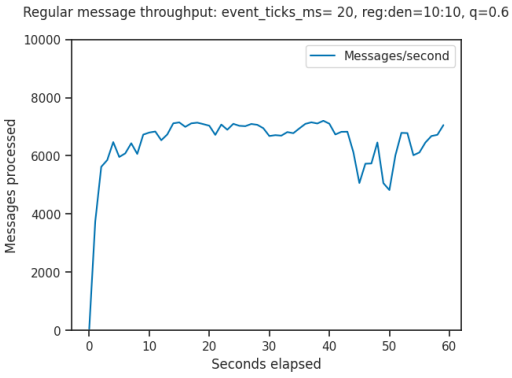
Id	Regular msgs			Deniable msgs		
	s	per s	per min	s	per s	per min
A1	0.014	7527	452k	-	0	0
A2	0.021	7299	438k	8.880	549	33k
A3	0.018	7111	427k	2.370	1352	81k
A4	0.024	6888	413k	0.366	2065	124k
A5	0.026	6599	396k	0.177	2638	159k
A6	0.028	6544	393k	0.140	3270	196k
A7	0.030	6338	380k	0.124	3801	228k
A8	0.032	6163	370k	0.0114	4312	259k
A9	0.034	5998	360k	0.0112	4796	288k
A10	0.036	5955	357k	0.111	5356	321k
A11	0.038	5709	343k	0.108	5706	342k

TABLE 5: The mean overhead for latency, throughput per second, and throughput per minute, for regular and deniable messages respectively

this can result in CPU loads greater than 100% when garbage collection is running. CPU load drops initially due to the clients waiting for key responses before they can encrypt messages.



(a) CPU load, note that spikes correlate with major garbage collection (GC) events.



(b) Regular messages processed over time. Higher is better.

Figure 4: Server statistics at $q = 0.6$.

In Figure 5 we show a box plot of the latency for regular messages measured in seconds. The latency increases when the value of q increases, as the server spends more time chunking and reassembling deniable payloads. As a baseline, we have included A1 where $q = 0$, which means no deniable traffic. The mean latency for A1 is 0.014s/message, and 0.038s/message for A11 where $q = 1.2$.

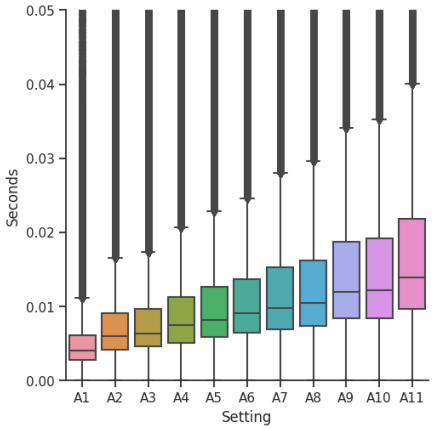


Figure 5: Client to client message regular latency measure from $q = 0$ to $q = 1.2$ with step 0.12. Lower is better.

For settings with a value of q that corresponds to 1.2 times the deniable to regular message ratio, e.g. $q = 0.6$ when the ratio is 10:5, the deniable buffers get drained and filled at a similar rate after approximately 10 seconds (see graphs in Appendix C). When the deniable padding is a relatively small number (recall that the average message plaintext is 109 characters) in comparison to the size of the DenIM headers, the deniable buffers can continue to grow on the server even when q is set in proportion to the deniable to regular message ratio. In our experiments, this happens at $q < 0.36$, i.e. for setting A2 and A3.

When the deniable buffers grow in size on the server, it contributes to large latency for deniable messages. We can see this for A2 and A3 in Figure 27b in Appendix C, with a mean latency of 8.9s for A2, and 2.4s for A3. In Figure 6 we have adjusted the scale to capture the settings with $q \geq 0.36$, i.e. settings A4 to A11. For A4 to A7 we see latency decreasing, and remaining stable from A8 to A11 with a mean latency of 0.11s/message.

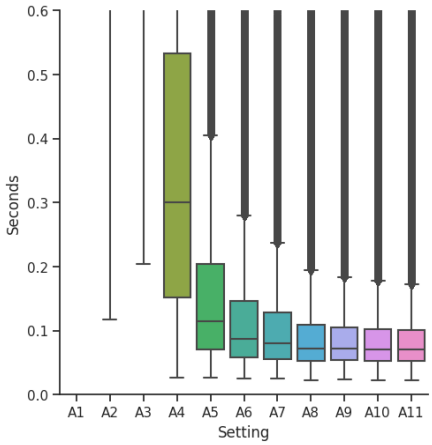


Figure 6: Client to client message latency deniable measure from $q = 0$ to $q = 1.2$ with step 0.12. Lower is better. Notice change in scale.

Notice that each client has their own deniable buffer on the server, so clients' latency for receiving deniable messages depends on how many regular messages they receive in proportion to deniable messages, and how that

proportion relates to the value of q. We use the settings from Table 4 and show in Figure 7 that a higher proportion of deniable messages to regular messages than supported by q (10:10 which would be supported by $q = 1.2$) results in growing deniable buffers, and increased latency for deniable messages. In the same graph, we show that a lower proportion (10:1 which would be supported by $q = 0.12$) allows the server to drain the deniable buffer, and while it lowers latency, it does not increase the throughput of deniable messages since there are no deniable messages for the server to process. A similar fill and drain rate of the deniable buffers is achieved when q is set to reflect the difference in regular to deniable message ratio, in this case 10:5 using setting A6, and as previously shown with the experiment settings A1 to A11.

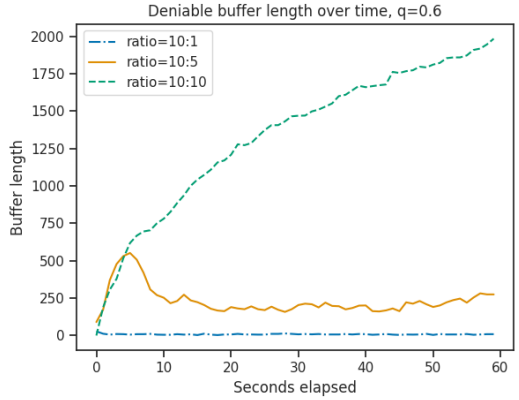


Figure 7: Length of deniable buffers when varying regular to deniable message ratio at $q = 0.6$.

7. Limitations and future work

This section discusses some current limitations of DenIM and provides our perspective on how they may lifted. We also elaborate on how the approach to designing DenIM on Signal could be generalized to a methodology for designing data-aware tunneling protocols.

Latency of deniable messages and practical trade-offs. A conscious design decision behind DenIM on Signal is to prioritize strong privacy over low latency. This naturally affects quality of service, and is a design decision we have taken to reduce the performance overhead of introducing deniable messages to IM services. A consequence of not prioritizing low latency is that a deniable message could remain undelivered, if the recipient has no incoming regular traffic. In other words, the currently chosen trade-off for DenIM on Signal supports delivery of deniable messages for users that *also* send and receive regular messages, and is not intended for users who *only* want to send deniable messages. We stress that users always are guaranteed privacy for their deniable messages in our system, whereas we provide no formal guarantees for utility.

It is possible to alter the trade-off and reduce latency by introducing additional bandwidth overhead. For example, we envision that a practical deployment of DenIM on Signal could feature a subscription feed, a group chat, or a Telegram-like channel, all of which would facilitate a steady stream of regular traffic to contribute to draining the deniable buffers server-side. Along the same lines, a realistic client can periodically send out heartbeats or statistical information to the server that can be used to push deniable messages out of the server.

Support for deniable group chat and deniable video. In theory, our methodology should also be applicable to group chats using the Signal protocol, but group chats would require a slightly different initialization as group chat needs to include all participants' keys. Currently, DenIM on Signal does not support deniable group chats. We also do not support deniable video chats as they come with different requirements on performance: video chats have low tolerance for latency, and the current trade-off in DenIM on Signal cannot guarantee low latency for deniable traffic.

Security of the deniable message buffers. The forwarding server's deniable message buffer can turn into an attractive attack target. We note that the messages in the buffer are end-to-end encrypted, and only include the recipient information – there is no need for the server to hold on to the sender information. Still, if the server were to be compromised, an attacker would gain access to all metadata currently held by the server. To further harden the system, one could for example use trusted enclaves to protect the state of the buffer, as proposed by Ahmed-Rengers et al. in CoverDrop [57].

Side-channels. DenIM on Signal is designed to handle certain potential timing leaks, but not all as they are outside the scope of this paper. For example, deserializing and queueing a deniable message would take longer time than inspecting and throwing away dummy padding. The order when forwarding messages is therefore important in DenIM on Signal: outgoing messages need to be padded with a deniable event (that may be dummy) and forwarded before the incoming deniable event can be processed and queued (unless it is a dummy event), as it would otherwise introduce a timing side-channel. Full protection against side-channel attacks require protection mechanisms that pierce through the entire stack from compiler, to runtime, through the operating system and lastly to the CPU, focusing on many instances of the *layer-below* attacks [58] and all shared resources [59]. We envision that future work could focus on eliminating side-channels by using e.g., predictive mitigation [60], [61], [62], with techniques that require compiler support [63] and runtimes [64], [65], [66], [67] and novel hardware [68].

Communicating with the adversary. While our current assumption of users not communicating with the adversary aligns with the threat model of IM apps – e.g., Signal and WhatsApp both warn users from communicating with untrusted parties rather than enforce the assumption – future work may focus on lifting this assumption. Because this will break noninterference, the top-level security condition will need to be weakened. Here, we anticipate that either techniques from quantitative information flow [69] or declassification [70], [71], [72] can provide useful security characterizations.

Utility guarantees. Our current system provides formal guarantees for privacy, but not utility. Future work could focus either on formally modeling user behavior to provide

utility guarantees, or on extending the empirical evaluation to include real user data. Acquiring real IM data is difficult since it is highly sensitive data that is not publicly available – previous work such as Bahramali et al. [32] have instead collected (but not released) data from alternate sources, namely public Telegram channels. While such alternate data is possible to collect, although not from Signal as Signal does not support public channels, we expect public channel chats to significantly differ from our intended use case which is one-on-one chats. In particular, channels allow a small group of admins broadcast messages to a large crowd, whereas group chats or one-on-one chats puts no restrictions on who can send messages. As such, the collection of quality data from IM usage would be a valuable, non trivial contribution for future work.

