Secure Group Communication

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Michael L. Schliep

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Nicholas J. Hopper

May, 2021

Acknowledgements

Many people were involved in this dissertation and I am ever grateful for your contributions and support. First and foremost, my wife, Anna Wagner Schliep, thank you for all of the support you provided for me to pursue my academic interests and conclude this dissertation.

I would like to thank my advisor Nick Hopper for providing the opportunity and guidance for this work. I would also like to thank my co-authors and collaborators: Max Schuchard, John Geddes, Shuai Li, Se Eun Oh, Ian Kariniemi, Rahul Parhi, Eugene Vasserman. Thank you for all of the conversations we have had and contributions you have provided; your collaboration was essential to this dissertation.

Additionally, I am grateful for the other members of the committee, Stephen McCamant, Kangjie Lu, and Andrew Odlyzko, for your volunteered time, feedback, and discussions.

Last, but certainly not least, I thank my parents, David and Denise Schliep. Your continued support and encouragement has made this possible and I would not have achieved this without you.

Dedication

To Anna,
I would never have finished this work without your endless support.

Abstract

Communication privacy is constantly under threat from Nation State Adversaries (NSA). This has led many platforms, such as Facebook and Apple, to implement secure conversation cryptographic protocols in their messaging applications. However, many of the protocols do not provide provable security and do not provide other security guarantees about the conversation. One example of a missing security property is message order consistency between all participants. In this dissertation I demonstrate practical attacks against a popular private messaging protocol and application (Signal). I propose two private group messing protocols with provable privacy properties for two networking models; online instant messaging, and mobile messaging. I show the performance of these models is practical for mutually authenticated groups ≤ 50 participants. Finally, I propose an improvement to the DP5 private presence protocol that reduces the message size from quadratic to logarithmic.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent disclosure of state level surveillance on citizens around the globe [1] has caused an increased need for secure means of electronic communication. What it means to have *secure* communication can be difficult to define with the myriad of devices and services in use today, and many systems have been introduced to fill this need each with varying adversarial models and security properties.

In response, 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 [2]; others, such as Cryptocat [3], implement XMPP OMEMO [4]; but most implement the Signal protocol or a protocol based on Signal, including Open Whisper Systems' Signal [5], WhatsApp [6], Facebook Messenger [7], and Google Allo [8]. These protocols have only recently started to undergo formal security analysis.

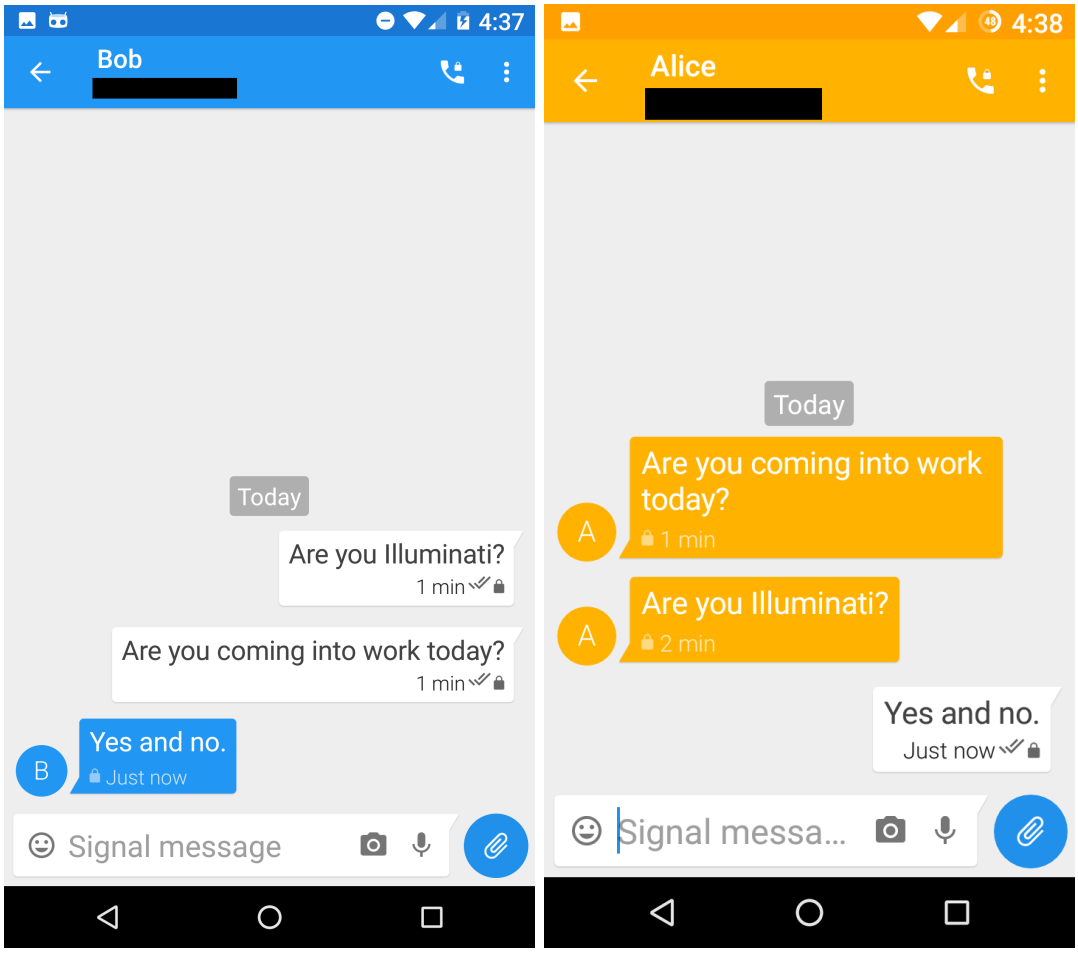
The main objectives of existing secure messaging solutions is to protect the confidentiality and integrity of the messages. However, these two security objectives alone are insufficient to provided meaningfully private messaging. Consider the following two scenarios.

Figure 1.1 contains screenshots from the perspective and Alice and Bob in a secure conversation using the Signal application. Signal provides message confidentiality and integrity but these screenshots demonstrate that even though the message integrity is provided, conversation integrity is not provided which leads to an inconsistent view of a conversation.

As a second example, consider the natural first approach to secure conversations is to encrypt and sign the conversation messages. This is efficient and provides confidentiality and authentication. But the act of signing the messages leaves a record that a user participated in a conversation and what that user said. This record is a loss in security from offline conversations. In the following conversation:

> **Reporter:** What is your company doing illegally?
>
> **Whistleblower:** They are dumping poison into the water.

The whistleblower's participation would not be deniable in the encrypt and sign approach.



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

Figure 1.1: Speaker inconsistency in a conversation.

1.1 Thesis Statement

This dissertation explores the following thesis:

There exists efficient protocols that provably provide strong privacy guarantees for group communication in instant messaging and mobile network settings.

To support this thesis, we analyze the most prevalent secure messaging protocol and identify weaknesses that reduce the confidentiality, integrity, and privacy goals of the protocol. We propose two new protocols for private group communication with provable privacy properties. The first protocol is in the instant messaging setting, that is all users

a online during the conversation, and the second protocol exists in the mobile messaging setting where users may go offline and come back online at a later time without missing messages. There are subtle differences in some of the privacy properties to capture the expectations of the individual network models. Additionally, we explore an auxiliary communication protocol for privately communicating the presence status of friends. We provide an improvement to the private presence protocol that reduces the storage and bandwidth costs significantly.

