Is Bob Sending Mixed Signals?

Michael Schliep
University of Minnesota
schliep@cs.umn.edu

Ian Kariniemi
University of Minnesota
karin010@umn.edu

Nicholas Hopper
University of Minnesota
hoppernj@umn.edu

ABSTRACT

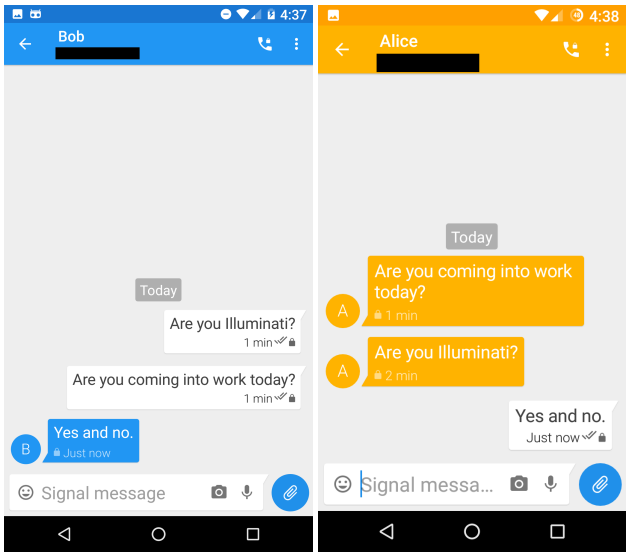
Demand for end-to-end secure messaging has been growing rapidly and companies have responded by releasing applications that implement end-to-end secure messaging protocols. Signal and protocols based on Signal dominate the secure messaging applications. In this work we analyze conversational security properties provided by the Signal Android application against a variety of real world adversaries. We identify vulnerabilities that allow the Signal server to learn the contents of attachments, undetectably re-order and drop messages, and add and drop participants from group conversations. We then perform proof-of-concept attacks against the application to demonstrate the practicality of these vulnerabilities, and suggest mitigations that can detect our attacks. The main conclusion of our work is that we need to consider more than confidentiality and integrity of messages when designing future protocols. We also stress that protocols must protect against compromised servers and at a minimum implement a trust but verify model.

1 INTRODUCTION

Recently many software developers and companies have been integrating end-to-end encrypted messaging protocols into their chat applications. Some applications implement a proprietary protocol, such as Apple iMessage [1]; others, such as Cryptocat [7], implement XMPP OMEMO [17]; but most implement the Signal protocol or a protocol based on Signal, including Open Whisper Systems' Signal [18], WhatsApp [21], Facebook Messenger [9], and Google Allo [10]. These protocols have only recently started to undergo formal security analysis.

Signal was known as TextSecure for versions 1 and 2 of the protocol but changed the name at version 3. TextSecure was developed to provided end-to-end secure messaging over SMS. This required TextSecure to support asynchronous conversations that are tolerable to delayed, out-of-order, and dropped messages. Version 2 of TextSecure added support for group conversations and the capability to send messages over the internet instead of SMS. TextSecure version 3 made a few changes to the cryptographic primitives and protocol along with dropping support for SMS encryption.

To support asynchronous conversations Signal and protocols based on it follow a consistent design. They assume a trusted central server that handles key distribution and message routing and caching. Most implementations provide a user interface to verify or authenticate the keys of conversation participants. To use one of these applications, a user, Alice, registers and uploads a collection of public keys and an identity key to the server. To send a message to Alice, Bob downloads the public key material from the server and initiates a protocol session, encrypting the first message and sending it along with all the necessary data for Alice to initialize her protocol session. The message is cached at the trusted server until Alice is able to retrieve it. The applications provide a method to verify that the server has distributed the correct public keys but not to verify any other functionality of the server.



(a) Alice's view of the conversation. **(b) Bob's view of the conversation.**

Figure 1: Speaker inconsistency in a conversation.

There are many other properties that the server is blindly trusted to provide e.g. participant consistency and speaker consistency. However, in many cases these properties are nearly as important as message authentication. For example, Figure 1 shows Alice's and Bob's differing views of a single Signal conversation. These screenshots demonstrate the need for additional security properties. The first transcript demonstrates Alice's view of the conversation in which she asks two questions of Bob. Without speaker consistency Bob may see the second transcript, which has drastically different meaning than the first.

We argue that a blindly trusted server is not a realistic threat model for secure communication: if a protocol designer does not trust the server to protect the content of messages, why should the server be trusted to protect the order of their delivery, or the list of

WPES'17, October 30, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5175-1/17/10…$15.00
https://doi.org/10.1145/3139550.3139568

their recipients? Nation State Adversaries have previously coerced private companies to provide access to servers or in the case of Lavabit [16] to provide the private keys for a secure email platform. Even without coercion the Signal application has pinned Google's TLS certificate to allow for censorship circumvention via domain fronting, intentionally providing a third party with strong Man-In-The-Middle (MITM) capabilities. We will show that in some cases, even this restricted model of server compromise can be sufficient to compromise the integrity of a Signal conversation.

In this work we analyze Signal with respect to conversations, from the standpoint of a compromised server or server connection. Other work has looked at the Signal messaging protocol and provided formal proofs for various properties of the protocol, sometimes requiring slight modifications. Since Signal does not document a formal threat model we describe one we believe to be consistent with the Signal developers' decisions, along with a stronger practical threat model. We analyze the conversation properties provided by signal under these threat models. Our work does not focus on the usability of Signal, but on whether the application provides any indication of violations of these conversation properties.