Parameter tuning. The only parameter in our design that needs tuning in a real-world deployment is the padding, q. We stress that this parameter should be global and not be personalized, as individual choice of q can be discriminating [32].

Generalizing the methodology. While this paper focuses on building a specific system for IM, DenIM on Signal, we anticipate the approach to translate into a generalized methodology to use IFC techniques for anonymous communication problems.

We believe that a form of noninterference is a desirable baseline property even when systems do not use an observable tunnel (in our case unmodified Signal). As our example attack in Section 5.2.1 shows that naïvely running a protocol on top of an unobservable tunnel, for example DC-nets, can still result in leakage on the application layer. Future work should investigate secure composition of guarantees from different layers.

8. Related work

Anonymous communication. We review the related literature through the categorization in Table 6. This categorization is based on five dimensions. First, *tunneling*: "does the approach tunnel traffic using an innocuous protocol?". Second, *censorship resilience*: "is the design intended to be difficult to detect and block?". Third, *provable guarantees*: "is privacy formally proven, and in that case, on what level of the network stack are the security guarantees?". Fourth, *trust*: "what part of the network is considered trusted?". And last, *threat model*: "what are the capabilities of the adversary?".

Related work fall into four main clusters. The first cluster provide provable guarantees on the transport layer, achieved mainly using DC-nets and mixes. None of these approaches consider application layers, and therefore the privacy guarantees do not extend to e.g., shared protocol states.

The second cluster is similar to the first, but instead the works here give probabilistic privacy guarantees, which allows them to achieve either lower latency or bandwidth but not both. This limitation is in line with the anonymity trilemma of Das et al. [100]. Most of the work from the second cluster offer probabilistic guarantees via differential privacy [101], which makes meaningful longitudinal privacy challenging as privacy degrades with each iteration.

The third clusters include designs focusing on low latency, using mixnets and onion routing. None of these offer any provable privacy guarantees, and e.g. Tor [94] has been shown to be vulnerable to many deanonymization attacks [18].

The last cluster consists mainly of cover protocols and steganography techniques. Most of these work have strong trust assumptions, and weaker adversaries than the work that does not offer strong censorship resilience. Unfortunately, none of them provide any provable privacy guarantees on the traffic shape of the protocol acting as the tunnel. In contrast, the work in this paper maintains the traffic shape of the tunneling ('regular') protocol by design, which we prove in Theorem 1.

DenIM is the only work that combines both strong censorship resilience with provable privacy guarantees. Moreover, this work is the only that models the information flows on the application layer when constructing the proof. With our instantiation DenIM, the simplicity of the design comes at the cost of a centralized server. The closest work is by Howes IV et al. [102], which proposes a framework to formalize and analyze tunneled traffic on the transport layer, to protect against global passive adversaries.

Relation to work on the Signal protocol. Initial work related to Signal included formalizing the protocol in terms of cryptographic primitives and proving its security under minimal trust assumptions [4], [103], [104], [105]. Over time the attention has shifted towards other aspects as well, including implicit versus robust authentication [106], [107]; and offline cryptographic deniability [108], [109], [110]. The latter line of research is closest to this work. The state-of-the-art on this matter is that, from the cryptographic perspective, the Signal protocol only provides offline deniability [109] (transcripts provide no evidence even if long-term key material is compromised); but no online deniability [108] (outsiders can obtain evidence of communication). We bypass this impossibility result by assuming receivers of deniable messages to be honest, and making deniable messages unobservable at a network level, thus strengthening the deniability claims achieved by DenIM in comparison to standard Signal.

Strategies. Our use of strategies for modeling privacy is inspired by their application in the literature on information flow for concurrent programs [55], [111]. For deterministic settings, it is not necessary to involve strategies [112], and simpler stream model of user behavior [71] is sufficient.

9. Conclusion

This work introduces DenIM on Signal, an instant messaging system that provides different privacy guarantees for messages in the same system: either with or without metadata privacy We show that it is possible to design and run a state-of-the-art, stateful protocol that guarantees provable metadata privacy. Specifically, we design a variant of the Signal protocol, *Deniable Instant Messaging* (DenIM), and show that subtle changes – all caught by

Protocol	Tunneling	Censorship resilience	Provable guarantees	Trust	Threat model
DC-nets [73]	No	Weak	Transport layer	Anytrust	GA
Dissent [27]	No	Weak	Transport layer	Anytrust	GA
Anonycaster [74]	No	Weak	Transport layer	Anytrust	GA
Riffle [75]	No	Weak	Transport layer	Anytrust	GA
Atom [76]	No	Weak	Transport layer	Anytrust	GA
Talek [77]	No	Weak	Transport layer	Anytrust	GA
Herd [78]	No	Weak	Transport layer	Chosen set	GP/LA
Pynchon [79]	No	Weak	Transport layer	Fraction	GA
XRD [80]	No	Weak	Transport layer	Fraction	GA
Express [81]	No	Weak	Transport layer	Fraction	GA
P^3 [82]	No	Weak	Transport layer	Honest-but-curious/Malicious	GA
Loopix [83]	No	Weak	Transport layer	Honest-but-curious	GA
Bitmessage [84]	No	Weak	Transport layer	Zero-trust	GA
Pung [85]	No	Weak	Transport layer	Zero-trust	GA
Riposte [86]	No	Weak	Transport layer	Zero-trust***	GA
Verdict [87]	No	Weak	Transport layer*	Anytrust	GA
Stadium [88]	No	Weak	Transport layer*	Anytrust	GA
Vuvuzela [28]	No	Weak	Transport layer*	Fraction	GA
Alpenhorn [89]	No	Weak	Transport layer*	Fraction	GA
Karaoke [29]	No	Weak	Transport layer*	Fraction	GP
Yodel [90]	No	Weak	Transport layer**	Fraction	GA
Groove [91]	No	Weak	Transport layer*	Zero-trust	GA
Mixminion [92]	No	Weak	–	Chosen path	GA
HORNET [93]	No	Weak	–	Fraction	LA
Tor [94]	No	Weak	–	One****	GP
DenIM	Yes	Strong	Application layer	Centralized	GA
CoverDrop [57]	Yes	Strong	–	Centralized	GP
Cirripede [95]	Yes	Strong	–	Proxies	LA
Telex [96]	Yes	Strong	–	Proxies	LA
CensorSpoofer [97]	Yes	Strong	–	Proxies	LA
FreeWave [35]	Yes	Strong	–	Proxies	LA
SkypeMorph [98]	Yes	Strong	–	Proxy	LA
IMProxy [32]	Yes	Strong	–	Proxy	LA
Protozoa [36]	Yes	Strong	–	Proxy	LA
Camoufler[37]	Yes	Strong	–	Proxy	LA
Balboa [99]	Yes	Strong	–	Zero-trust	GA

TABLE 6: Comparison of related work. Footnotes: *via differential privacy, **with failure probability 10^{-8} per round, ***for privacy, all servers need to be trusted for availability, ****in chosen set. G=Global, L=Local, A=Active, P=Passive.

a formal model – to the original protocol are necessary for privacy. Through our proof-of-concept implementation DenIM on Signal we showcase how to strike performance trade-offs for real-world applications – the overhead of deniable traffic in DenIM is parameterizable through a variable q, and we empirically show the behavior under different traffic loads.

Acknowledgments

We thank the anonymous reviewers for their valuable suggestions for improving the paper. This work was funded by the Danish Council Independent Research for the Natural Sciences (DFF/FNU, project 6108-00363). Boel Nelson was funded by the MSCA European Postdoctoral Fellowships project 101064140 (ProPriM).

References

[1] WhatsApp, "WhatsApp Encryption Overview Technical white paper," Oct. 2020. [Online]. Available: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf

[2] Wire Swiss GmbH, "Wire Security Whitepaper," 2021.

[3] Facebook Newsroom, "Messenger Starts Testing End-to-End Encryption with Secret Conversations," Jul. 2016. [Online]. Available: https://about.fb.com/news/2016/07/messenger-starts-testing-end-to-end-encryption-with-secret-conversations/

[4] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *Journal of Cryptology*, vol. 33, no. 4, pp. 1914–1983, 2020.

[5] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service Usage Classification with Encrypted Internet Traffic in Mobile Messaging Apps," *IEEE Transactions on Mobile Computing*, vol. 15, no. 11, 2016.

[6] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust Smartphone App Identification via Encrypted Network Traffic Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, 2018.

[7] R. S. Raman, A. Stoll, J. Dalek, R. Ramesh, W. Scott, and R. Ensafi, "Measuring the Deployment of Network Censorship Filters at Global Scale," in *NDSS*, 2020.

[8] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining How the Great Firewall Discovers Hidden Circumvention Servers," in *IMC*, 2015.

[9] Johns Hopkins University, "The Johns Hopkins Foreign Affairs Symposium Presents: The Price of Privacy: Re-Evaluating the NSA," Apr. 2014. [Online]. Available: https://www.youtube.com/watch?v=kV2HDM86XgI

[10] R. Kang, L. Dabbish, N. Fruchter, and S. Kiesler, ""My data just goes everywhere": User mental models of the internet and implications for privacy and security," in *SOUPS*, 2015.

[11] P. Story, D. Smullen, Y. Yao, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub, "Awareness, Adoption, and Misconceptions of Web Privacy Tools," *PoPETS*, vol. 2021, no. 3, 2021.

[12] N. Gerber, V. Zimmermann, and M. Volkamer, "Why Johnny Fails to Protect his Privacy," in *EuroUSEC*, 2019.

[13] G. Norcie, J. Blythe, K. Caine, and L. J. Camp, "Why Johnny Can't Blow the Whistle: Identifying and Reducing Usability Issues in Anonymity Systems," in *Proceedings 2014 Workshop on Usable Security*. San Diego, CA: Internet Society, 2014.