1.2 Outline

Signal Protocol Analyzes (Chapter 3)

In Chapter 3 we analyze the most prevalent secure messaging protocol and the authors implementation; Signal by Open Whisper Systems. We identify the intended security properties of the protocol and application. We then identify weaknesses and demonstrate vulnerabilities in the protocol. We demonstrate conversation integrity and participant consistency attacks against the protocol and a confidentiality attack due to additional features of the application, specifically sending attachments and Giphy [9] integration.

Synchronous Secure Communication (Chapter 4)

In Chapter 4 we provide a formal definition for the security and privacy goals of secure group communication in the instant messaging model. We design a secure messaging protocol (SYM-GOTR) that provably achieves these goals. We implement this protocol and show that is is effect under real world conditions. Going beyond the provably secure protocol, we examine the usability of our implementation and confirm the user expertise does not compromise the security provided by the protocol via a cognitive walkthrough.

Mobile Communication With Privacy and Integrity (Chapter 5)

In a similar fashion, in Chapter 5 we define group communication security and privacy goals in a mobile messaging model where uses may go offline for a period of time. We detail how these goals change compared to the instant messaging model and propose a

protocol (Mobile CoWPI) that provably achieves these goals. We implement the protocol as a Java library and demonstrate the performance of a deployment with Android clients and servers running on Amazon Web Services [10] and Linode [11].

Private Presence (Chapter 6)

In Chapter 6 we explore an axially protocol to secure message, that of private presence. Starting with an existing protocol DP5 [12] that provides a mechanism for users to retrieve the presence status of their friends without revealing their social graph or identity to a server. We propose an improved protocol (MP3) that significantly improves the network and bandwidth performance via careful utilization of Dynamic Broadcast Encryption(DBE) [13]. Due to these improvements we demonstrate our protocol requires about half the bandwidth compared to DP5.

Chapter 2

Background

This chapter discusses two common networking models for secure messaging protocols and discusses the threat model and security properties from a high level as they relate to both deployment models. We then overview related literature and discuss how it does and does not achieve the required security properties or deployment models.

2.1 Networking Models

We consider two networking models separately. The first be an instant messaging model where all users are online during the entire conversation. Protocols in this model typically make use of a service provider to route messages between clients. The other model is a mobile messaging model where clients may go offline for extended periods of time. It is expected that when a client return online they receive all messages that were sent while they were offline. This requires a service provider to route and store messages for clients.

We must consider these models separately as the definition of some of the security properties differ between the models. For example, consider conversation integrity, in the instant messaging model the protocol can enforce that all users have received a message before the next can be sent. However, in the mobile messaging model some user may be offline when a message is sent and cannot enforce receipt without blocking future messages.

2.2 Threat Model

In either networking model the security provided by any modern protocol needs to withstand strong adversaries. We consider an adversary that may compromise multiple service providers and multiple users. The adversary also has full network control and may drop, modify, reorder, and add network traffic. In effect the adversary can control the service providers any number of participants, unless it would trivially allow the adversary to compromise a target security property. We specifically call out what restrictions are placed on the adversary for each security property when discussing both of the proposed protocols.

2.3 Security Properties

Unger et. al. [14] provide a comparison of security goals of different secure messaging applications. We relate our security goals to the goals of their work where appropriate.

Message Confidentiality is the property that only conversation participants can read a message. More formally, an adversary that does not control a participant in the conversation cannot learn the plaintext of a message. There are two additional properties related to message confidentiality which limit the window of compromised messages even if an adversary is able to compromise any or all participants.

Forward Secrecy is similar to message confidentiality but introduces the concept of key ratcheting. After users have ratcheted their key material all messages sent prior to the key ratchet are confidential even if the adversary is able to reveal the long-term and session state information of any or all participants after the key ratchet.

Post-Compromise Secrecy is similar to message confidential but introduces the concept of key healing. If an adversary is allowed to reveal the long-term and prior session state of any or all of the users in a conversation, after the key healing, all future message remain confidential. The forward and post-compromise secrecy properties bound the window of exposure of any key compromise to a limited period.

Message Authentication is the security property that all participants can verify the author of a message and that a message has not been modified in transit. Message authentication implies message integrity.

Participant Authentication is the property that all honest participants can verify all other honest participants are really who they claim to be. Participant verification transfers between conversations. This is commonly accomplished by verifying long-term key fingerprints in person.

Conversation Integrity is the property that all participants see the same conversation. This includes the order of messages in a conversation and the order of participant changes in a conversation. In relation to Unger et. al. this goal implies speaker consistency, causality preservation, and a global transcript.

Additionally, we consider post-compromise conversation integrity which introduces key healing. A protocol provides post-compromise conversation integrity if after a key healing process, the conversation integrity of future message is not compromised by an

adversary that may have revealed the long-term and session state of users or service providers.

Participant Consistency guarantees all participants of a conversation agree on the set of all participants in the conversation.

Deniability is the property that participants may deny taking part in a conversation. Unger et al. refer to this as participant repudiation. They also discuss two additional deniability properties: message repudiation and message unlinkability. Message repudiation allows participants to deny sending a message and is implied by participant repudiation. Message unlinkability is the property that if a distinguisher can be convinced a user authored one message, this should not prove the authorship of any other message.

2.4 Related Work

Off-The-Record (OTR) [15] is the first academic work to look at providing private instant messaging. OTR provides message confidently, integrity, authentication, repudiation, and unlinkability. However OTR does not provide participant repudiation or conversation integrity. The main limitation of OTR is it only supports conversations between two individuals. There is not a straight forward mechanism to apply OTR in a group setting.

Multiparty OTR (mpOTR) [16] tries to provide the properties of OTR for group conversations. At a high level it works as follows. First, All participants setup pairwise secure channels using a deniable authenticated key agreement (DAKE). Then over the secure channels the participants execute a Group Key Agreement (GKA) to compute an ephemeral encryption key. The users also distribute ephemeral verification keys used to sign conversation messages. The participants also compare a hash of the group information to enforce participant consistency. When Alice wants to send a message to the group she encrypts the message with the ephemeral group key then signs the ciphertext with her ephemeral verification key. Then broadcasts the ciphertext and signature to all participants of the conversation. All recipients can verify the signature is from Alice and decrypt the message. To enforce conversation integrity, at the end of a conversation the participants execute a byzantine agreement on a lexographically

ordered list of the messages. Even though mpOTR provides participant repudiation via the DAKE during setup it does not provide message unlinkability due to the use of the verification keys. With knowledge of a verification key a distinguished can verify all messages authored by a particular user. mpOTR also lacks strong conversation integrity since the transcript consistency is not checked until the conversation has ended and is only checked on a lexographically order transcript. This requires mpOTR to operate in the synchronous instant messaging model with static participants.

Group Off-The-Record (GOTR)[17] utilizes a "hotplugable" Bermister-Desmedt GKA to provide secure messaging for dynamic groups. To set up a conversation all the users first set up secure pairwise channels. Then over those channels the participants execute the GKA. When sending a message Alice encrypts the message with her sending key generated by the GKA. Then periodically the participants perform a transcript consistency check to verify all users have seen the same conversation. The details of the consistency check are not addressed in the paper. GOTR only works in the instant messaging model as all users must be online to execute the GKA and consistency checks, making it not suitable for mobile communication.