Our primary conclusion is that the Signal application puts too much trust in a single entity. We describe attacks against the confidentiality of messages, and integrity of conversations, along with simple application modifications to mitigate these attacks by detecting their presence and alerting the user. We emphasize future applications with a single provider should use an untrusted server model or at least a trust but verify model.

In Section 2 we discuss related work and detail the security and conversation goals we consider. Section 3 and 4 detail the Signal protocol and attacks we demonstarte against the Android and desktop applications. Section 3 focuses on two party conversations with Section 4 looking at group conversations. We discuss traffic analysis of Signal in Section 5 and conclude in Section 6.

2 BACKGROUND

2.1 Signal Design

Briefly the Signal protocol works as follows for the user Alice:

(1) When installing, Alice registers her device with the Signal server using SMS or voice to verify she owns the phone number.
(2) Alice generates a handful of public-private key piars. An identity key, a signed prekey, a last resort prekey, and 100 prekeys. She signs the signed prekey with her identity key.
(3) She uploads the public keys to the server.

Next we quickly describe Alice sending a message to Bob. We describe the process in greater detail in Section 3

(1) The first time Alice sends a message to Bob she fetches his public identity key, signed prekey, last resort prekey, and a single prekey from the server. The server should not hand out a prekey more than once.
(2) Alice then initiates a Signal session on her device and produces a symmetric encryption key. She encrypts her message with this key.
(3) Alice uploads her encrypted message along with the key material required for Bob to initialize his corresponding session.
(4) Bob fetches this material and ciphertext from the server, initializes his view of the session and decrypts the message.

There is only a single protocol session between Alice and Bob. Multiple types of messages may be sent using this single session e.g. two party conversation, attachments, group messages. For a more detailed description of the protocol [13] describes the key exchange, [12] describes the key ratcheting, and [2] details the protocol as implemented in the Signal application.

2.2 Related Work

Frosch et. al. [5] were the first to formally analyze the TextSecure messaging protocol. Their work was performed before TextSecure changed its name to Signal. The authors show the key chaining of Signal is an authenticated encryption scheme. They also describe an Unknown Key Share Attack̃(UKS).

The goal of the Unknown Key Share Attack is to have an adversarial user Bob convince Alice his keys are those of Charlies. Then when Alice sends a message to Bob, Bob can forward the message to Charlie as if Alice sent it to Charlie. Charlie then believes that Alice has sent the message. This attack could be mitigated by simply including the phone numbers of the participants in the Key Derivation Function (KDF) of the Authenticated Key Exchange (AKE). If Alice was to included Bob's phone number in the KDF Charlie would compute a different decryption key than Alice's encryption key. This attack seems to be out of scope of the Signal developers [14]. For Bob to perform the attack in Signal, Bob downloads Charlie's public keys from the server and uploads them as his own. The server does not require proof of knowledge of private keys while uploading public keys.

Cohn-Gordon et. al. [2] formally analyze the Signal protocol as a multi-stage key exchange protocol. They provide the most detailed and up-to-date description of the protocol. They also define and analyze a freshness model for considering forward secrecy in Signal.

Kobeissi et. al. [8] provide a novel method for automated verification of protocols and implementations. They implement a variation of Signal in their framework and apply their automated analysis. They also demonstrate the UKS attack exists in the protocol and a message replay attack exists when a prekey is not used in the initial message of a session.

Rösler et. al. [15] recently analyzed security properties of group conversations in three common messaging applications including Signal. They describe a new model for considering the security of group messaging and define similar properties to ours. Their properties are end-to-end confidentiality, perfect forward secrecy, future secrecy, message authentication, traceable delivery, no duplication, no creation, and closeness.

Traceable delivery is the property that a sender is notified of successful or unsuccessful deliver of a message. No duplication or creation are the properties that a message can not be replayed and an outsider user can not send a message to the group. Finally closeness is the property that only administrative users can modify the group participants.

They identify and demonstrate two attacks we describe in this paper. One attack allows a non-participating user to update group membership and the other allows an adversary to forge receipt of a message. They also identify a potential message ordering vulnerability but describe the attack incorrectly. We discovered these attacks independent of their work.

2.3 Threat Model

Signal does not have a formally documented threat model. We first describe five adversaries with differing capabilities and indicate which ones we believe to be included the Signals original threat model.

Adversary 1 is that of a passive Internet Service Provider. The adversary can monitor all internet traffic originating from and distend for a target device. We assume the adversary can not monitor SMS or voice data.

Adversary 2 has the capabilities of a Signal user participating in a target conversation. The adversary may only control messages originating from its own device.

Adversary 3 is that of an active Internet Service Provider. The adversary may drop, inject, delay, or reorder network traffic but can not break any of the cryptographic assumptions of Signal or TLS.

Adversary 4 is an adversary that has access to the private TLS keys of Signal or a domain front of Signal. This adversary may intercept a target TLS session between a device and the Signal infrastructure.

Adversary 5 is capable of corrupting the Signal servers. The adversary may modify, inject, drop, or reorder any message between any pair of users.

We assume that Signal's threat model only includes *Adversaries 1* and *3*. A more realistic threat model would include all of our adversaries. These adversaries represent the capabilities of Nation State Adversaries that are known to exist.

2.4 Conversation Properties

Unger et. al. [19] identified security and usability properties recent secure messaging research and applications have attempted to provide. We now describe these security and usability properties in a manner consistent with [19] and discuss the academic research related to each property in Signal.

Confidentiality is the property that only the intended recipients are able to read a message. [5] and [2] show that the Signal protocol provides confidentiality but in Section 3.1 we show a passive adversary can learn the contents of some attachments.