[14] S. Samuel, "China is installing a secret surveillance app on tourists' phones," Jul. 2019. [Online]. Available: https://www.vox.com/future-perfect/2019/7/3/20681258/china-uighur-surveillance-app-tourist-phone

[15] K. Wagstaff, "Failing grade: Alleged Harvard bomb hoaxer needed more than Tor to cover his tracks, experts say," Dec. 2013. [Online]. Available: http://www.nbcnews.com/technolog/failing-grade-alleged-harvard-bomb-hoaxer-needed-more-tor-cover-2D11767028

[16] Tor Project, "The Tor Project — Privacy & Freedom Online," 2021. [Online]. Available: https://torproject.org

[17] The Tor Project, "Users – Tor Metrics," 2021. [Online]. Available: https://metrics.torproject.org/userstats-relay-country.html

[18] I. Karunanayake, N. Ahmed, R. Malaney, R. Islam, and S. K. Jha, "De-anonymisation attacks on Tor: A Survey," *IEEE Communications Surveys Tutorials*, 2021.

[19] R. Jansen, T. Vaidya, and M. Sherr, "Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor," in *USENIX Security*, 2019.

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

[21] "How it works." [Online]. Available: https://www.zbay.app/how.html

[22] "How it works - Briar." [Online]. Available: https://briarproject.org/how-it-works/

[23] "Overview and History - Cwtch Secure Development Handbook." [Online]. Available: https://docs.openprivacy.ca/cwtch-security-handbook/overview.html

[24] "Ricochet Refresh." [Online]. Available: https://www.ricochetrefresh.net/

[25] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The Parrot Is Dead: Observing Unobservable Network Communications," in *S&P*, 2013.

[26] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in Numbers: Making Strong Anonymity Scale," in *OSDI*, 2012.

[27] H. Corrigan-Gibbs and B. Ford, "Dissent: Accountable anonymous group messaging," in *CCS*, 2010.

[28] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *SOSP*, 2015.

[29] D. Lazar, Y. Gilad, and N. Zeldovich, "Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis," in *OSDI*, 2018.

[30] Y. Gilad, "Metadata-private communication for the 99%," *Communications of the ACM*, vol. 62, no. 9, 2019.

[31] Electronic Frontier Foundation, "Communicating with Others," 2020. [Online]. Available: https://ssd.eff.org/en/module/communicating-others

[32] A. Bahramali, A. Houmansadr, R. Soltani, D. Goeckel, and D. Towsley, "Practical Traffic Analysis Attacks on Secure Messaging Applications," in *NDSS*, 2020.

[33] T. Perrin, "The Noise Protocol Framework," 2018. [Online]. Available: https://noiseprotocol.org/noise.pdf

[34] E. Zuckerman, "Cute Cats to the Rescue? Participatory Media and Political Expression," in *From Voice to Influence: Understanding Citizenship in a Digital Age*. University of Chicago Press, 2015.

[35] A. Houmansadr, T. Riedl, N. Borisov, and A. Singer, "I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention," in *NDSS*, 2013.

[36] D. Barradas, N. Santos, L. Rodrigues, and V. Nunes, "Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC," in *CCS*, 2020.

[37] P. K. Sharma, D. Gosain, and S. Chakravarty, "Camoufler: Accessing The Censored Web By Utilizing Instant Messaging Channels," in *ASIA CCS*, 2021.

[38] K. O'Neill, M. Clarkson, and S. Chong, "Information-flow security for interactive programs," in *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006, pp. 12 pp.–201.

[39] The Radicati Group, inc., "Instant Messaging Statistics Report, 2019-2023," 2019. [Online]. Available: https://radicati.com/wp/wp-content/uploads/2019/01/Instant_Messaging_Statistics_Report,_2019-2023_Exceutive_Summary.pdf

[40] Statista, "Most popular global mobile messenger apps as of October 2021, based on number of monthly active users," Nov. 2021. [Online]. Available: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/

[41] J. Kastrenakes, "Apple says there are now over 1 billion active iPhones," Jan. 2021. [Online]. Available: https://www.theverge.com/2021/1/27/22253162/iphone-users-total-number-billion-apple-tim-cook-q1-2021

[42] C. Jenik, "Here's what happens every minute on the internet in 2021," Aug. 2021. [Online]. Available: https://www.weforum.org/agenda/2021/08/one-minute-internet-web-social-media-technology-online/

[43] iXsystems, Inc., "Rick Reed - WhatsApp: Half a billion unsuspecting FreeBSD users," MeetBSD California, Nov. 2014. [Online]. Available: https://www.youtube.com/watch?v=TneLO5TdW_M

[44] Telegram, "MTProto Mobile Protocol." [Online]. Available: https://core.telegram.org/mtproto

[45] Apple inc., "How iMessage sends and receives messages securely," Feb. 2021. [Online]. Available: https://support.apple.com/guide/security/how-imessage-sends-and-receives-messages-sec70e68c949/web

[46] S. Salim, "Finally: Snapchat comes up with end-to-end encryption to secure users conversations and data," Jan. 2019. [Online]. Available: https://www.digitalinformationworld.com/2019/01/snapchat-end-to-end-encryption-users-media-messages.html

[47] M. Marlinspike, "Advanced cryptographic ratcheting," 2013. [Online]. Available: https://signal.org/blog/advanced-ratcheting/

[48] ——, "The X3DH Key Agreement Protocol," 2016. [Online]. Available: https://signal.org/docs/specifications/x3dh/x3dh.pdf

[49] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use PGP," in *WPES*, 2004.

[50] V. Moscaritolo, G. Belvin, and P. Zimmermann, "Silent Circle Instant Messaging Protocol Protocol Specification," 2012. [Online]. Available: https://netzpolitik.org/wp-upload/SCIMP-paper.pdf

[51] M. Marlinspike, "The Double Ratchet Algorithm," Nov. 2016. [Online]. Available: https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

[52] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky, "Deniable Encryption," in *CRYPTO*, 1997.

[53] J. A. Goguen and J. Meseguer, "Security policies and security models," in *S&P*. IEEE, 1982.

[54] M. Abadi, "Secrecy by typing in security protocols," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 749–786, 1999.

[55] K. R. O'Neill, M. R. Clarkson, and S. Chong, "Information-flow security for interactive programs," in *19th IEEE Computer Security Foundations Workshop (CSFW'06)*. IEEE, 2006, pp. 12–pp.

[56] Google Developers, "Encoding | Protocol Buffers | Google Developers," May 2022. [Online]. Available: https://developers.google.com/protocol-buffers/docs/encoding

[57] M. Ahmed-Rengers, D. A. Vasile, D. Hugenroth, A. R. Beresford, and R. Anderson, "CoverDrop: Blowing the Whistle Through A News App," *PoPETS*, 2022.

[58] F. Piessens, "Security across abstraction layers: old and new examples," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 271–279.

[59] R. A. Kemmerer, "Shared resource matrix methodology: An approach to identifying storage and timing channels," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 3, pp. 256–277, 1983.

[60] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 297–307. [Online]. Available: https://doi.org/10.1145/1866307.1866341

[61] D. Zhang, A. Askarov, and A. C. Myers, "Predictive mitigation of timing channels in interactive systems," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 563–574. [Online]. Available: https://doi.org/10.1145/2046707.2046772

[62] ——, "Language-based control and mitigation of timing channels," *SIGPLAN Not.*, vol. 47, no. 6, p. 99–110, jun 2012. [Online]. Available: https://doi.org/10.1145/2345156.2254078

[63] L. Simon, D. Chisnall, and R. Anderson, "What you get is what you c: Controlling side effects in mainstream c compilers," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 1–15.

[64] M. V. Pedersen and A. Askarov, "From trash to treasure: Timing-sensitive garbage collection," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 693–709.

[65] ——, "Static enforcement of security in runtime systems," in *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*. IEEE, 2019, pp. 335–33 515.

[66] P. Buiras and A. Russo, "Lazy programs leak secrets," in *Nordic Conference on Secure IT Systems*. Springer, 2013, pp. 116–122.

[67] T. Brennan, N. Rosner, and T. Bultan, "Jit leaks: Inducing timing side channels through just-in-time compilation," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1207–1222.

[68] A. Ferraiuolo, M. Zhao, A. C. Myers, and G. E. Suh, "Hyperflow: A processor architecture for nonmalleable, timing-safe information flow security," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1583–1600.

[69] M. S. Alvim, K. Chatzikokolakis, A. McIver, C. Morgan, C. Palamidessi, and G. Smith, *The Science of Quantitative Information Flow*, ser. Information Security and Cryptography. Cham: Springer International Publishing, 2020.

[70] A. Sabelfeld and D. Sands, "Dimensions and principles of declassification," in *18th IEEE Computer Security Foundations Workshop (CSFW'05)*, Jun. 2005, pp. 255–269.

[71] A. Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007, pp. 207–221.

[72] ——, "Tight enforcement of information-release policies for dynamic languages," in *2009 22nd IEEE Computer Security Foundations Symposium*, 2009, pp. 43–59.

[73] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, 1988.

[74] C. C. D. Head, "Anonycaster: Simple, Efficient Anonymous Group Communication," 2012. [Online]. Available: https://blogs.ubc.ca/computersecurity/files/2012/04/anonycaster.pdf

[75] A. Kwon, D. Lazar, S. Devadas, and B. Ford, "Riffle: An Efficient Communication System With Strong Anonymity," *PoPETS*, 2016.

[76] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally Scaling Strong Anonymity," in *SOSP*, 2017.

[77] R. Cheng, W. Scott, E. Masserova, I. Zhang, V. Goyal, T. Anderson, A. Krishnamurthy, and B. Parno, "Talek: Private Group Messaging with Hidden Access Patterns," in *ASAC*, 2020.

[78] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt, "Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems," in *SIGCOMM*, 2015.

[79] L. Sassaman, B. Cohen, and N. Mathewson, "The pynchon gate: a secure method of pseudonymous mail retrieval," in *WPES*, 2005.

[80] A. Kwon, D. Lu, and S. Devadas, "XRD: Scalable Messaging System with Cryptographic Privacy," in *NSDI*, 2020, pp. 759–776.

[81] S. Eskandarian, H. Corrigan-Gibbs, M. Zaharia, and D. Boneh, "Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy," in *USENIX Security*, 2021.

[82] L. Kissner, A. Oprea, M. K. Reiter, D. Song, and K. Yang, "Private Keyword-Based Push and Pull with Applications to Anonymous Communication," in *ACNS*, 2004.

[83] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix Anonymity System," in *USENIX Security*, 2017.

[84] J. Warren, "Bitmessage: A Peer-to-Peer Message Authentication and Delivery System," 2012. [Online]. Available: https://bitmessage.org/bitmessage.pdf

[85] S. Angel and S. Setty, "Unobservable Communication over Fully Untrusted Infrastructure," in *OSDI*, 2016.