Signal [5] (formerly TextSecure) is the most widely deployed protocol for secure mobile messaging. However it has only recently received formal analysis of its security properties [18, 19, 20]. With [21, 22] identifying multiple participant consistency and conversation integrity vulnerabilities in two-party and group conversations. We now quickly describe the group conversation protocol of Signal. When Alice registers with the Signal server she uploads pre-keys allowing other users (Bob) to execute an X3DH [23] two-party key agreement with her while she is offline. When Bob wants to start a conversation with Alice and Charlie he fetches a pre-key for each of them, then executes the X3DH key agreement and sends each a secure "Group Setup" message. Conversation messages are sent in the same fashion, setting up or ratcheting forward a two-party symmetric key with every pair of users, then sending an encryption of the conversation message to each user individually. When Alice receives a group message from Bob she sends a receipt of the message back to Bob. When Bob's phone receives the first receipt of a messages it indicates to Bob the message was delivered. Signal lacks conversation consistency of messages and receipts, Charlie can not verify if Alice has received Bob's message and no order of messages is enforced. We explore Signal in more detail in

Chapter 3.

(n+1)sec [24] is a published draft of a secure group communication protocol. The goals of (n+1)sec are similar to ours with eventual transcript consistency. (n+1)sec utilizes an mBD+P GKA [25] to generate a group sending key and pairwise secrets between all participants. Each group member generates a signing key used to authenticate messages. Periodically participants send a consistency check message that contains their view of the conversation. The delayed consistency checks allow for low latency when displaying message but messages are displayed before a global transcript is enforced. (n+1)sec does not provide a formal security analysis or proofs that the desired security properties are provided. In comparison to (n+1)sec, our protocols favor simplicity and security at the expense of performance.

Asynchronous Ratcheting Trees (ART) [26] describes a group key agreement protocol with forward and backward—Post Compromise—secrecy. The protocol is asynchronous in that it allows a single user to set up the group key while the other users are offline. ART is only a group key agreement and not a full messaging protocol. It does not provide authentication of the author of a message, support for dynamic groups, or conversation integrity. ART works by bootstrapping on secure two-party channels similar to our NAXOS two-party channels. When setting up a group all participants are added one at a time. The group key agreement forms a DH tree where the root node is the group key. Setting up a group with ART is $O(n)$ but performing a single user key ratchet is $O(log(n))$ where n is the number of users in the group.

Recently, the IETF has formed a working group to provide a standard for Message Layer Security (MLS) [27, 28]. The focus of the working group has been on improving scalability to thousands of users with limited security trade offs. The two major trade offs is the lack of conversation integrity and deniability of MLS. These security properties are currently considered open issues by the working group.

Chapter 3

Signal Messaging Protocol

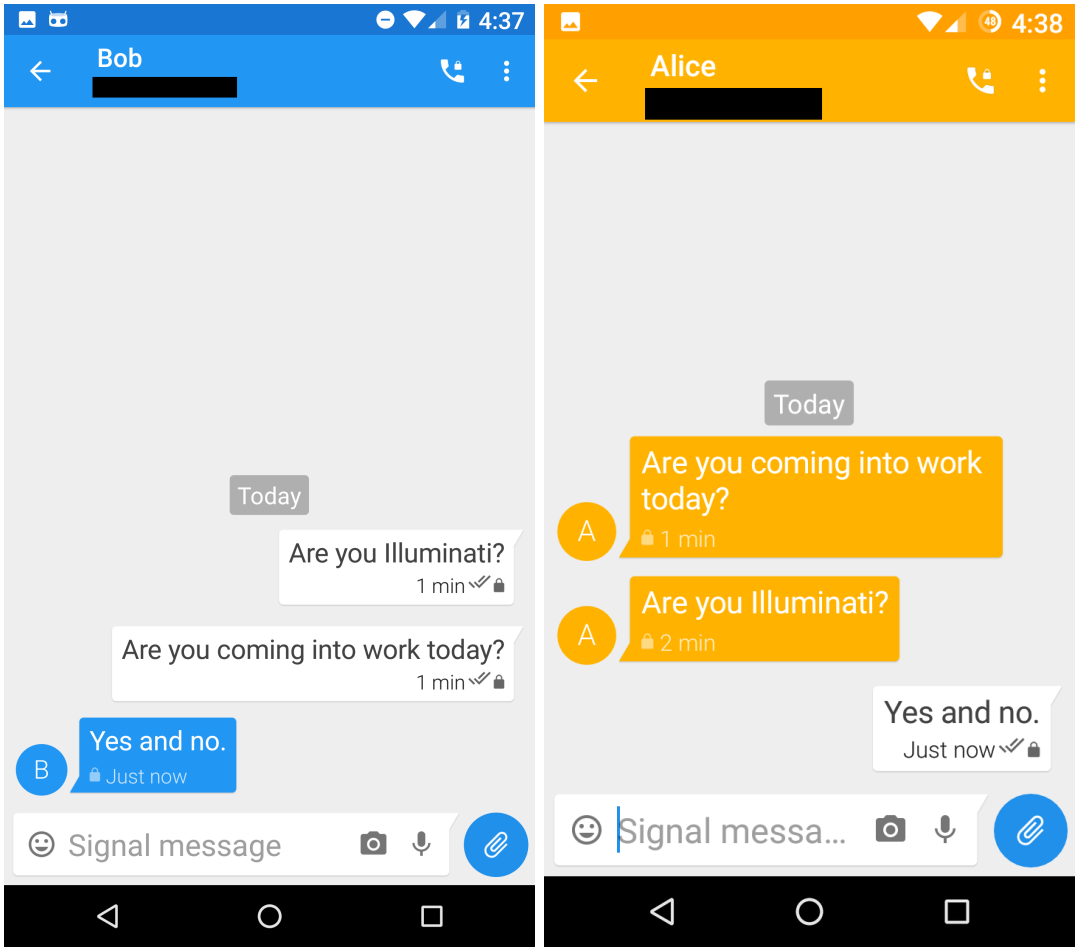
This chapter details the Signal protocol as implemented by the Singal messaging Android application and a multitude of attacks and limitations to the privacy properties of the protocol.

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 transport.

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 application provides a method to verify that the server has distributed the correct public keys but not to verify any other functionality of the server.

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 3.1 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.

This work demonstrates 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



(a) Alice's view of the conversation.

(b) Bob's view of the conversation.

Figure 3.1: Speaker inconsistency in a conversation.

delivery, or the list of their recipients? Nation State Adversaries have previously coerced private companies to provide access to servers or in the case of Lavabit [29] 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 confidentiality and 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. Chapters 4 and 5 detail two protocols in this untrusted server model.

In Section 3.1 we discuss related work and detail the security and conversation goals we consider. Section 3.2 and 3.3 detail the Signal protocol and attacks we demonstrate against the Android and desktop applications. Section 3.2 focuses on two party conversations with Section 3.3 looking at group conversations. Finally, we discuss traffic analysis of Signal in Section 3.4.

3.1 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 pairs. 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 3.2

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 [23] describes the key exchange, [30] describes the key ratcheting, and [31] details the protocol as implemented in the Signal application.

3.1.1 Prior Security Analysis

Frosch et. al. [18] 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 Attack$\tilde{}$(UKS).

The goal of the Unknown Key Share Attack is to have an adversarial user Bob convince Alice his keys are those of Charlie. 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 [32]. 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. [31] 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. [33] 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. [34] 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 outside 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 identified in parallel in this work. 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.

3.1.2 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 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.

3.1.3 Secure Conversation Properties

We now describe the security and usability properties of Section 2.3 and how to specifically relate to Signal. This section also discusses the academic research related to each property in Signal.

Confidentiality is the property that only the intended recipients are able to read a message. [18] and [31] show that the Signal protocol provides confidentiality but in

Section 3.2.1 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 3.3.