Integrity guarantees that a message will not be accepted if it has been modified in transit. Since Signal provides authenticated encryption it also provides message integrity.

Message Authentication implies all recipients can verify the source of a message. Authenticated encryption also implies message authentication.

Participant Authentication implies all participants in a conversation receive proof of possession of a long-term secret from all other participants. The Unknown Key Share Attack is an attack against participant authentication.

Participant Consistency is the property that all participants agree on the participants in a conversation. The UKS attack is an attack against participant consistency. We demonstrate another attack against participant consistency of group conversations that is easier to exploit in Section 4.

Destination Validation is provided when a recipient of a message can verify they are an intended recipient of the message. The existence of the Unknown Key Share attack violates destination validation. Since the attack may have been performed the recipient can not be convinced they were the intended recipient.

Forward Secrecy guarantees all previously encrypted messages remain confidential after all key material has been compromised. [2] and [8] show Signal is forward secure under a passive network adversary that may corrupt past messages.

Backward Secrecy guarantees future encrypted messages remain confidential after compromising key material. [8] shows Signal is backward secret under a passive adversary that may reveal past key material.

Anonymity Preserving is the property that the protocol does not undermine any privacy preserving features of the underling transport protocol. This is not a stated goal of Signal but the application does attempt to circumvent state level censorship. We describe a theoretical attack against this circumvention along with a participant correlation attack in Section 5.

Speaker Consistency implies all participants agree on the order of messages as sent by an individual participant. We demonstrate an attack against speaker consistency under *Adversaries 2, 4,* and *5* in Section 3

Causality Preserving is provided if messages are only displayed after messages that causally proceed them. We demonstrate an attack against causality preservation under *Adversaries 2, 4,* and *5* in Section 4.

Global Transcript is the property that all participants see all the messages in the same order. Since Signal is not speaker consistent or causality preserving it can not provide a global transcript.

Message Unlinkability is the property that proving ownership of one messages does not prove ownership of another.

Message Repudiation exists if there does not exist a cryptographic proof that a user authored a message. [5] discusses message repudiation and argues Signal provides this property.

Participant Repudiation exists if there does not exists a cryptographic proof that a user participated in a conversation.

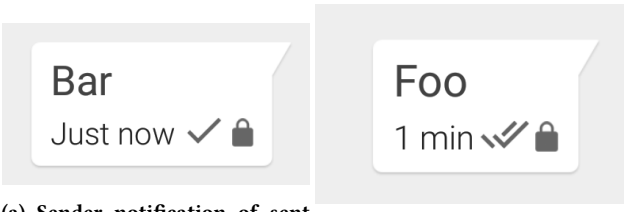
Out-of-Order Resilience is provided if a message is displayed when it has been delayed in transit. This was an important goal of Signal's design.

Dropped Message Resilience is provided if a message can be decrypted without receipt of all previous messages. This was another important goal for Signal.

Asynchronicity is provided if messages can be sent and received when other participants are offline. Asynchronicity was a requirement of the Signal protocol.

Multi-Device Support implies an individual user may participate in a conversation under the same identity from multiple devices. Signal supports multiple devices.

No Additional Service. Signal requires a central service for user key distribution, and message routing and caching.



(a) Sender notification of sent message.

(b) Sender notification of received message.

Figure 2: Sender notification of successful message delivery

Computation Equality is the property that all participants in a conversation perform computation equivalent operations. Signal provides computational equality between participants.

Trust Equality is provided if all participants ave equivalent responsibility. Signal is trust equivalent.

Subgroup Messaging allows messages to be sent to a subset of the participants of a group conversation. We show Signal can provide subgroup messaging but these messages appear as normal messages in the group conversation effecting the transcript consistency. There does not exist a user interface to subgroup messaging.

Contractable groups are supported if participants may leave a group conversation without restarting the protocol. Signal provides contractible groups.

Expandable group are supported if participants can join a group without restarting the protocol. Signal provides expandable groups. We demonstrate a vulnerability in Section 4 where Signal allows participants to be added to a group by non-participating Signal users.

3 PROTOCOL USAGE ATTACKS

The process of Alice sending an encrypted message to Bob via the Signal application is as follows:

(1) Alice computes a ciphertext for her message and assigns it an ID of her current timestamp. She then uploads the message and ID to the server addressed for Bob.
(2) The server notifies Alice when the message has been stored for delivery to Bob. The application notifies the user via a checkmark next to the outgoing message.
(3) If the server has an open websocket channel with Bob the server will send the message to Bob over this channel. If not the server will send a notification to Bob via Google Cloud Messenger (GCM) then Bob will download the message via an HTTP GET request to the server.
(4) After receiving the message Bob notifies the server of receipt via the websocket channel or an HTTP DELETE request. Bob then processes the ciphertext for display.
(5) When the server receives Bob's notification it creates a receipt message with a source of Bob and destination of Alice. The server sets the ID for the receipt to the ID of the message. The server sends this receipt to Alice.
(6) When Alice receives the receipt the application adds a second checkmark to the outgoing message.

Figure 2 shows the user interface of Alice when sending a message. The single checkmark in (a) represents the message was received by the server. The double checkmark in (b) informs Alice the message has been delivered to Bob.

The encrypted message format is

key material||counter||ciphertext

where the key material is used to ratchet the encryption keys, the counter is the sequence number of messages sent from Alice to Bob. The message plaintext is formatted as

flags||body||attachment pointers||group context||expiration timer

The relevant fields of the plaintext are the body which contains the message to display, the attachment pointers which contain the ID, key, and digest of an attachment along with other information that is not relevant to this work, and the group context which is described later.