[86] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An Anonymous Messaging System Handling Millions of Users," in *S&P*, 2015.

[87] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, "Proactively Accountable Anonymous Messaging in Verdict," in *USENIX Security*, 2013.

[88] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A Distributed Metadata-Private Messaging System," in *SOSP*, 2017.

[89] D. Lazar and N. Zeldovich, "Alpenhorn: Bootstrapping Secure Communication without Leaking Metadata," in *OSDI*, 2016.

[90] D. Lazar, Y. Gilad, and N. Zeldovich, "Yodel: Strong metadata security for voice calls," in *SOSP*, 2019.

[91] L. Barman, M. Kol, D. Lazar, Y. Gilad, and N. Zeldovich, "Groove: Flexible Metadata-Private Messaging," in *OSDI*, 2022.

[92] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: design of a type III anonymous remailer protocol," in *S&P*, 2003.

[93] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: High-speed Onion Routing at the Network Layer," in *CCS*, 2015.

[94] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," Defense Technical Information Center (DTIC), Technical report, 2004.

[95] A. Houmansadr, G. T. Nguyen, M. Caesar, and N. Borisov, "Cirripede: Circumvention infrastructure using router redirection with plausible deniability," in *CCS*, 2011.

[96] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman, "Telex: Anticensorship in the Network Infrastructure," in *USENIX Security*, 2011.

[97] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov, "CensorSpoofer: asymmetric communication using IP spoofing for censorship-resistant web browsing," in *CCS*, 2012.

[98] H. Mohajeri Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph: protocol obfuscation for Tor bridges," in *CCS*, 2012.

[99] M. B. Rosen, J. Parker, and A. J. Malozemoff, "Balboa: Bobbing and Weaving around Network Censorship," in *USENIX Security*, 2021.

[100] D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity Trilemma: Strong Anonymity, Low Bandwidth Overhead, Low Latency - Choose Two," in *S&P*, 2018.

[101] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating Noise to Sensitivity in Private Data Analysis," in *Theory of Cryptography*. Springer Berlin Heidelberg, 2006, no. 3876, pp. 265–284.

[102] J. K. Howes IV, M. Georgiou, A. J. Malozemoff, and T. Shrimpton, "Security Foundations for Application-Based Covert Communication Channels," in *S&P*, 2022.

[103] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati, and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *CRYPTO*, 2017.

[104] J. Jaeger and I. Stepanovs, "Optimal channel security against fine-grained state compromise: The safety of messaging," in *CRYPTO*, 2018.

[105] J. Alwen, S. Coretti, and Y. Dodis, "The double ratchet: security notions, proofs, and modularization for the signal protocol," in *EUROCRYPT*, 2019.

[106] O. Blazy, A. Bossuat, X. Bultel, P.-A. Fouque, C. Onete, and E. Pagnin, "Said: reshaping signal into an identity-based asynchronous messaging protocol with authenticated ratcheting," in *EuroS&P*, 2019.

[107] O. Blazy, P.-A. Fouque, T. Jacques, P. Lafourcade, C. Onete, and L. Robert, "Marshal: Messaging with asynchronous ratchets and signatures for faster healing," in *SAC*, 2022.

[108] N. Unger and I. Goldberg, "Improved strongly deniable authenticated key exchanges for secure messaging," *PoPETS*, vol. 2018, no. 1, pp. 21–66, 2018.

[109] N. Vatandas, R. Gennaro, B. Ithurburn, and H. Krawczyk, "On the cryptographic deniability of the signal protocol," in *ACNS*, 2020.

[110] J. Brendel, R. Fiedler, F. Günther, C. Janson, and D. Stebila, "Post-quantum asynchronous deniable key exchange and the signal handshake," in *PKC*, 2022.

[111] A. Askarov, S. Chong, and H. Mantel, "Hybrid monitors for concurrent noninterference," in *2015 IEEE 28th Computer Security Foundations Symposium*, 2015, pp. 137–151.

[112] D. Clark and S. Hunt, "Non-interference for deterministic interactive programs," in *International Workshop on Formal Aspects in Security and Trust*. Springer, 2008, pp. 50–66.

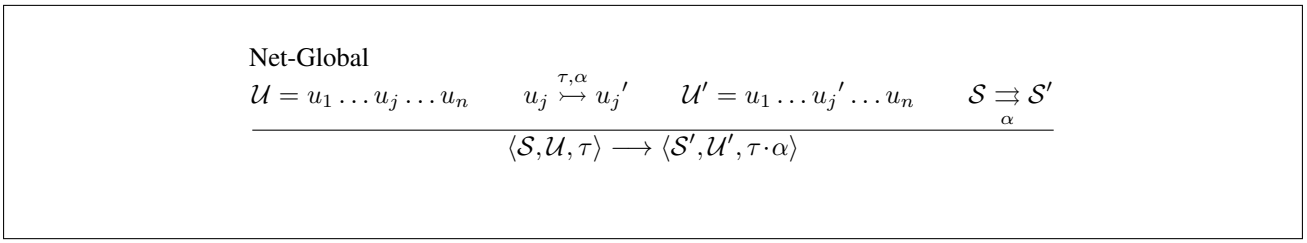


Figure 8: Network transitions

1. Full formal model

To formally model keys generated in the protocol, we introduce the notion of *formal randomness* that captures two kinds of key generation we have in the protocol. Local key generation takes place on the user nodes when they start new sessions or refill keys. Seeded key generation takes place on the server, and later on the client when their seed counters are updated. We define formal randomness via *tags* defined as follows.

$$\mathsf{r} ::= \$\mathsf{local}(n, g) \mid \$\mathsf{seeded}(s, c)$$

We denote the set of formal randomness tags as $\mathbf{FR}$. Keys are represented as values of form $\mathsf{k}\langle\mathsf{r}\rangle$, effectively propagating the randomness tag associated with their creation.

Definition 7 (Regular message)**.**

$$\gamma ::= \mathsf{msg}(n, \rho)$$

Definition 8 (User state)**.** *A user state is a tuple $\langle n, L, M, R, s, c, g \rangle$, where n is the identity of the user, L and M are sets containing the user's own keys or keys of other users paired with a state of the form (k, fresh) or (k, used), R is the abstraction of a ratchets mapped by receiving users to $(w, \{k_1, k_2\}, I)$ containing the starting index assigned to the user, the initiating keys, and the observed message indices I, s is a seed for the deterministic random number generator, c is a formal randomness counter for the server-side key generation, and g is the formal randomness counter for the client-side key generation.*

Definition 9 (Server state)**.** *A server state is represented by a tuple $\langle H, rq, D \rangle$ where H represents the state of an arbitrary host protocol, rq is a list of outgoing regular messages, and D represents the deniable state. The deniable state is a mapping of a user to a tuple of the form (s, c, K, B, dq), where s represents a seed for a deterministic random number generator, c a counter, K a set of keys, B a set of blocked users, and dq a list of DenIM payloads.*

1.1. Network configuration. A network configuration is represented by a tuple containing server state, user states and a network trace: $\langle \mathcal{S}, \mathcal{U}, \tau \rangle$.

1.2. Network and state transitions. Figure 8 Figure 11 Figure 12 Figure 13 Figure 14 Figure 10 Figure 9

User-Denim-Kresp

$$\frac{\xi = \mathsf{kresp}((n_1, k)\ \mathsf{for}\ n) \qquad M' = M[n_1 \mapsto (k, \mathsf{fresh})]}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L, M', R, s, c, g\rangle}$$

User-Denim-Fwd-New-Session

$$\frac{\begin{array}{c}\xi = \mathsf{fwd}(n_1 \to n, tok) \qquad n_1 \notin \mathsf{dom}(R) \qquad tok = \{(n_1, k_1), (n, k), (0, y)\} \\ L = L' \uplus \{(k, \mathsf{fresh})\} \qquad L'' = L' \cup \{(k, \mathsf{used})\} \qquad R' = R[n_1 \mapsto (1, \{k, k_1\}, \{(0, y)\})]\end{array}}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L'', M, R', s, c, g\rangle}$$

User-Denim-Fwd-Initialized

$$\frac{\begin{array}{c}\xi = \mathsf{fwd}(n_1 \to n, tok) \qquad n_1 \in \mathsf{dom}(R) \\ tok = \{(n_1, k_1), (n, k), (x, y)\} \qquad R(n_1) = (w, \{k, k_1\}, I) \qquad R' = R[n_1 \mapsto (w, \{k, k_1\}, I \cup \{(x, y)\})]\end{array}}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L, M, R', s, c, g\rangle}$$

Figure 9: Client downstream state transitions

User-Denim-Refill

$$\frac{k = \mathsf{k}\langle\$\mathsf{local}(n, g)\rangle \qquad \xi = \mathsf{refill}(n, k) \qquad L' = L \cup \{(k, \mathsf{fresh})\}}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L', M, R, s, c, g + 1\rangle}$$

User-Denim-Kreq

$$\frac{\xi = \mathsf{kreq}(n_1\ \mathsf{for}\ n)}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L, M, R, s, c, g\rangle}$$

User-Denim-Block

$$\frac{\xi = \mathsf{block}(n\ \mathsf{by}\ n_1)}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L, M, R, s, c, g\rangle}$$

User-Denim-Send-New-Session

$$\frac{\begin{array}{c}\{n, n_1\} \notin \mathsf{dom}(R) \\ k = \mathsf{k}\langle\$\mathsf{local}(n, g)\rangle \qquad L' = L \cup \{(k, \mathsf{used})\} \qquad M(n_1) = (k_1, \mathsf{fresh}) \qquad M' = M[n_1 \mapsto (k_1, \mathsf{used})] \\ tok = \{(n, k), (n_1, k_1), (0, 0)\} \qquad \xi = \mathsf{send}(n \to n_1, tok) \qquad R' = R[n_1 \mapsto (0, \{k, k_1\}, \{(0, 0)\})]\end{array}}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L', M', R', s, c, g + 1\rangle}$$