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. [31] and [33] 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. [33] 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 3.4.

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 3.2

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 3.3.

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. [18] 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.

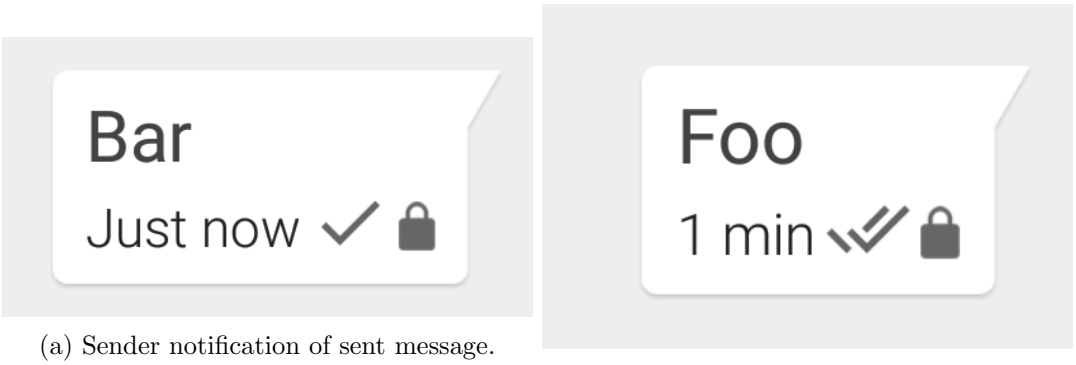
No Additional Service. Signal requires a central service for user key distribution, and message routing and caching.

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



(a) Sender notification of sent message.

(b) Sender notification of received message.

Figure 3.2: Sender notification of successful message delivery

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 3.3 where Signal allows participants to be added to a group by non-participating Signal users.

3.2 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 3.2 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.

3.2.1 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 [9]. 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 [9]. 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

Figure 3.3: The top two images are always downloaded when the user makes a query (assuming the images are not cached).

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 3.1 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. 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.

Table 3.1: Algorithms

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

3.2.2 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.

Dropped Messages

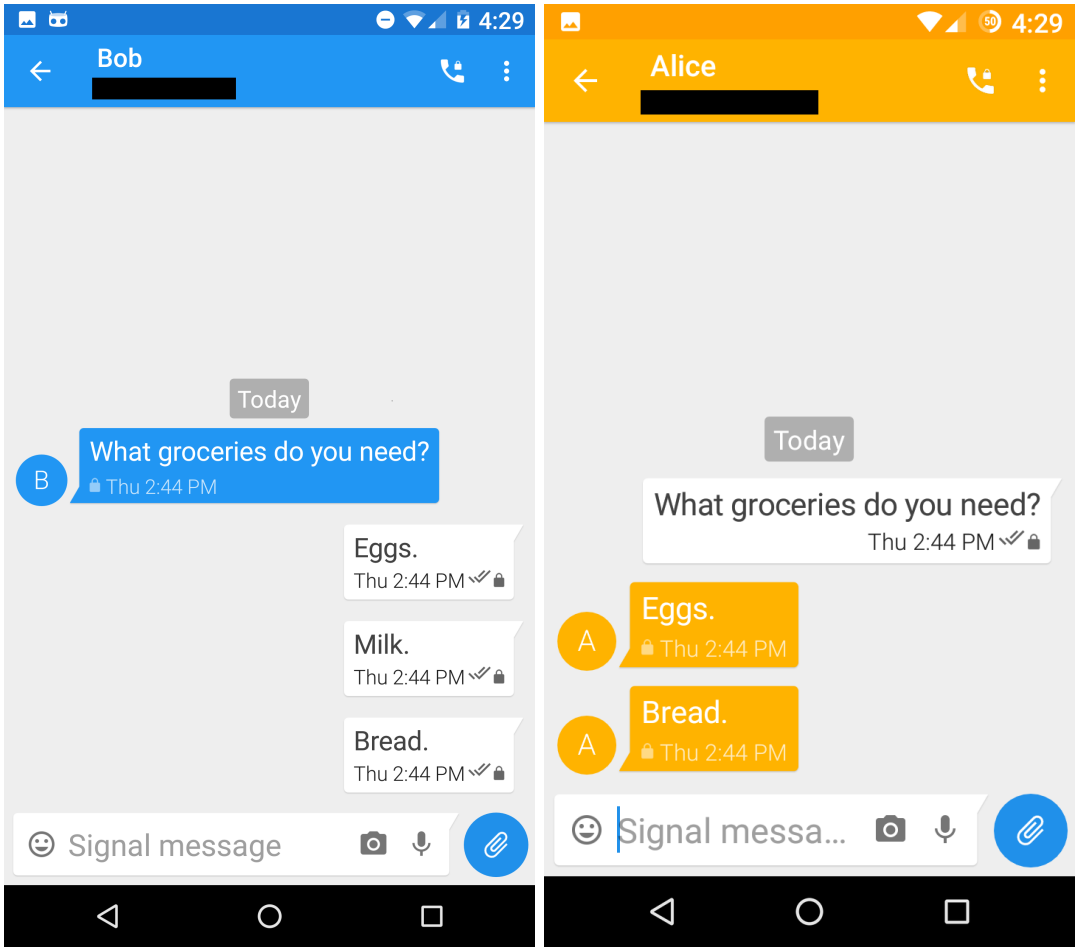
The Signal application is vulnerable to dropped messages. An adversary with the capability to intercept and modify communication between the message recipient and the server can selectively drop messages while going unnoticed by the sender and receiver.

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.

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



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

Figure 3.4: Dropped message attack experience by Alice and Bob.

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 1.1 shows the conversation as seen by Alice and Bob when this attack is carried out.

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.

3.3 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 3.2. 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 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.

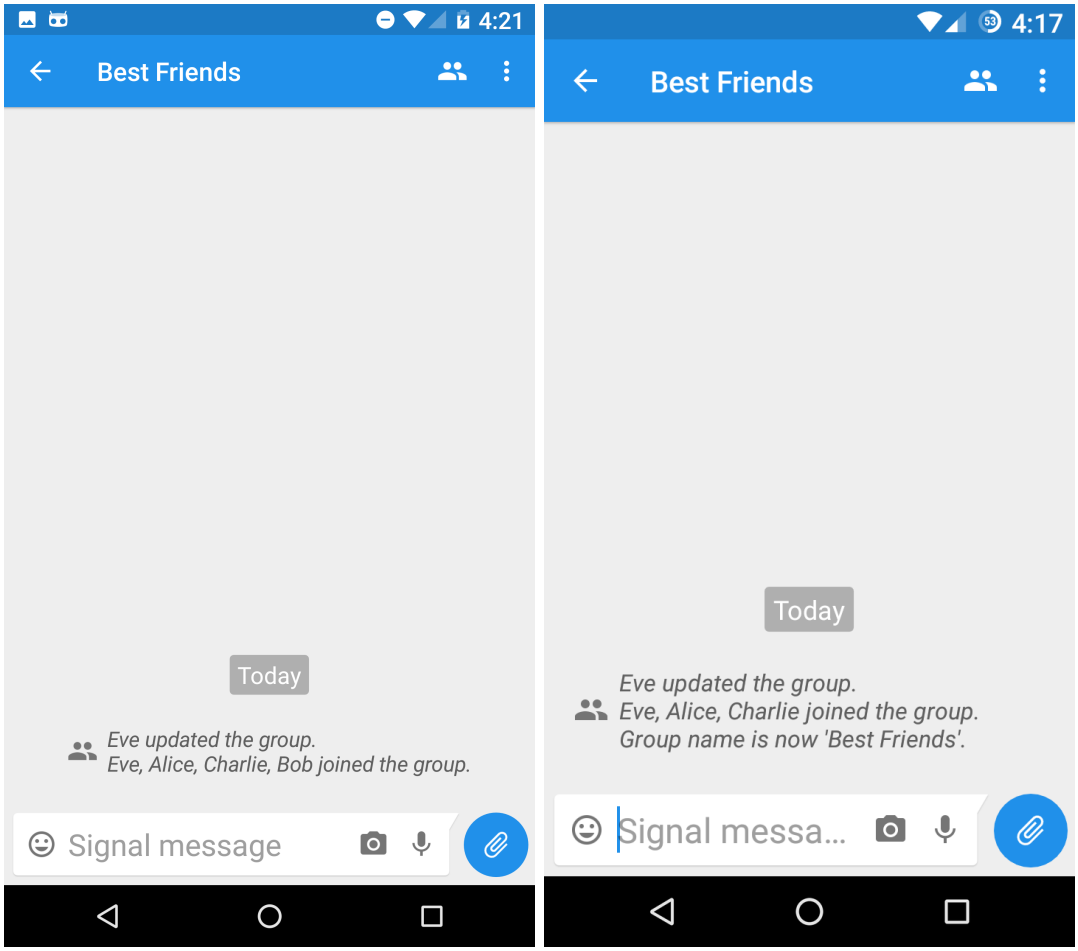
3. Bob and Charlie receive the messages and create a group with all three participants and the ID, name, and avatar provided.