When Alice includes an attachment in a message to Bob the following happens.

(1) Alice request an attachment ID and an attachment pointer URL from the server.
(2) Alice then generates a symmetric encryption key, encrypts the attachment with this key, and uploads the ciphertext to the attachment URL.
(3) Alice then creates an attachment pointer which contains the ID, key, and digest of the attachment.
(4) Alice sends an encrypted message to Bob which contains this attachment pointer.
(5) Bob requests the attachment URL from the server for the ID in the attachment pointer and downloads the attachment ciphertext.
(6) Bob then verifies the digest, decrypts the ciphertext, and displays the attachment.

Our goal in rest of this section is to demonstrate our attacks against the Android Signal application downloaded from the Google Play Store. The application is distributed with a pinned TLS certificate for the Signal server. We modified the APK to include our own pinned certificate. This allowed us to intercept communication between the application and the server and demonstrate attacks as adversaries 4 and 5. We used mitmproxy to intercept and modify the connections between the application and the server. We clearly state when we intercept or modify communication in this manner, it is only *required* for two of our attacks.

3.1 Confidentiality

Signal, like other messaging applications, is commonly used to send animated images in GIF format. Many messaging services provide integration with databases of gifs, allowing for convenient search of gifs within the app. Signal introduced GIF searching in January 2016, giving users access to the Giphy database inside the Signal application and using Giphy's network API [11]. In order to retain privacy for its users, Signal deployed an HTTP proxy through which a TLS Tunnel is negotiated to Giphy servers. The user's query and Giphy's response flows through this tunnel. According to Signal, this arrangement prevents them from seeing the plaintext content of the search term or the GIF being selected. Giphy sees the search term but not the address of the user who issued the request [11].

Figure 3: The top two images are always downloaded when the user makes a query (assuming the images are not cached).

We evaluated the strength of this approach against fingerprinting attacks and found it to be lacking.

For an average user, the experience of sending a GIF is simple. She selects Giphy from the attachments menu, opening a pane with a search box and beginning the proxied phase of the transaction. She searches and selects an image, which downloads the image to her phone. Once she clicks send, her connection is no longer proxied through an HTTP proxy and the image is sent like any other attachment in Signal.

However, this approach presents an avenue for fingerprinting of Signal traffic. Signal reveals the fact that an attachment exists by communicating directly with the attachment server on both sides of the transaction. Both the sender and receiver use the same amount of bandwidth in their communication with the attachment server and attachments are padded deterministically to 16 bit alignment (in order to satisfy block cipher requirements).

This makes it possible for adversaries 1, 3, 4 and 5 to fingerprint a victim's traffic and obtain with reasonable accuracy the image and the search term that the user entered. This is made easier by the following conditions:

- Users are most likely to pick items that occur earlier in the list of search results. If users cannot see the image they are looking for, they are more likely to modify their search term than scroll down a significant amount.
- There are very few collisions (Our scrapes returned 30 cases) of image ciphertext sizes between terms on the first two images.
- The Signal application downloads images as the user scrolls through the list, allowing the adversary to select an image from a smaller subset of images that are loaded rather than the set of all images in the database.

As a proof of concept, we developed a low cost method for scraping Giphy servers to obtain a database of images to correlate against Signal traffic. Our script queries Giphy in the same manner as Signal. Each query returns a list of 100 items encoded in JSON format, including sizes. It iterates through a list of search terms, parsing and enumerating 100 items from each search (Signal requests 100 results at a time). For each image the size, ciphertext size, and position in the displayed list is recorded. This makes for a database of about 92,000 images. We note that some searches returned less than 100 results.

The list of search terms was obtained from Giphy by scraping for hashtag terms on each of the categories subpages on giphy.com. There are about 25 categories, which include 'actions', 'emotions', 'reactions', and 'animals' among others. At the time of writing our script resulted in 1030 unique search terms. These terms remained consistent for the duration of our research.

The database was inexpensive to build. The script had a run time of 423 seconds on average running on an Intel Xeon 5500 Core i7 CPU. Making search queries to Giphy was the bottleneck in the process, taking on average 361 seconds. Each scrape downloaded 47 kB of data.

We also studied the long term relevance of our scraped data. We ran our script hourly to determine the consistency of the Giphy database. We considered images to be consistent if they remained in the same position in the list using the same search term as before. Under this definition, 90% of images remained consistent after 7 hours. We also looked at the consistency of the top two images in each search, as these images are always loaded for every query. 90% of those images remained consistent after 87 hours (3 days and 15 hours). Therefore, an adversary can scrape every 7 hours for high consistency, or scrape every 87 hours for lower but manageable consistency.

We offer two algorithms for fingerprinting Giphy images from a trace of Signal traffic. We consider a naive algorithm, which returns an exact image and term 65% of the time and otherwise narrows it down to a range of images. We also consider a similar algorithm that expends slightly more resources to get nearly exact results. We refer to this as the 'Precise Algorithm'. Both algorithms are enumerated below. We assume the victim searched with a term within the adversary's list of terms and that the adversary has scraped Giphy at the same time. We also assume the user has not cached any images.