User-Denim-Send-Initialized

$$\frac{\begin{array}{c}n_1 \in \mathsf{dom}(R) \qquad (x, y) = \mathsf{next}(R, n_1) \qquad (k, \mathsf{used}) \in L \qquad tok = \{(n, k), (n_1, k_1), (x, y)\} \\ \xi = \mathsf{send}(n \to n_1, tok) \qquad R(n_1) = (w, \{k, k_1\}, I) \qquad R' = R[n \mapsto (w, \{k, k_1\}, I \cup \{(x, y)\})]\end{array}}{\langle n, L, M, R, s, c, g\rangle \xrightarrow{\xi} \langle n, L, M, R', s, c, g\rangle}$$

$$\begin{array}{l}\mathsf{next}(R, n) = \\ \quad x, y = \mathsf{latest}(R, n) \\ \quad w = R(n).w \\ \quad \mathsf{if} x\%2 = w \\ \quad\quad \mathsf{then}\ (x, y + 1) \\ \quad\quad \mathsf{else}\ (x + 1, 0)\end{array}$$

Figure 10: Client upstream state transitions, and auxiliary function next, where *latest* returns the latest index in the ratchet

Aux-Upstream-User-Event

$$\frac{\sigma = \langle n, L, M, R, s, c, g\rangle \qquad \sigma \overset{\xi}{\dashrightarrow} \sigma' \qquad \alpha = \mathsf{denimup}(n, \rho, \xi) \qquad \omega(\lfloor \tau \rfloor_n) = \mathrm{kind}(\xi)}{\omega; \sigma \overset{\tau,\alpha}{\rightarrowtail} \omega; \sigma'}$$

Aux-Upstream-User-•

$$\frac{\sigma = \langle n, L, M, R, s, c, g\rangle \qquad \alpha = \mathsf{denimup}(n, \rho, \bullet) \qquad \omega(\lfloor \tau \rfloor_n) = \bullet}{\omega; \sigma \overset{\tau,\alpha}{\rightarrowtail} \omega; \sigma}$$

Aux-Downstream-User-Event

$$\frac{\sigma = \langle n, L, M, R, s, c, g\rangle \qquad \alpha = \mathsf{denimdn}(n, \rho, \xi, c') \qquad L' = L \cup \mathsf{seededfreshkeys}(s, c, c') \qquad \langle n, L', M, R, s, c', g\rangle \overset{\xi}{\dashrightarrow} \langle n, L', M', R', s, c', g\rangle \qquad \sigma' = \langle n, L', M', R', s, c', g\rangle}{\omega; \sigma \overset{\tau,\alpha}{\rightarrowtail} \omega; \sigma'}$$

Aux-Downstream-User-Dummy-Or-Malformed

$$\frac{\sigma = \langle n, L, M, R, s, c, g\rangle \qquad \alpha = \mathsf{denimdn}(n, \rho, \xi, c') \qquad L' = L \cup \mathsf{seededfreshkeys}(s, c, c') \qquad \sigma' = \langle n, L', M, R, s, c', g\rangle \qquad \sigma' \overset{\xi}{\not\dashrightarrow}}{\omega; \sigma \overset{\tau,\alpha}{\rightarrowtail} \omega; \sigma'}$$

$$\mathsf{seededfreshkeys}(s, c, c') = \bigcup_{c \le j < c'} (\mathsf{k}\langle\$\mathsf{seeded}(s, j)\rangle, \mathsf{fresh})$$

Figure 11: Auxiliary user transitions

Aux-Upstream-Server-Event

$$\frac{\alpha = \mathsf{denimup}(n, \rho, \xi) \qquad \langle H, rq\rangle \xrightarrow{(n,\rho)} \langle H', rq'\rangle \qquad D \overset{\xi}{\rightsquigarrow} D'}{\langle H, rq, D\rangle \overset{\alpha}{\rightrightarrows} \langle H', rq', D'\rangle}$$

Aux-Upstream-Server-Dummy-Or-Malformed

$$\frac{\alpha = \mathsf{denimup}(n, \rho, \xi) \qquad \langle H, rq\rangle \xrightarrow{(n,\rho)} \langle H', rq'\rangle \qquad D \overset{\xi}{\not\rightsquigarrow}}{\langle H, rq, D\rangle \overset{\alpha}{\rightrightarrows} \langle H', rq', D\rangle}$$

Aux-Downstream-Server-Event

$$\frac{D(n) = (s, c, K, B, \xi \cdot dq) \qquad \alpha = \mathsf{denimdn}(n, \rho, \xi, c) \qquad D' = D[n_1 \mapsto (s, c, K, B, dq)]}{\langle H, (n_1, \rho) \cdot rq, D\rangle \overset{\alpha}{\rightrightarrows} \langle H, rq, D'\rangle}$$

Aux-Downstream-Server-•

$$\frac{D(n) = (s, c, K, B, []) \qquad \alpha = \mathsf{denimdn}(n_1, \rho, \bullet, c)}{\langle H, (n, \rho) \cdot rq, D\rangle \overset{\alpha}{\rightrightarrows} \langle H, rq, D\rangle}$$

Figure 12: Auxiliary server transitions

Server-Host-Action

$$\frac{H', \gamma' = \mathsf{hostresponse}(H, \gamma)}{\langle H, rq\rangle \xrightarrow{\gamma} \langle H', rq{\cdot}\gamma'\rangle}$$

Server-Host-No-Action

$$\frac{H', \bot = \mathsf{hostresponse}(H, \gamma)}{\langle H, rq\rangle \xrightarrow{\gamma} \langle H', rq\rangle}$$

Figure 13: Server state transitions processing host protocol messages

Server-Denim-Refill

$$\frac{\xi = \mathsf{refill}(n, k) \qquad D(n) = (s, c, K, B, dq) \qquad K' = K \cup k \qquad D' = D[n \mapsto (s, c, K', B, dq)]}{D \overset{\xi}{\rightsquigarrow} D'}$$

Server-Denim-Kreq

$$\frac{\xi = \mathsf{kreq}(n_1 \text{ for } n_2) \qquad D(n_1) = (s_1, c_1, K_1, B_1, dq_1) \qquad D(n_2) = (s_2, c_2, K_2, B_2, dq_2) \qquad K_1 = K_1' \uplus \{k_1\}}{\xi' = \mathsf{kresp}((n_1, k_1) \text{ for } n_2) \qquad D' = D[n_1 \mapsto (s_1, c_1, K_1', B_1, dq_1), \qquad n_2 \mapsto (s_2, c_2, K_2, B_2, dq_2{\cdot}\xi')]}$$

$$D \overset{\xi}{\rightsquigarrow} D'$$

Server-Denim-Kreq-Out-of-Keys

$$\frac{\xi = \mathsf{kreq}(n_1 \text{ for } n_2) \qquad D(n_1) = (s_1, c_1, \varnothing, B_1, dq_1) \qquad D(n_2) = (s_2, c_2, K_2, B_2, dq_2) \qquad k_1 = \mathsf{k}\langle\$\mathsf{seeded}(s_1, c_1)\rangle}{\xi' = \mathsf{kresp}((n_1, k_1) \text{ for } n_1) \qquad D' = D[n_1 \mapsto (s_1, c_1 + 1, \varnothing, B_1, dq_1), \qquad n_2 \mapsto (s_2, c_2, K_2, B_2, dq_2{\cdot}\xi')]}$$

$$D \overset{\xi}{\rightsquigarrow} D'$$

Server-Denim-Send

$$\frac{\xi = \mathsf{send}(n_1 \to n_2, tok) \qquad D(n_2) = (s, c, K, B, dq) \qquad n_1 \notin \mathsf{dom}(B) \qquad \xi' = \mathsf{fwd}(n_1 \to n_2, tok) \qquad D' = D[n_2 \mapsto (s, c, K, B, dq{\cdot}\xi')]}{D \overset{\xi}{\rightsquigarrow} D'}$$

Server-Denim-Send-Blocked-Or-Unregistered

$$\frac{\xi = \mathsf{send}(n_1 \to n_2, tok) \qquad D(n_2) = (s, c, K, B, dq) \qquad (n_1 \in \mathsf{dom}(B) \vee n_2 \notin \mathsf{dom}(D))}{D \overset{\xi}{\rightsquigarrow} D}$$

Server-Denim-Block

$$\frac{\xi = \mathsf{block}(n_2 \text{ by } n_1) \qquad D(n_1) = (s, c, K, B, dq) \qquad B' = B \cup \{n_2\} \qquad D' = D[n_1 \mapsto (s, c, K, B', dq)]}{D \overset{\xi}{\rightsquigarrow} D'}$$

Figure 14: Server state transitions caused by processing of DenIM payloads

$$\frac{}{\mathsf{block}(n \text{ by } n) \overset{\phi}{\sim} \mathsf{block}(n \text{ by } n)} \qquad \frac{}{\mathsf{kreq}(n \text{ for } n) \overset{\phi}{\sim} \mathsf{kreq}(n \text{ for } n)} \qquad \frac{}{\bullet \overset{\phi}{\sim} \bullet}$$

$$\frac{tok_1 = \{(n_1, \mathsf{k}\langle \mathsf{r}_1 \rangle), (n_2, \mathsf{k}\langle \mathsf{r}_2 \rangle), (x, y)\} \qquad tok_2 = \{(n_1, \mathsf{k}\langle \phi(\mathsf{r}_1) \rangle), (n_2, \mathsf{k}\langle \phi(\mathsf{r}_2) \rangle), (x, y)\}}{\mathsf{send}(n_1 \to n_2, tok_1) \overset{\phi}{\sim} \mathsf{send}(n_1 \to n_2, tok_2)}$$

$$\frac{tok_1 = \{(n_1, \mathsf{k}\langle \mathsf{r}_1 \rangle), (n_2, \mathsf{k}\langle \mathsf{r}_2 \rangle), (x, y)\} \qquad tok_2 = \{(n_1, \mathsf{k}\langle \phi(\mathsf{r}_1) \rangle), (n_2, \mathsf{k}\langle \phi(\mathsf{r}_2) \rangle), (x, y)\}}{\mathsf{fwd}(n_1 \to n_2, tok_1) \overset{\phi}{\sim} \mathsf{fwd}(n_1 \to n_2, tok_2)}$$

$$\frac{}{\mathsf{refill}(n, \mathsf{k}\langle \mathsf{r} \rangle) \overset{\phi}{\sim} \mathsf{refill}(n, \mathsf{k}\langle \phi(\mathsf{r}) \rangle)} \qquad \frac{}{\mathsf{kresp}((n_1, \mathsf{k}\langle \mathsf{r} \rangle) \text{ for } n_2) \overset{\phi}{\sim} \mathsf{kresp}((n_1, \mathsf{k}\langle \phi(\mathsf{r}) \rangle) \text{ for } n_2)}$$

Figure 15: Event indistinguishability

2. Indistinguishability

Our low equivalence relations are parameterized by permutation functions on formal randomness: $\phi : \mathbf{FR} \to \mathbf{FR}$. The idea is that when comparing configurations from different runs, we replace randomness tags r with $\phi(\mathsf{r})$. As such, indistinguishability relations use up to two parameters: the set of adversary nodes $\mathbf{N}$ (not relevant for event indistinguishability below) and the permutation function ϕ. We build up indistinguishability bottom-up, starting from events and traces, and leading to system configurations.