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).

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



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

Figure 3.5: Alice's and Charlie's view of a conversation without participant consistency

phone numbers of Alice and Eve. She then encrypts and sends this message to Charlie. Figure 3.5 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. [34]; 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.

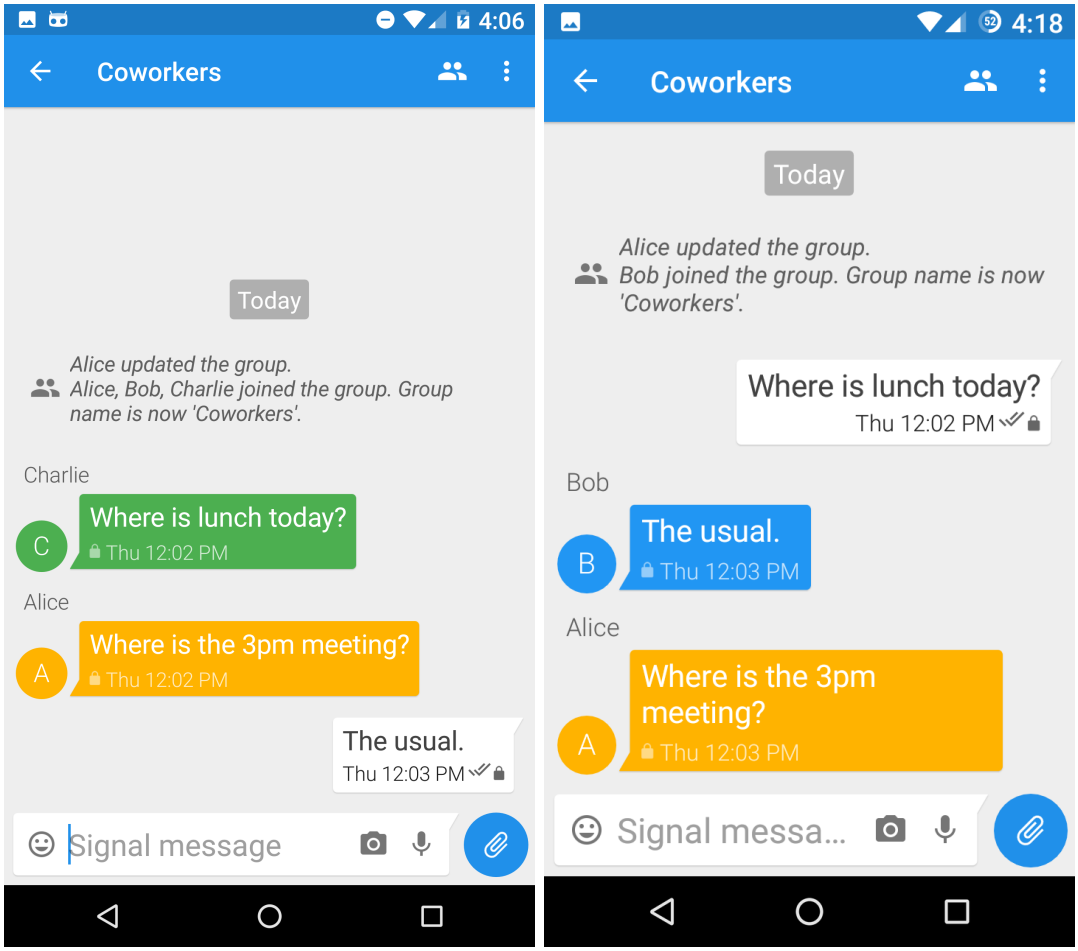
Causality Preserving Attack

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 3.6 depicts Bob's and Charlie's view of the conversation.

3.3.1 Mitigations

To provide participant consistency all users should participate in setup and updates to come to a consensus on the group similar to [16]. This mitigation will not allow



(a) Bobs view of the conversation.

(b) Charlies view of the conversation.

Figure 3.6: Bob and Charlies views of a conversation that does not preserve causality

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.

3.4 Traffic Analysis

Former Director of the Central Intelligence Agency General Michael Hayden is quoted stating "We kill people based on metadata." [35] 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.

3.4.1 Censorship Circumvention

Signal implements domain fronting [36] 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 [36]. 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. [37] 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

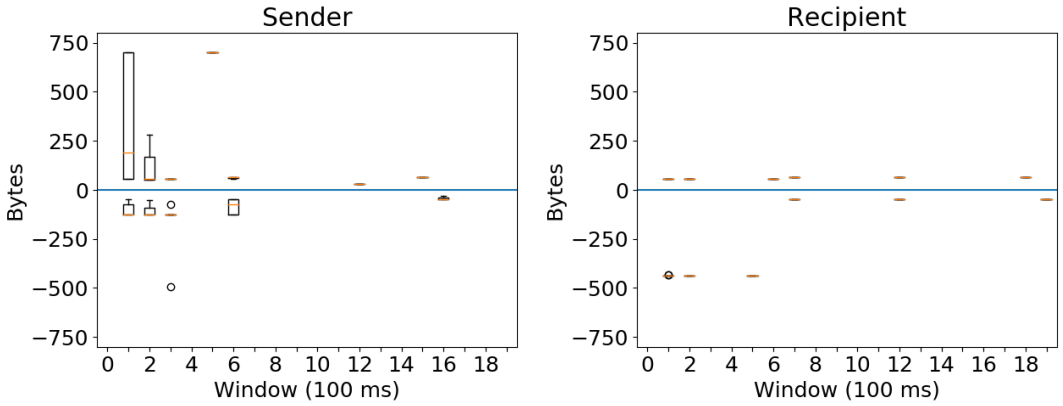


Figure 3.7: Network traffic pattern for sending and receiving Signal messages.

ongoing research problem.

3.4.2 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.

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 3.7 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.

Chapter 4

Synchronous Secure Communication

In this chapter we propose a novel protocol for secure group instant messaging.