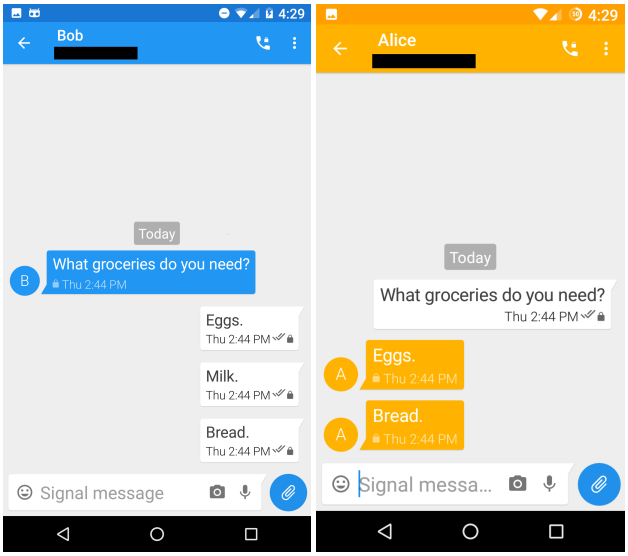
(1) Find the ciphertext size of the image from network flows between the devices and the Signal attachment server.
(2) If the ciphertext size is unique in our database then the process is complete. If there is a collision create a set of the first two image sizes downloaded after the search. The Naive Algorithm will stop here and not investigate collisions.
(3) Compare this set with corresponding sets inside our database to obtain the term.
(4) Find an image with the same size within the subset of images you have in our database for that search term.

We see in Table 1 a demonstration of the precision that can be expected from both algorithms. Both algorithms can only give a range of images due to collisions of image size, but the expected range differs between algorithms. The Naive Algorithm will return a range of less than five images in 99.29% of cases, while The Precise Algorithm provides an adversary with 2 or less images in all cases. The Precise Algorithm will also return the search term used to find the image, which is not the case with the Naive Algorithm.

Size collisions of images that appear in the same search term are the reason the Precise Algorithm cannot obtain the exact image, but these cases are rare. In one scrape, we discovered 430 pairs of images which had size collisions within the same search term.

Table 1: Algorithms

Adversary Algorithm	Range of Images				
	≤ 5	≤ 4	≤ 3	≤ 2	1
Naive	99.29%	97.89%	92.8%	75.93%	65.33%
Precise	100%	100%	100%	100%	99.85%



(a) Alice's view of the conversation. **(b) Bob's view of the conversation.**

Figure 4: Dropped message attack experience by Alice and Bob.

339 of these pairs were instances of duplicate images, which we discovered by hashing the downloaded images. This left 91 pairs of genuine collisions of different images which are linked by appearing in the same search term.

Mitigating this attack is difficult as there is only a limited plaintext space of images on Giphy. Since the HTTPS connection between the application and Giphy leaks so much information any mitigation would require Giphy to pad the HTTP response sent to the client. This is unrealistic to assume of Giphy so Signal should stop tunneling Giphy connections and consider dropping Giphy integration completely.

3.2 Speaker Consistency Attacks

We demonstrate two speaker consistency attacks. These attacks can be performed by adversaries 4 and 5, that is an active MITM or a corrupt server.

3.2.1 Dropped Messages. The Signal application is vulnerable to dropped messages. An adversary with the capability to intercept and modify communication between the message recipient and the server can selectively drop messages while going unnoticed by the sender and receiver.

We describe the attack on a conversation where the adversary drops a message sent from Alice to Bob. When Bob downloads the message from the server in step (3), the adversary modifies the response to contain an empty list of messages. Then the adversary acts as Bob in step (4) and sends a receipt to the server for the message. The server will then act as though Bob issued the delete request by sending a receipt to Alice for the message. The conversation appears as though the messages was delivered correctly to both Alice and the server but Bob has not seen the message.

3.2.2 Message Order. The Signal application does not verify the order of messages when retrieving the list from the server. We detail an attack on message order in a conversation between Alice and Bob.

To break the speaker consistency property our adversary intercepts the message retrieval list from the server to the Bob in step (3). If there is only a single item in the list, the adversary can send an empty list to the Bob. If there is two or more items in the list our adversary reverses the order of the list. Bob will display these messages in the order they appear in the list. Bob will then acknowledge receipt of these messages to the server in reverse order in step (4). Our adversary simply reverses the order of these acknowledgments. Alice and the server assume the messages were displayed in the correct order. Figure 1 shows the conversation as seen by Alice and Bob when this attack is carried out.

3.2.3 Mitigations. To mitigate these two attacks the application should trust but verify the server. Receipts of messages should be end-to-end authenticated and include enough information for the sender to verify in-order delivery.

The sequence number of the message should be used to guarantee in-order delivery. An early goal of Signal is to support asynchronous channels for encrypted messages. Since version 2.7.0 Signal no longer supports SMS and only allows communicating with the server over TCP. There is no need to display messages out-of-order anymore. Messages should never be dropped or delayed.

The receipt of a message should be end-to-end authenticated. As it stands the Signal server creates the receipt that is sent to the sender. It should be generated on the receiving device and authenticated in the same manner as encrypted messages.

We stress that there is currently to much trust in the server. With simple changes to the application it could be a stronger trust but verify model.