Definition 10 (Event indistinguishability)**.** *We define event indistinguishability structurally as per Figure 15.*

Definition 11 (Message indistinguishability)**.** *Given two messages α_1 and α_2, define* message indistinguishability *w.r.t. a set of adversarial nodes $\mathbf{N}$, and permutation ϕ, written $\alpha_1 \overset{\phi}{\sim}_{\mathbf{N}} \alpha_2$, based on the structure of the messages, as follows:*

$$\frac{n \in \mathbf{N} \implies \xi_1 \overset{\phi}{\sim} \xi_2}{\mathsf{denimup}(n, \rho, \xi_2) \overset{\phi}{\sim}_{\mathbf{N}} \mathsf{denimup}(n, \rho, \xi_1)} \qquad \frac{n \in \mathbf{N} \implies \xi_1 \overset{\phi}{\sim} \xi_2 \wedge c_1 = c_2}{\mathsf{denimdn}(n, \rho, \xi_1, c_1) \overset{\phi}{\sim}_{\mathbf{N}} \mathsf{denimdn}(n, \rho, \xi_1, c_1)}$$

Definition 12 (Trace indistinguishability)**.** *Consider two traces τ_1, τ_2 of length n, each composed of events α_{1i} and α_{2i} respectively. Consider a set of adversarial nodes $\mathbf{N}$. Say that two traces are* indistinguishable *to the set of attacker nodes $\mathbf{N}$, written $\tau_1 \sim_{\mathbf{N}} \tau_2$, if for all $i = 1..n$, it holds that $\alpha_{1i} \overset{\phi}{\sim}_{\mathbf{N}} \alpha_{2i}$.*

We use notation $\tau_1 \simeq_{\mathbf{N}}{}^{init} \tau_2$ if $\tau_1 \overset{\phi_{init}}{\sim}_{\mathbf{N}} \tau_2$ holds for the empty bijection ϕ_{init}. We write $\tau_1 \simeq_{\mathbf{N}} \tau_2$, if there exists a bijection ϕ such that $\tau_1 \overset{\phi}{\sim}_{\mathbf{N}} \tau_2$.

Definition 13 (User state indistinguishability)**.**

$$\frac{\begin{array}{c}\sigma_1 = \langle n, L_1, M_2, R_1, s, c, g\rangle \qquad \sigma_2 = \langle n, L_2, M_2, R_2, s, c, g\rangle \qquad n \in \mathbf{N} \qquad (\mathsf{k}\langle \mathsf{r}\rangle, z) \in L_1 \Leftrightarrow (\mathsf{k}\langle \phi(\mathsf{r})\rangle, z) \in L_2 \\ M(n') = (\mathsf{k}\langle \mathsf{r}\rangle, z) \Leftrightarrow M(n') = (\mathsf{k}\langle \phi(\mathsf{r})\rangle, z) \qquad R(n') = (w, \{\mathsf{k}\langle \mathsf{r}\rangle, \mathsf{k}\langle \mathsf{r}_1\rangle\}, I) \Leftrightarrow R(n') = (w, \{\mathsf{k}\langle \phi(\mathsf{r})\rangle, \mathsf{k}\langle \phi(\mathsf{r}_1)\rangle\}, I)\end{array}}{\omega; \sigma_1 \overset{\phi}{\sim}_{\mathbf{N}} \omega; \sigma_2}$$

$$\frac{\sigma_1 = \langle n, L_1, M_1, R_1, s_1, c_1, g_1\rangle \qquad \sigma_2 = \langle n, L_2, M_2, R_2, s_2, c_2, g_2\rangle \qquad n \notin \mathbf{N}}{\omega_1; \sigma_1 \overset{\phi}{\sim}_{\mathbf{N}} \omega_2; \sigma_2}$$

Definition 14 (Server user configuration indistinguishability)**.**

$$\frac{\mathsf{k}\langle \mathsf{r}\rangle \in K_1 \Leftrightarrow \mathsf{k}\langle \phi(\mathsf{r})\rangle \in K_2 \qquad dq_1 = \xi_1^1 \ldots \xi_j^1 \qquad dq_2 = \xi_1^2 \ldots \xi_j^2 \qquad \xi_i^1 \overset{\phi}{\sim} \xi_i^2, \quad i = 1..j}{(s, c, K_1, B, dq_1) \overset{\phi}{\sim} (s, c, K_2, B, dq_2)}$$

Definition 15 (System configuration indistinguishability)**.** *Given two system configurations $\langle \mathcal{S}_1, \mathcal{U}_1, \tau_1\rangle$, $\langle \mathcal{S}_2, \mathcal{U}_2, \tau_2\rangle$, we say that two configurations are* indistinguishable *written $\langle \mathcal{S}_1, \mathcal{U}_1, \tau_1\rangle \sim_{\mathbf{N}} \langle \mathcal{S}_2, \mathcal{U}_2, \tau_2\rangle$, if they are indistinguishable component-wise. Technically,*

$$\frac{\mathcal{S}_1 = \langle H, rq, D_1\rangle \qquad \mathcal{S}_2 = \langle H, rq, D_2\rangle \qquad \forall n \in \mathbf{N}\ .\ D_1(n) \overset{\phi}{\sim} D_2(n)}{\mathcal{U}_1 = \sigma_1^1 \ldots \sigma_j^1 \ldots \sigma_k^1 \qquad \mathcal{U}_2 = \sigma_1^2 \ldots \sigma_j^2 \ldots \sigma_k^2 \qquad \sigma_j^1 \overset{\phi}{\sim}_{\mathbf{N}} \sigma_j^2, j = 1..k \qquad \tau_1 \overset{\phi}{\sim}_{\mathbf{N}} \tau_2}$$
$$\langle \mathcal{S}_1, \mathcal{U}_1, \tau_1\rangle \overset{\phi}{\sim}_{\mathbf{N}} \langle \mathcal{S}_2, \mathcal{U}_2, \tau_2\rangle$$

3. Proof of Theorem 1

Definition 16 (Strategy determinism w.r.t. formal randomness)**.** *A strategy is* deterministic *w.r.t formal randomness, if for any two traces τ_1 and τ_2, and any $\mathbf{N}, \phi$ such that $\tau_1 \overset{\phi}{\sim}_{\mathbf{N}} \tau_2$, it holds that $\omega(\tau_1) = \omega(\tau_2)$.*

Definition 17 (Valid destinations)**.** *Given a signal configuration $\sigma = \langle n, L, M, R, s, c, g\rangle$, define* valid destinations *for this configuration, denoted as* validdests(σ)*, as* validdests$(\sigma) = \mathsf{dom}(M) \cup \mathsf{dom}(R)$.

Definition 18 (Tight bijections)**.** *Given a pair of configurations $\langle \mathcal{S}, \mathcal{U}, \tau_1\rangle$ and $\langle \mathcal{R}, \mathcal{W}, \tau_2\rangle$, and a set of adversarial nodes $\mathbf{N}$, say that a bijection ϕ is* tight *w.r.t. these configurations and $\mathbf{N}$, if $\mathsf{dom}(\phi)$ is restricted to formal randomness tags that appear in the adversarial components of $\mathcal{S}$ and $\mathcal{U}$, and $\mathsf{img}(\phi)$ is restricted to formal randomness tags that appear in the the adversarial components of $\mathcal{R}$ and $\mathcal{W}$.*

Lemma 1 (Unwinding)**.** *Suppose a set of adversarial nodes $\mathbf{N}$ and two configurations $\langle \mathcal{S}, \mathcal{U}, \tau_1\rangle$ and $\langle \mathcal{R}, \mathcal{W}, \tau_2\rangle$, such that*

- *there is a tight bijection ϕ such that $\langle \mathcal{S}, \mathcal{U}, \tau_1\rangle \overset{\phi}{\sim}_{\mathbf{N}} \langle \mathcal{R}, \mathcal{W}, \tau_2\rangle$, and*
- *all user strategies are valid, i.e.,* valid$(\mathcal{U} \mid \mathbf{N})$ *and* valid$(\mathcal{W} \mid \mathbf{N})$
- *traces τ_1 (resp. τ_2) are consistent with valid destinations for user signal configurations in $\mathcal{U}$ (resp. $\mathcal{W}$), i.e., for any pair signal state σ in $\mathcal{U}$ (resp. $\mathcal{W}$) of node n, if for any valid strategy ω it holds that $\omega(\lfloor\tau\rfloor_n) = \mathsf{SEND}\ n_{dest}$, then $n_{dest} \in$* validdests(σ)
- *there is a message α that is valid w.r.t. $\mathbf{N}$ such that $\langle \mathcal{S}, \mathcal{U}, \tau_1\rangle \longrightarrow \langle \mathcal{S}', \mathcal{U}', \tau_1 \cdot \alpha\rangle$*

Then there is a message β, and a tight bijection ϕ' that extends ϕ, such that $\langle \mathcal{R}, \mathcal{W}, \tau_2\rangle \longrightarrow \langle \mathcal{R}', \mathcal{W}', \tau_2 \cdot \beta\rangle$ and $\langle \mathcal{S}', \mathcal{U}', \tau_1 \cdot \alpha\rangle \overset{\phi'}{\sim}_{\mathbf{N}} \langle \mathcal{R}', \mathcal{W}', \tau_2 \cdot \beta\rangle$ and extended traces $\tau_1 \cdot \alpha$ (resp. $\tau_2 \cdot \beta$) are consistent with valid destinations in the updated configurations $\mathcal{U}$ (resp. $\mathcal{W}$).