A natural first approach to secure electronic conversations is to encrypt and sign the chat messages. This is efficient and provides confidentiality and authentication. But the act of signing the messages leaves a record that a user participated in a conversation and what that user said. This record is a loss in security from offline conversations. In addition to maintaining repudiability, online group conversations present further challenges; there is also an implicit agreement to the participants of an offline conversation by seeing everyone involved, and an agreement on the transcript of the conversation by virtue of every user hearing the same conversation.

Earlier works have attempted to address these problems. Borisov, Goldberg, and Brewer [15] introduced the challenges of providing similar security properties for online conversations and offline conversations, and proposed the original OTR protocol, which provides these properties but is limited to a two user setting. Multi-party OTR (mpOTR) [16] was introduced for secure group conversations. While offering confidentiality, authentication, participant consistency and repudiation, mpOTR does not provide message unlinkability, since all messages from the same sender in a transcript can be linked together; transcript consistency, since an adversary controlling the network can cause participants to terminate with differing transcripts; or a mechanism for dynamic groups.

Two more recent systems have the functionality to handle dynamic groups, GOTR [17] and Signal [5]. GOTR is similar to our work, but lacks message unlinkability and does not provide participant consistency or strong transcript consistency. Signal's goals are different from the previously mentioned works, in that Signal focuses on asynchronous messaging. In Chapter 3 we have shown Signal does not provided speaker integrity, a global transcript, or participant consistency.

We describe a new protocol for group OTR instant messaging, Symmetric Group-Off-the-Record (SYM-GOTR). SYM-GOTR is the first protocol to simultaneously provide message unlinkability, participant consistency, and strong transcript consistency for dynamic groups, allowing an online group chat with similar properties to private offline conversations. SYM-GOTR provides high-security for small synchronous groups where as previous protocols have made a trade-off in terms of security properties provided to support larger or asynchronous groups. Compared to previous work, SYM-GOTR is

efficient in terms of the number and size of protocol messages; under benign conditions all operations require a constant number of messages sent between all pairs of users and all messages are of constant size.

At a high level, SYM-GOTR works as follows. First, each pair of users constructs a secure deniable peer-to-peer (p2p) channel between them. Using these p2p channels, each user shares a group-wide secret, and each pair perform a participant consistency check to verify that all users agree on the participants of the group. To send a broadcast message, a symmetric key for each user is derived from the shared secrets of all participants. This key is used to encrypt the message, then it is broadcast to all users – in contrast to GOTR, the broadcast message has constant size in the number of participants. Because any group member can compute the encryption key of any other group member, these broadcasts are deniable. To achieve the global transcript property users then pairwise compare the digest of the received broadcast – message origin is authenticated in this step. The p2p digest sharing messages are signed with an ephemeral key which encourages participants to behave honestly or be detected and exposed as malicious. When a user joins or leaves the group, users share new group secrets and perform another participant consistency check. With these new secrets the group can again send secure broadcast messages.

Along with the protocol description we provide an open-source Java library implementing SYM-GOTR[1] . We analyze the overhead of this implementation and show its practicality for everyday use. We also provide a plugin[2] for the Jitsi instant messaging client to bring SYM-GOTR to XMPP Multi-User Chats.

In Section 4.1 we formally describe the goals of SYM-GOTR and discuss the current state of related private group messaging protocols. Section 4.2 describes the protocol with Section 4.3 overviewing the security of SYM-GOTR with the full proofs in Section 4.6. We analyze our implementation's performance in Section 4.4. Finally we discuss some challenges of implementing a secure messaging application in Section 4.5.

[1] https://github.com/mschliep/gotr4j
[2] https://github.com/mschliep/jitsi

4.1 Private Group Instant Messaging

4.1.1 Goals

In Section 2.3 we presented a list of goals a secure messaging protocol may provide. Below we describe the goals as they relate to the secure instant messaging model and specifically SYM-GOTR. In this chapter we separate the Conversation Integrity and Deniability properties to be consistent with Unger et al. [14].

Conversation properties. Basic properties of a secure group conversation protocol should include:

- **Confidentiality** A message may only be read by conversation participants.
- **Integrity** No honest party will accept a modified message.
- **Authentication** Participants receive proof of possession of a long term secret from every other participant. Additionally, all participants can verify the author of a broadcast message.
- **Participant Consistency** All honest parties agree on the set of participants.
- **Destination Validation** Honest parties can verify they are an intended recipient of a message.
- **Forward and Backward Secrecy** Previous and future messages are secure when the key material is compromised.
- **Speaker Consistency** All honest parties agree on the sequence of messages sent by any one participant.
- **Causality Preserving** Messages may only be displayed after messages that causally proceed them have been displayed. If two messages (m_2, m_3) are both sent in response to m_1, a causality-preserving protocol can display the messages in the order (m_1, m_2, m_3) or (m_1, m_3, m_2), even if m_2 arrives before m_3.
- **Global Transcript** All participants agree on the order of all messages before processing the next message. A global transcript implies speaker consistency and causality preservation.

- **Anonymity Preserving** The protocol does not undermine the anonymity features of the underlying messaging service or transport.

Deniability properties. Unger et al. also enumerate several properties related to deniability of conversations. Adapted to our setting these include:

- **Message Unlinkability** Proving authorship on one message does not prove authorship of any other message.
- **Message Repudiation** There is no way to prove a user authored any message.
- **Participation Repudiation** There is no way to prove a user participated in a chat.

Group properties. Finally, we also include the following goals that make sense when a conversation has three or more participants, as this is an explicit goal of SYM-GOTR.

- **Computation Equality** All participants perform the same computations.
- **Trust Equality** All participants are trusted to the same degree.
- **Contractible Membership** Participants can leave without restarting the protocol.
- **Expandable Membership** Participants can join without restarting the protocol.

Unger et al. [14] discussed additional properties that are sometimes mutually exclusive to ours. The properties we do not offer focus on asynchronous protocols without a global transcript. These properties are:

- **Out-of-Order Resilient** The ability to process a message that has been delayed in transit.
- **Dropped Message Resilient** Messages may be processed without receipt of all previous messages
- **Multi-Device Support** Users can participate in a conversation from multiple devices. This is a technical detail that we do not address in this work. With SYM-GOTR users can act as a unique participant for each device used in the session.

- **Subgroup Messaging** All messages in SYM-GOTR must be sent to all participants of the conversation. If users wish to communicate with a subgroup they can form a new SYM-GOTR session with the subgroup of participants.

Finally, SYM-GOTR cannot protect against Denial of Service (DoS). DoS is trivial in a group communication system with computation and trust equality. To provide a global transcript every user must see a receipt for a message from every other participant. If an adversary can cause a DoS of this receipt the protocol cannot guarantee transcript consistency. SYM-GOTR halts until the DoS ends or the user is removed from the group. We draw an analogy to an in person group conversation. If a member of the conversation does not wish to participate and abstains from the group that member is causing a DoS on the group conversation. The group can simply continue the conversation without the disruptive participant, SYM-GOTR can be continued in a similar fashion. This is an inherent problem of group conversations without a leader. A weaker global transcript definition could allow conversations to make progress while under DoS. This weaker property is akin to an asynchronous communication model.