4 GROUP CONVERSATION ATTACKS

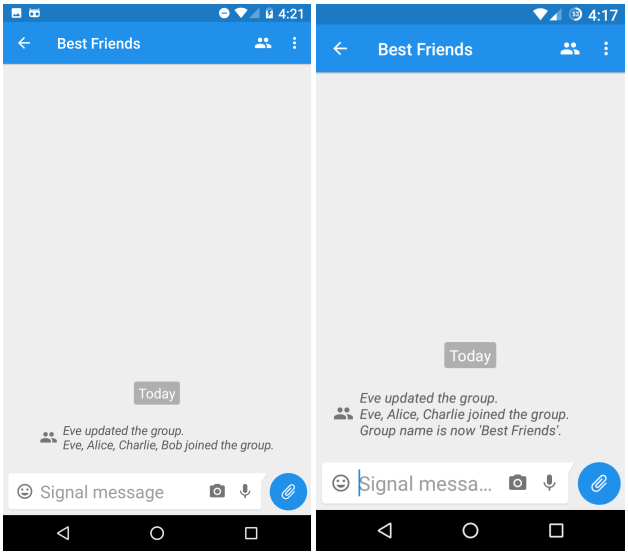
Group conversations were added to Signal in an ad hoc manner. The application only maintains a single Signal protocol session between any two parties. Group conversations are tunneled over these two party protocol sessions.

Recall the encrypted and plaintext message formats from Section 3. The group context data is formatted as

$$\text{ID}||\text{type}||\text{name}||\text{members}||\text{avatar}$$

where ID is the ID of the group. The type field is one of unknown, update, deliver, quit, or request info. The name is the group name displayed to the user. Members is a list of phone numbers of the participant in the conversation, and avatar is an attachment pointer to an image for the group avatar.

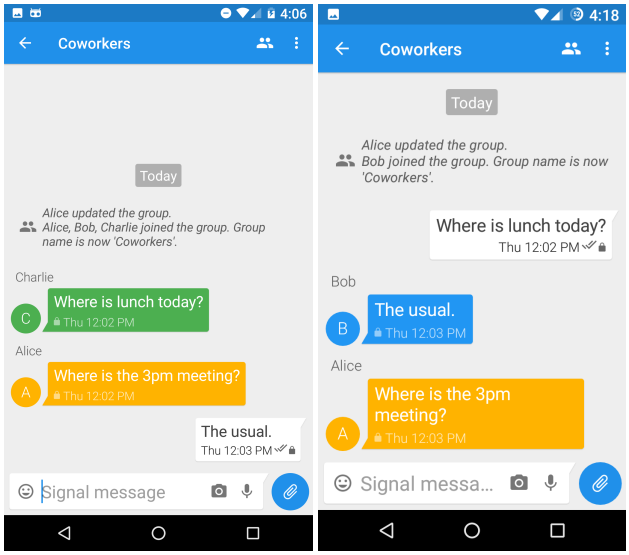
In this work we focus on update and deliver messages. A group message of type deliver indicates the body of the plaintext is to be



(a) Alice's view of the conversation.

(b) Charlie's view of the conversation.

Figure 5: Alice's and Charlie's view of a conversation without participant consistency



(a) Bobs view of the conversation.

(b) Charlies view of the conversation.

Figure 6: Bob and Charlies views of a conversation that does not preserve causality

displayed to the group. Update messages are used to setup or add participants to a group conversation.

The process of Alice setting up a group with Bob and Charlie is as follows:

(1) Alice generates a random group ID and creates a group context message of update type. She sets the members to include Alice, Bob, and Charlie.
(2) She then encrypts and sends the message to Bob and Charlie individually using the Signal protocol session she maintains with each. The messages are sent in the same manner as two party messages discussed prior.
(3) Bob and Charlie receive the messages and create a group with all three participants and the ID, name, and avatar provided.

Updating a group is the same as setting up a group except the existing group ID is used instead of generating a new one. Sending a message to a group is similar except the message has type deliver and the body of the plaintext holds the message to display.

The speaker consistency attacks discussed earlier exist in both two party and group conversations. We now describe three attacks that can be performed by an adversary that can corrupt a participant, intercept communication between a single participant and the server, or corrupt the server. The adversary only needs one of those capabilities (adversaries 2, 4, and 5).

4.0.1 Participant Consistency. A malicious participant would perform the attack as follows. In our attack an adversary Eve constructs a group that contains Alice, Bob, Charlie, and Eve. Eve convinces Alice and Bob that the group contains all four participants while convincing Charlie that the group contains Alice, Charlie, and Eve but not Bob. To create the group Eve creates a group update message containing the phone numbers for Alice, Bob, Charlie, and Eve with a random group ID. Eve encrypts and sends this message to Alice and Bob. Then Eve constructs a new group update message with the same ID but with only the phone numbers of Alice and Eve. She then encrypts and sends this message to Charlie. Figure 5 shows Bob's and Charlie's views of the group under this attack.

There currently exists an open vulnerability in Android and desktop clients where the author of a group update message is not verified to be in the group. This allows any Signal user with knowledge of the group ID to add any new participants to the group. To obtain the group ID the user must have at some point participated in the group or corrupted a participant. This vulnerability was independently discovered by Rösler et. al. [15]; they refer to it as group burgling.

A MITM adversary may perform a similar participant consistency attack. For a group setup by Alice between Alice, Bob, and Charlie. When Alice sends the encrypted group update messages to the server, she will send two at the same time: one addressed to Bob and one addressed to Charlie. The adversary drops the message for Charlie. Alice and Bob will believe they are in a group conversation with each other and Charlie but Charlie does not participate in the group.

4.0.2 Causality Preserving Attack. Signal also does not preserve casuality of messages. We describe an attack from an adversary with MITM capabilities. The goal of the adversary is to convince Charlie that Bob has responded to his message when in fact Bob replies to Alice's.

The adversary only needs to MITM connections between a single participant, Alice, and the server. First Charlie then Alice send a message to the group. When the adversary sees two messages from Alice, one destined for Bob and the other for Charlie, the adversary

stores the message for Charlie and only forwards the message for Bob to the server. Bob then replies to Alice's message, sending the reply to both Alice and Charlie. After the reply is received by Charlie, the adversary forwards the stored message to the server. The transcript for Alice and Bob will contain the conversation as intended by Alice and Bob but Charlie's conversation will have a different meaning. Figure 6 depicts Bob's and Charlie's view of the conversation.