Proof. Let $\mathcal{S} = \langle H, rq, D\rangle$ and $\mathcal{R} = \langle H, rq, F\rangle$. The only way to produce the message α at the network level is by rule (Net-Global). By inversion of the rule, there is a user state u_j for which it holds that $u_j \overset{\tau_1, \alpha}{\rightarrowtail} u_j'$ and $\mathcal{S} \overset{\alpha}{\rightrightarrows} \mathcal{S}'$. We proceed by case analysis on α.

Case $\alpha = \mathsf{denimup}(n, \rho, \xi_1)$ On the server side, the two possible transitions are (Aux-Upstream-Server-Event) or (Aux-Upstream-Server-Dummy-Or-Malformed). In either case, the host server is updated through the host response function $H', \gamma = \mathsf{hostresponse}(H, (n, \rho))$, where γ is a potential host response, and the reply queue is potentially extended depending on γ, per rules (Server-Host-Action) and (Server-Host-No-Action). We proceed by distinguishing whether the sending node is adversarial or not.

Sending node is not part of $\mathbf{N}$ In this case, the adversary can observe only the host protocol aspects of the message.

We consider two sub-cases, depending on whether there is a transition $D \overset{\xi_1}{\rightsquigarrow} D'$.

Case $D \overset{\xi_1}{\rightsquigarrow} D'$ Because α is produced by a valid non-adversarial strategy, it does not contain messages that modify the parts of the server state for nodes in $\mathbf{N}$.

Case $D \overset{\xi_1}{\not\rightsquigarrow}$. We hit this case, if ξ_2 is dummy $\bullet$, or it is a malformed event and no premises of Figure 14 are satisfied. The state of the nodes in $\mathbf{N}$ is not affected.

Suppose now that ω_2 is the strategy of user n in configuration $\mathcal{W}$. Let $\omega_2(\tau_2) = \kappa_2$ be the decision of the strategy at this point. Because the strategy is valid it must be the case that we can construct a matching DenIM event ξ_2 such that $\mathrm{kind}(\xi_2) = \kappa_2$.

Let $\beta = \mathsf{denimup}(n, \rho, \xi_2)$. Similar to the above reasoning about α, we can argue that the state of the adversarial nodes on the server is not affected by this message, regardless of whether the server uses the rules where $F \overset{\xi_2}{\rightsquigarrow} F'$ or $F \overset{\xi_2}{\not\rightsquigarrow}$.

Because α is an upstream message, it does not change the set of valid destination for user n. For the same reason, if a valid strategy chooses a destination based on the trace $\tau_1 \cdot \alpha$, the same destination is allowed to be chosen for the trace τ_1. This means that the extended trace is consistent with respect to the updated signal configuration for user n (all other users are unchanged). Similar argument holds for β.

Finally, because no attacker-visible keys are generated by these messages, we keep the bijection, and let $\phi' = \phi$.

Putting everything together we have that both the server and the user states are extended in a way that is not distinguishable to the adversary; and that the updated traces are consistent with valid destinations, which concludes this case of the proof.

Sending node is part of N We consider two sub-cases, depending on whether there is a transition $D \overset{\xi_1}{\rightsquigarrow} D'$.

Case $D \overset{\xi_1}{\not\rightsquigarrow}$. We hit this case, if ξ_2 is dummy $\bullet$, or it is a malformed event and no premises of Figure 14 are satisfied. The state of the nodes in **N** is not affected.

Case $D \overset{\xi_1}{\rightsquigarrow} D'$ This message modifies the server state corresponding to the adversary. However, because this is an adversarial node, it means that configurations $\mathcal{U}$ and $\mathcal{W}$ agree on both the signal state and the strategy ω used by this node. We have that $\kappa_1 = \omega(\lfloor \tau_1 \rfloor_n)$ and $\kappa_2 = \omega(\lfloor \tau_2 \rfloor_n)$. Because the traces are indistinguishable and the strategy is valid, $\kappa_1 = \kappa_2$. We proceed by considering the different non-dummy possibilities for κ_1.

Case SEND n' If this message initiates a new session, cf. rule (User-Denim-Send-New-Session), a fresh key is generated. Because the two configurations agree on the key generation counters, the keys generated in both runs can be matched in the extension of the bijection. If no new keys are generated at this step, which means that sending happens along an established session cf (User-Denim-Send-Initialized), then the bijection remains unchanged.

Case REFILL Similar to the above, we can extend the bijection in both runs.

Case KREQ n' There are two ways the server can process this message

1) There are available keys for n' on the server, cf. rule (Server-Denim-Kreq), and the server picks one of the keys from the keystore of n'. We know that these keys are not in the domain/image of the bijection, because the bijection is tight.
2) There are no available keys for n' on the server, cf. rule (Server-Denim-Kreq-Out-of-Keys), and the server generates a seeded key. These keys are also not in the domain/image of the bijection.

Let $\mathsf{k}\langle \mathsf{r}_1 \rangle$ be the key obtained in the first run, and $\mathsf{k}\langle \mathsf{r}_2 \rangle$ be the key generated in the second run, and we know that r_1 and r_1 are not in the bijection. We can therefore extend ϕ by mapping r_1 to r_2. This extension preserves tightness property of the bijection, because by moving them to the deniable output queue on the server, they are now part of the adversarial state.

Case BLOCK n' This case does not affect the generated keys; and therefore can be matched exactly in both runs, without extending the bijection.

To conclude this case, we observe that when the adversary state on the server is extended, it happens in a way that can be matched in the second run, including matching new keys, which concludes the case.

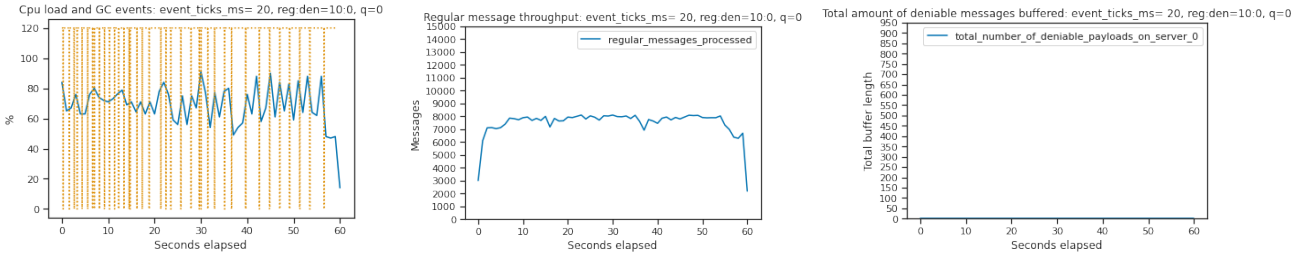
Case $\alpha = \mathsf{denimdn}(n, \rho, \xi_1, c_1)$ We consider two cases.

The receiving node is not part of N This does not change any parts of the adversary state, and we are done immediately.

The receiving node is part of N The parts of the adversarial state are changed in the way that is matched across both runs, but no new keys are generated, and we therefore are done immediately as well.

□

Using the Lemma 1, the proof of Theorem 1 is immediate by induction on the trace τ_1.

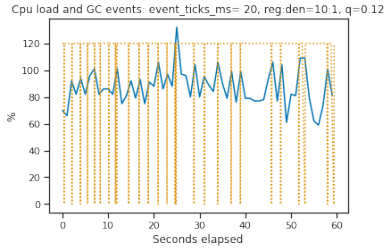


(a) CPU load over time.

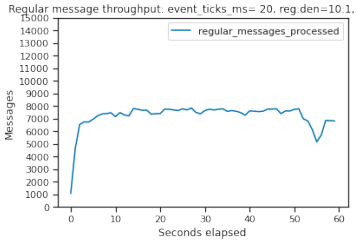
(b) Regular messages processed over time.

(c) Deniable buffer length over time. Notice that since $q = 0$, no deniable traffic is piggybacked to the server.

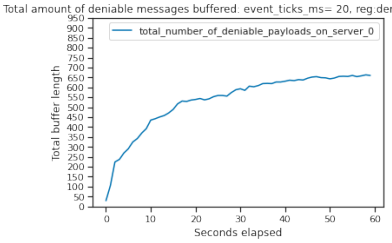
Figure 16: Server statistics collected with setting A1.



(a) CPU load over time.

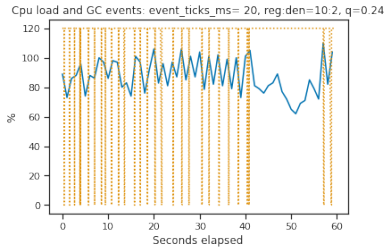


(b) Regular messages processed over time.

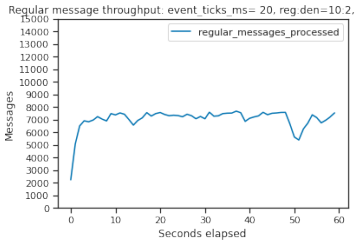


(c) Deniable buffer length over time.

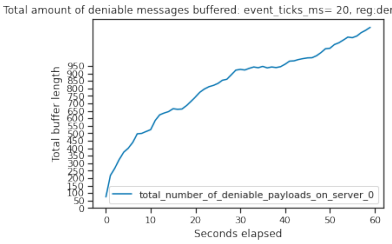
Figure 17: Server statistics collected with setting A2.



(a) CPU load over time.

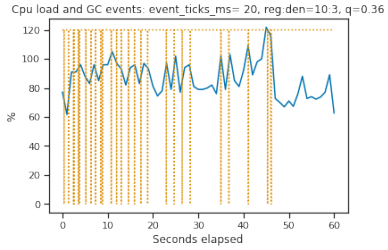


(b) Regular messages processed over time..

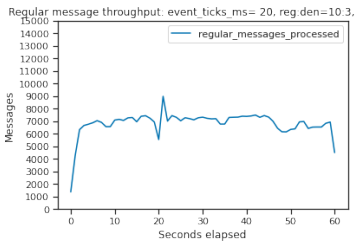


(c) Deniable buffer length over time

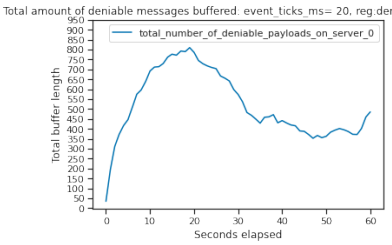
Figure 18: Server statistics collected with setting A3.