4.1.2 Prior Secure Messaging Protocols

The original OTR protocol [15] provided the security properties we desire for two party communication but there does not exist a mechanism to scale the protocol to a group setting. Goldberg et al. proposed a multi-party OTR protocol (mpOTR) [16] with a subset of our goals. They were the first to discuss deniability in the group setting, both in terms of message deniability and also the requirement of participant repudiation. mpOTR provides secure group messaging for a static set of users. Briefly, the mpOTR protocol operates as follows. Out of band, users agree to the set of participants. Then in-band, users perform a group consistency check, set up deniable peer-to-peer (p2p) channels between each pair of users, and then share an ephemeral signature verification key. After this the group performs a Group Key Agreement (GKA), the details of which are not addressed. When a user wishes to send a message, they encrypt it with the group key, and sign it with their ephemeral signing key. After the conversation is complete, users sign and share the lexicographical ordered digest of the entire chat, along with any unreceived messages.

In regards to transcript consistency mpOTR assumes a weaker adversarial model than SYM-GOTR. Their adversarial model does not consider an adversary with full control of the network. Additionally mpOTR can only accommodate conversations between a static group. mpOTR does not provide message unlinkability, causality consistency, or a global transcript. In particular, message unlinkability is violated by the use of a single ephemeral signing key for an entire session. Moreover, causality consistency and global transcript consistency is violated since the signed digests are over lexicographically ordered messages. Finally, mpOTR only provides partial forward and backward secrecy between sessions, not during a single session.

Liu, Vasserman, and Hopper [17] proposed an improved group OTR (GOTR) protocol which we refer to as GOTR due to its use of the Burmester-Desmedt group key agreement protocol. The design and goals of their protocol are most similar to our work. As we will discuss, however, GOTR provides only partial message unlinkability and does not offer participant consistency, or strong guarantees about a global transcript. The protocol provides optional forward and backward secrecy at the cost of re-executing the setup phase. Finally, the size of some messages in GOTR scales linearly with the size of the group, which makes the protocol inefficient for larger group conversations.

GOTR works as follows. Out of band, users agree to the list of chat participants. Users set up secure p2p channels between every pair of users U_i and U_j. Then U_i and U_j perform a three step process to compute a Burmester-Desmedt GKA [38] between the pair. Each user then uses a "Hotplugging" property to combine their $n-1$ (pairwise) group keys into a single "circle" key. Each user broadcasts their circle public key, and all other users are able to compute the circle key from having participated in a single BD-GKA with that user. To send a secure message a participant uses a Key Derivation Function (KDF) to generate a symmetric encryption and MAC key from their circle key, then encrypts the message and chat transcript digest with the keys. The user then broadcasts the encrypted message along with their circle public key to the group. Upon receiving the broadcast all users perform a digest consistency check over the p2p channels. This allows the participants to agree upon the author of the message.

GOTR provides *partial* message unlinkability in that if it can be shown that user U_i contributed some message to a conversation, the user's confirmation of the conversation digest implicitly confirms any previous messages attributed to the user in the transcript,

while *not* linking the user to messages appearing later in the transcript (in contrast to mpOTR, in which confirming one message sent by U_i confirms all messages sent by U_i in the transcript).

4.1.3 System Model

The system model of SYM-GOTR consists of clients and a server. No client is trusted more than any other and all operations are communication and computationally equivalent. The server is used for group coordination and routing messages between the clients. There are two types of messages, peer-to-peer (p2p) and broadcast. Peer-to-peer messages may be sent between any pair of clients. Broadcast messages should be broadcast from one client to every other client in the group. The server should maintain a global order of broadcast messages within the group. Client agreement of the broadcast message order is enforced by the protocol.

In practice we provide a plugin for the Jitsi IM client that allows SYM-GOTR conversations with existing XMPP servers. The server requires no modifications. The conversation takes place in an XMPP Multi-User-Chat (MUC). The clients rely on the server to broadcast message and participant changes but the conversation cannot progress until all clients agree on the participants and transcript.

4.1.4 Threat Model

The properties of SYM-GOTR are provided under a realistic threat model. We describe three threat models and the properties that hold under each. Multiple threat models are needed because some properties are trivially broken by a strong adversary. For example, an adversary with the ability to corrupt SYM-GOTR participants can trivially decrypt messages and break the confidentiality property.

The first adversary we consider has the abilities to perform active and passive network attacks. The adversary may intercept, drop, inject, and delay messages between any participants or the routing server. The adversary may also corrupt a subset of the participants. Finally, this adversary may kick any participant from the SYM-GOTR session and participate in multiple SYM-GOTR sessions with the participants. These abilities are those of an adversarial routing server with corrupt participants.

Under this first model SYM-GOTR provides message integrity, authenticity of participants and their messages, consistency in the set of participants and their secrets among participants, destination validation, speaker consistency, causality preservation, a global transcript, message unlinkability and repudiation along with participant repudiation.

The second adversary we consider is that of a malicious routing server. The adversary may perform passive and active network level attacks, kick participants from a SYM-GOTR session, and participate in multiple non-target SYM-GOTR sessions with the victim(s). Message confidentiality is provided under this threat model.

Our third threat model is that of a passive network level adversary with access to any participant's keys at a given time. This may be an adversary with passive network access while the SYM-GOTR session is ongoing, or an adversary that has stored all SYM-GOTR session network traffic for later attacks. The adversary may participate in non-target SYM-GOTR sessions with the victim. Forward and backward secrecy hold under this third adversarial model.

4.2 Design

4.2.1 Strawman Design

A simple protocol would utilize deniable and authenticated two party channels between all pairs of users in a group to generate ephemeral encryption keys along with ephemeral signing and verification keys for each user. Then to send a message the users would simply encrypt and sign the message then broadcast it to the group. The group then performs a consistency check on the broadcast before processing the next message. This approach is similar to mpOTR with consistency checks after every broadcast. However, this simple protocol does not achieve message unlinkability. Messages by the same author within a single session are linked together via the author's signatures.

To provide unlinkability, new signing and verification keys need to be generated and distributed for every message. Distributing these new verification keys in an authenticated and unlinkable manner is not cheap. The first idea would be to include them in the plaintext of the previous message. However, this method does not provide unlinkability. The signature on the previous ciphertext commits to the next verification key.

These verification keys must be sent over pairwise deniable, unlinkable, authenticated, and confidential channels. This increases the communication cost of the protocol but is needed to achieve all of the properties we want in secure group communication.

We now present the SYM-GOTR protocol.

4.2.2 Primitives

We rely on existing cryptographic primitives for SYM-GOTR. Let l be the system wide security level in bits. Symmetric Authenticated Encryption($AEnc_K(M)$ and $ADec_K(C)$) is used to provide confidentiality and integrity of messages. For authentication we make use of a group G of prime order p with generator g, where the Computational Diffie-Hellman assumption holds.

Four collision resistant hash functions $H_{\{1,2,3,4\}}$ are used as follows. $H_1 : \{0,1\}^* \mapsto Z_p$ is used during the secure channel set up and $H_{\{2,3,4\}} : \{0,1\}^* \mapsto \{0,1\}^l$ for participant consistency, message consistency, and signature consistency respectively.

SYM-GOTR requires three cryptographically secure pseudo-random functions as Key Derivation Functions(KDF): $KDF_1 : (G, G, G, U, U) \mapsto \{0,1\}^l$ where U is the set of all possible user identities, is used during secure channel set up; $KDF_2 : (S, G, U, U) \mapsto \{0,1\}^l$ for key ratcheting; and finally, $KDF_3 : \{0,1\}^* \mapsto \{0,1\}^l$ for broadcast message keys.