4.1 Mitigations

To provide participant consistency all users should participate in setup and updates to come to a consensus on the group similar to [6]. This mitigation will not allow asynchronous group setup. Consistent asynchronous group communication is still an open research problem.

Currently the Signal application creates a single Signal protocol session between any two participants. This protocol session is used to send all two party conversation messages and all group messages between the two users. To mitigate the speaker consistency attacks a new Signal protocol session should be created for the two party conversation and for each group conversation between the participants.

Causality preservation of group conversations requires more information to be encoded within the encrypted message. Each group message should include a reference to the most recently received message. This reference could be a hash of the most recently received message. Since there does not exist a global ordering of messages, a simple sequence number does not exist to use for message order. The digest can be verified to exist in the transcript before the new message is displayed. These mitigations move away from a blindly trusted server to a stronger trust but verify model.

5 TRAFFIC ANALYSIS

Former Director of the Central Intelligence Agency General Michael Hayden is quoted stating "We kill people based on metadata." [3] It is not sufficient to provide only confidentiality of messages. We should also consider the metadata of a conversation, such as who is participating in a conversation and when messages are being sent. There is an ongoing research effort to provide private communication but little work has attempted to do so for end-to-end secure messaging.

Although it is not a goal of Signal to hide metadata, we quickly discuss what is being leaked to a passive network adversary and potential mitigations for these leaks. These mitigations may not be difficult to implement with a trusted server model.

5.1 Censorship Circumvention

Signal implements domain fronting [4] to circumvent censorship in repressive countries. Domain fronting is automatically applied by the application when associated with phone numbers from those countries. Domain fronting is a censorship circumvention technique that has been implemented by multiple applications [4]. The technique works by connecting to an overt host in a hard to censor cloud provider, then routing the traffic to the covert host. When using domain fronting the Signal application will connect to one of five google.com hosts at random and included a Host header field used by Google servers to reroute the connection to the correct host within Google's cloud. The application has a separate Google certificate pinned when accessing Signal via a fronting domain. This provides Google with the capabilities of adversary 4.

Previous research has looked at how effective domain fronting is in other applications. Wang et. al. [20] applied theoretical fingerprinting attacks to a campus network traffic dataset to evaluate each attack and finds the domain fronted services to be highly fingerprintable with low false positive. We believe Signal to be highly fingerprintable as well due to its traffic pattern being fairly distinct from normal HTTP web traffic.

Signals traffic pattern is consistent between connections to the server while sending or receiving messages. We discuss the traffic pattern in enough detail to fingerprint the application but can not perform a qualitative analysis without representative background traffic which we were not able to procure for this research.

When the application is launched it first creates a websocket connection to the server that stays idle except for heartbeat messages. To send a message the client sends a relatively constant sized message to the server over the websocket channel and receives a short OK message in response. When an encrypted message is received, the server pushes a relatively constant sized message to the client over the websocket channel and a short OK is sent to the server in response. When receiving a message while the application is closed, the phone receives a Google Cloud Messaging (GCM) notification then makes an HTTP GET request to the server for all messages in the client's inbox. The server sends a response with all the requested messages. After parsing the list, the client issues an HTTP DELETE for each message individually. These messages have consistent sizes and consistent timings. This differs from normal web traffic, which downloads a file and may optionally create more connections or download more files after the initial download.

Finally, the application does not use domain fronting for the Giphy proxy or for attachment downloads, leaving the users vulnerable to detection.

To increase the censorship resistance of domain fronting in Signal, the application should use domain fronting for all network traffic. Avoiding fingerprinting is still an ongoing research problem.

5.2 Conversation Metadata

We assume an adversarial network provider. The adversary may record information about network traffic but not modify or delay it in any way. These are the capabilities of an ISP. The adversary has knowledge of all DNS traffic of the clients and all size and timing of packets in a network stream.

When Alice sends a message to the server to be delivered to Bob, the server sends the message if there is an open websocket channel between Bob and the server. Then the server sends a receipt to Alice. The only time this pattern differs is if there is not a websocket channel between the server and Bob, in which case the server sends a GSM message to Bob. Then Bob fetches the message and deletes it, causing the server to send a receipt to Alice. The clients and the server try to minimize the latency and network traffic produced by Signal. This traffic pattern should allow a passive adversary to infer participants of a conversation.