(a) CPU load over time.

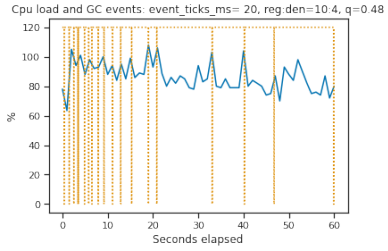


(b) Regular messages processed over time.

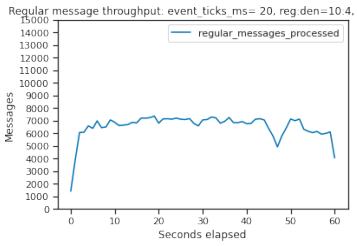


(c) Deniable buffer length over time.

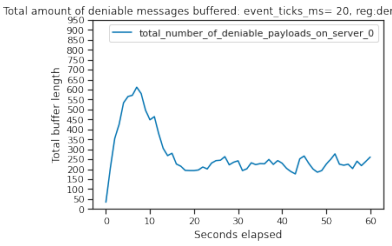
Figure 19: Server statistics collected with setting A4.



(a) CPU load over time.

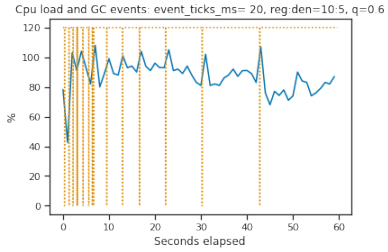


(b) Regular messages processed over time.

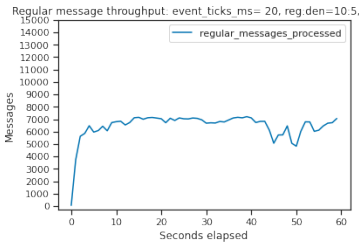


(c) Deniable buffer length over time.

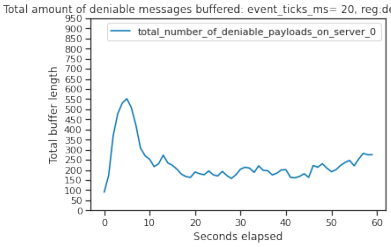
Figure 20: Server statistics collected with setting A5.



(a) CPU load over time.

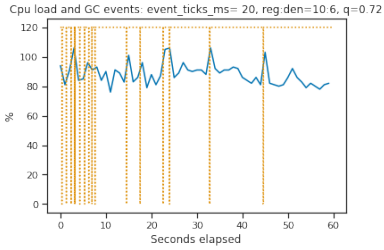


(b) Regular messages processed over time.

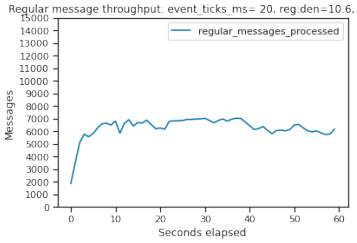


(c) Deniable buffer length over time.

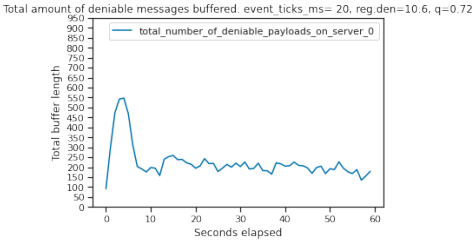
Figure 21: Server statistics collected with setting A6.



(a) CPU load over time.

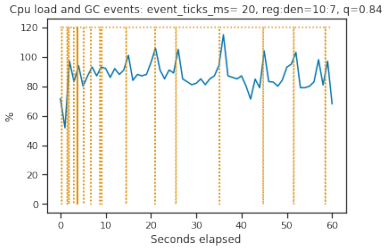


(b) Regular messages processed over time.

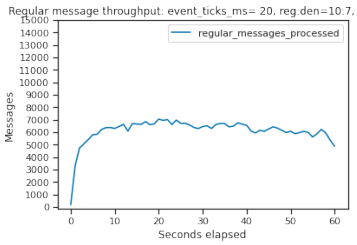


(c) Deniable buffer length over time.

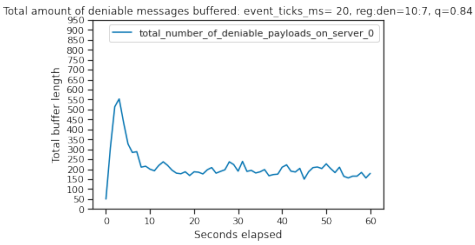
Figure 22: Server statistics collected with setting A7.



(a) CPU load over time.

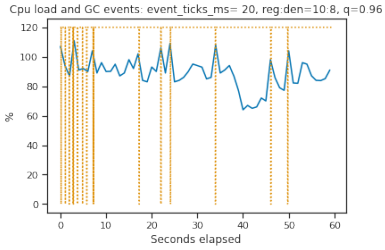


(b) Regular messages processed over time.

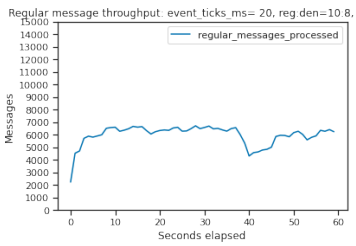


(c) Deniable buffer length over time.

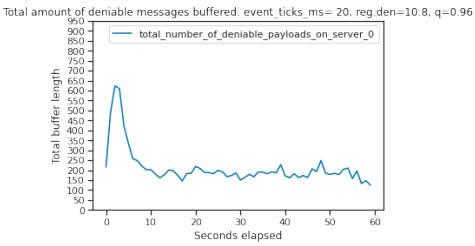
Figure 23: Server statistics collected with setting A8.



(a) CPU load over time.

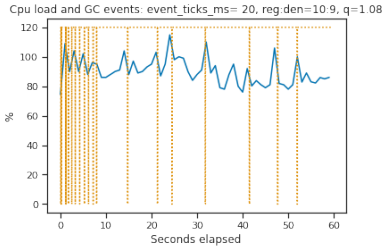


(b) Regular messages processed over time.

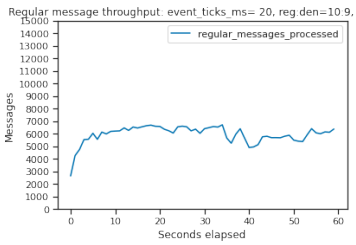


(c) Deniable buffer length over time.

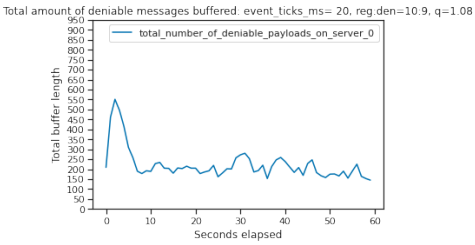
Figure 24: Server statistics collected with setting A9.



(a) CPU load over time.

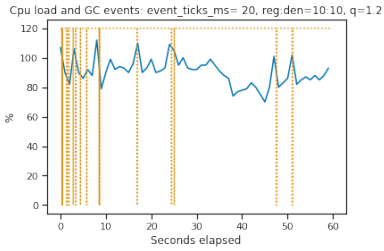


(b) Regular messages processed over time.

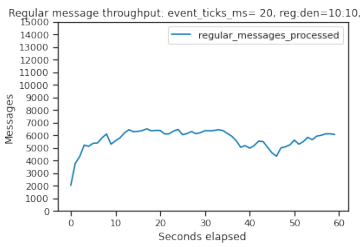


(c) Deniable buffer length over time.

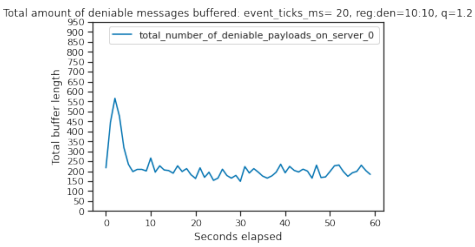
Figure 25: Server statistics collected with setting A10.



(a) CPU load over time.

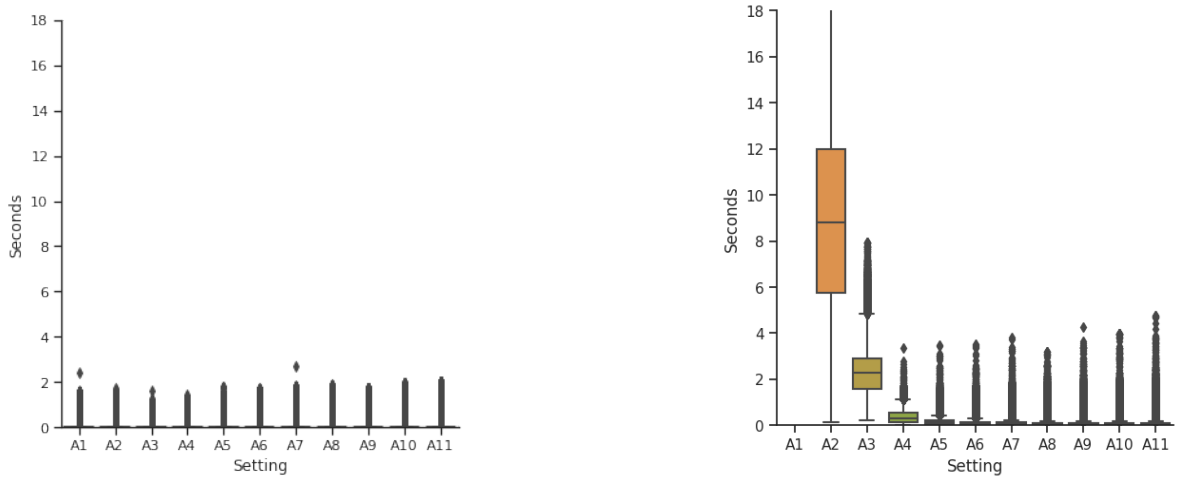


(b) Regular messages processed over time.



(c) Deniable buffer length over time.

Figure 26: Server statistics collected with setting A11.



(a) Regular message latency over time.

(b) Deniable messages latency over time.

Figure 27: Client-to-client message latency measure from $q = 0$ to $q = 1.2$ with step 0.12.