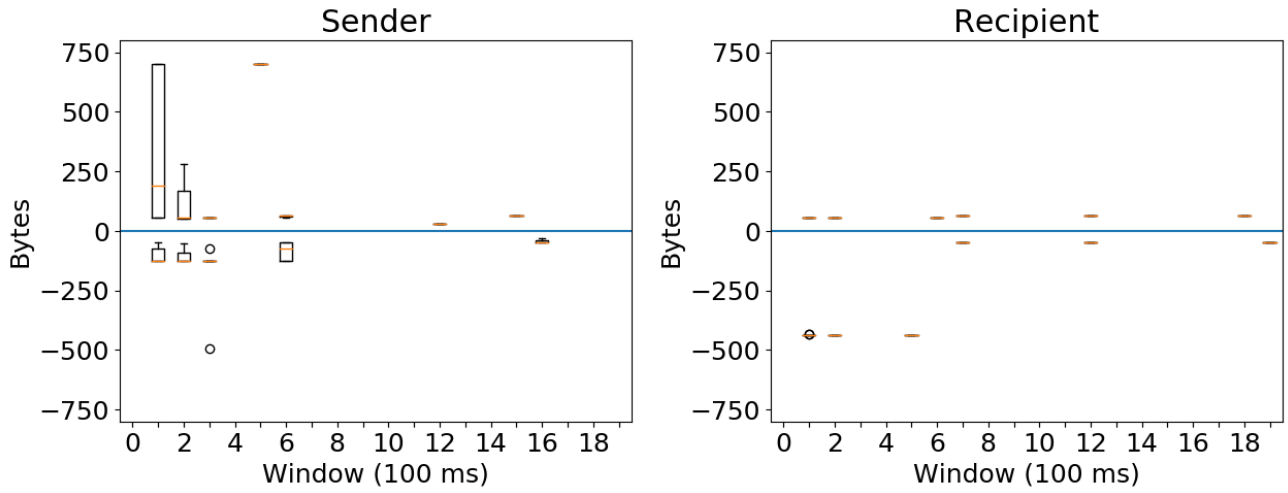


Figure 7: Network traffic pattern for sending and receiving Signal messages.

To demonstrate Signals network traffic patterns we sent 100 message from a sending client to a receiving client and recorded the timing and size of TCP packets. Figure 7 represents the network traffic of these messages. Positive values represent outgoing traffic and negative incoming traffic. We summed all traffic in 100ms windows of sending. The plots show that there is very little delay in sending messages and both the sender and receiver have consistent spikes in the first 200ms.

Mitigating these metadata leaks may not be too difficult. Since Signal relies on a trusted server, it can simply inject delays and noisy messages between clients to degrade the accuracy of these simple attacks. Without an accurate model of Signal's deployment and usage we cannot provide further information on how much noise to add to the communication.

The application may also provide access to Signal via Tor. This would hide the destination of the client from the adversary and help hide the traffic pattern of Signal. This may not hide all the metadata from a determined adversary but will raise the bar for mass surveillance.

6 CONCLUSION

We argue that although Signal is a strong move in the right direction there is still room to improve end-to-end encrypted messaging applications. Starting from the assumption that the Signal server should not be trusted with conversations any more than it would be for messages, we have identified attacks on the extensions of Signal messaging to attachments, conversations, and group participation. These attacks allow an untrustworthy server or third-party relay to violate confidentiality, speaker consistency, participant consistency, and message causality in the Signal Android and Desktop messaging applications.

Our attacks are simple to implement and can be carried out by realistic adversaries without detection against the current implementations. However, in most cases it is also possible to detect and mitigate these attacks with small changes to the implementation or protocol messages. In some cases, further work may be required to determine the most effective way to notify users when these attacks are detected.

It is our contention that the reason these vulnerabilities were not identified sooner is because models of the protocol did not directly incorporate the server as an independent entity. As a result, future work in end-to-end secure messaging protocols should explicitly model both conversational integrity properties and the possibility of a compromised central server, and provide mechanisms to verify the behavior of the server.

7 ACKNOWLEDGMENTS

This work was sponsored by the National Science Foundation under grant 1314637.

REFERENCES

[1] Apple. 2017. (2017). https://www.apple.com/privacy/approach-to-privacy/
[2] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2017. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 451–466.
[3] David Cole. 2014. We Kill People Based on Metadata. (2014). http://www.nybooks.com/daily/2014/05/10/we-kill-people-based-metadata/
[4] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. 2015. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 46–64.
[5] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. 2016. How secure is TextSecure?. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 457–472.
[6] Ian Goldberg, Berkant Ustaoğlu, Matthew D Van Gundy, and Hao Chen. 2009. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 358–368.
[7] Nadim Kobeissi. 2017. (2017). https://crypto.cat/security.html
[8] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. 2017. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.
[9] Moxie Marlinspike. 2016. Facebook Messenger deploys Signal Protocol for end to end encryption. (2016). https://whispersystems.org/blog/facebook-messenger/
[10] Moxie Marlinspike. 2016. Open Whisper Systems partners with Google on end-to-end encryption for Allo. (2016). https://whispersystems.org/blog/allo/
[11] Moxie Marlinspike. 2016. Signal and GIPHY. (jan 2016). https://whispersystems.org/blog/giphy-experiment/
[12] Moxie Marlinspike and Trevor Perrin. 2016. The Double Ratchet Algorithm. (2016). https://whispersystems.org/docs/specifications/doubleratchet/

[13] Moxie Marlinspike and Trevor Perrin. 2016. The X3DH Key Agreement Protocol. (2016). https://whispersystems.org/docs/specifications/x3dh/
[14] Trevor Perrin. 2014. [messaging] How secure is TextSecure? (Nov 2014). https://moderncrypto.org/mail-archive/messaging/2014/001073.html
[15] Paul Rösler, Christian Mainka, and Jörg Schwenk. 2017. More is Less: How Group Chats Weaken the Security of Instant Messengers Signal, WhatsApp, and Threema. (2017).
[16] Dominic Rushe. 2013. Lavabit founder refused FBI order to hand over email encryption keys. (2013). https://www.theguardian.com/world/2013/oct/03/lavabit-ladar-levison-fbi-encryption-keys-snowden
[17] Andreas Straub. [n. d.]. OMEMO Encryption. ([n. d.]). https://crypto.cat/security.html
[18] Open Whisper Systems. 2017. (2017). https://whispersystems.org
[19] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. 2015. SoK: secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 232–249.
[20] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 57–69.
[21] WhatsApp. 2017. (2017). https://www.whatsapp.com/security