SYM-GOTR uses a signature scheme for accusation of malicious group members. The signature scheme consists of three functions: $KeyGen()$ to generate a random signing and verification key pair (sk_i, vk_i) for user i, $Sign_{sk}(m)$ to sign the message m with key sk, and $Verify_{vk}(m, sig)$ to verify the signature sig of m with verification key vk.

In our implementation the authenticated encryption scheme uses AES in Counter Mode along with an HMAC of the ciphertext in an Encrypt-then-MAC fashion as described by Bellare and Namprempre [39]. The IV is always 0 because symmetric keys are never reused. The hash functions and KDFs are built upon SHA-256. The group G is NIST-p256 and ECDSA is used for the signature scheme.

We also assume a messaging server. The server provides group coordination such as group set up and participant change notification as well as peer-to-peer (p2p) communication channels between all participants and a broadcast channel for the group. We

also assume the server attempts to send broadcast messages in a consistent order. This order is verified by all clients before message are displayed and the conversation can progress.

When user U_i calls $\textsc{Send}(Sid, U_j, m)$ the server sends m to U_j for session Sid from U_i. Similarly when U_i calls $\textsc{Recv}(Sid, U_j)$ the function returns the next message m sent by U_j in session Sid to U_i. The broadcast channel provides $\textsc{BcastSend}(Sid, m)$ which broadcasts the sender U_i and the message m to all users of session Sid. Finally the broadcast channel also provides $\textsc{BcastRecv}(Sid)$ returning the next broadcast message m for session Sid along with the claimed sender U_i. These methods are not trusted and do not need to provide any security properties. Sid is a unique session identifier that is determined out-of-band (chatroom name). Users must only initiate a single SYM-GOTR session per Sid. The users U of a SYM-GOTR session are also determined out-of-band. The protocol enforces all participants agree on the Sid and participants.

4.2.3 Overview

The high level view of our protocol is to set up deniable, unlinkable, forward and backward secure, authenticated channels between every pair of users. Then use these secure p2p channels to communicate group state and check the consistency of the transcript. Each user U_i generates a long-term DH public-private key pair (lpk_i, lsk_i). This key pair may be used between protocol sessions to maintain the same identity. U_i also generates an ephemeral secret s_i of l random bits and an ephemeral signing key pair (sk_i, pk_i) that will be used when performing the message consistency checks later. U_i shares the secret, the verification key, and a digest of the group participants over a secure p2p channel with all other participants of the session.

Using the shared values of all participants, symmetric encryption keys are generated with a Key Derivation Function. Each participant has a unique sending key that every other participant can generate. When U_i wishes to send a secure group message she encrypts the message with her sending key and broadcasts the ciphertext to all other participants. Since SYM-GOTR requires secure pairwise channels we do not need a complicated Group Key Agreement protocol. We simply compute symmetric keys from the secret input of all participants.

Upon receiving a secure broadcast message a consistency check is performed. To perform the consistency check all participants share, over the secure p2p channel, a digest of the received broadcast message. U_i will sign her consistency check message with her ephemeral signing key. Then users compare all signed consistency check messages they have received to detect any dishonest participants. A user must only perform a consistency check on a single message at any one time. This enforces ordering of the messages. New key inputs are shared during the consistency check allowing SYM-GOTR to be forward and backward secure. All protocol messages are encrypted with unique keys, resisting replay attacks.

Algorithm 1 describes the SYM-GOTR protocol. There are four main steps to the protocol: secure p2p channel setup (lines 1-2), group setup (line 3), broadcast consistency check (line 6), and signature consistency check (line 7). The secure p2p channel setup builds the deniable, authenticated, forward, and backward secure channels. The group setup phase shares the keys to be used for communication. The message consistency check guarantees that every user has seen the same message and agree on the authorship of the message. Finally, the signature consistency check guarantees that the whole group has accepted the message.

Algorithm 1 SYM-GOTR protocol

1: **for all** $U_j \in U \setminus \{U_i\}$ **do**
2: $\quad$ ChannelSetup(Sid, U_j)
3: $state \leftarrow$ GroupSetup(Sid, U)
4: **while** $state \neq \perp$ **do**
5: $\quad (U_j, c, m) \leftarrow$ SecRecvBcast$(Sid, state)$
6: $\quad sigs \leftarrow$ BcastConCheck$(Sid, U_j, c, m, state)$
7: $\quad state \leftarrow$ SigConCheck$(Sid, U_j, c, sigs, state)$

4.2.4 SYM-GOTR Protocol

We first describe the SYM-GOTR protocol assuming we have a protocol for a confidential, deniable, unlinkable, authenticated p2p channel. With a secure p2p channel the group setup phase is simple. User U_i generates a group-wide secret $s_i \in \{0,1\}^l$. These group-wide secrets are used to generate the ephemeral encryption keys used for secure group messages. U_i also generates a signing key sk_i and a corresponding verification key

vk_i for message consistency checks. Finally, U_i generates a group view gv_i as a hash of all group participants. This group view is used to check that honest participants agree on the group. The group-wide secrets, verification keys, and group views are shared over the secure p2p channels.

Algorithm 2 describes the GroupSetup function as executed by user U_i. GroupSetup outputs the list of group-wide secrets S, list of verification keys VK, and the users signing key sk_i.

Algorithm 2 Group Setup

```
1:  function GroupSetup(Sid, U)
2:      S ← [], VK ← []
3:      s_i ← {0,1}^l
4:      (sk_i, vk_i) ← KeyGen()
5:      S[i] = s_i, VK[i] = vk_i
6:      gv_i = H_2(Sid, U)
7:      for all U_j ∈ U \ {U_i} do
8:          SecSend(Sid, U_j, s_i, vk_i, gv_i)
9:          s_j, vk_j, gv_j ←SecRecv(U_j, Sid)
10:         if gv_j ≠ gv_i then
11:             return ⊥
12:         S[j] = s_j, VK[j] = vk_j
13:     return (S, VK, sk_i)
```

After GroupSetup completes users can now send and receive secure broadcast messages. When user U_i wishes to broadcast a secure message m, she first computes ephemeral sending key $K_i = KDF_3(Sid, (U_i, s_i), (U_1, s_1) \ldots, (U_n, s_n))$. In KDF_3 she places herself at the front of the input as well as in the list of all inputs. This approach allows a recipient to infer the sender and receivers. It also ensures that a key will never be reused even if two participants attempt to send a message at the same time. This approach is used consistently throughout the protocol. She then encrypts the message as $c = AEnc_{K_i}(m)$ and broadcasts the ciphertext, BcastSend(Sid, c).

When U_i receives a broadcast message c from U_j, U_i computes $K_j = KDF_3(Sid, (U_j, s_j), (U_1, s_1), \ldots, (U$ and checks that c is an authenticated encryption with K_j. If c does not authenticate it is dropped and the next broadcast is received. Algorithm 3 describes the process. When an authentic c is received a broadcast consistency check is performed (Alg. 1,

line 6).

Algorithm 3 Receive Secure Broadcast

1: **function** SecRecvBcast($Sid, (S, VK, sk_i)$)
2: $U_j, c \leftarrow$ BcastRecv(Sid)
3: $K_j \leftarrow KDF_3(Sid, (U_j, S[j]), (U_1, S[1]), \ldots, (U_n, S[n]))$
4: $m \leftarrow ADec_{K_j}(c)$
5: **if** $m \neq \perp$ **then**
6: **return** (U_j, c, m)
7: **else**
8: SecRecvBcast($Sid, (S, VK, sk_i)$)

The broadcast consistency check guarantees all participants have received the same c from U_j and that U_j authored the message, then the plaintext $m = ADec_{K_j}(c)$ is displayed. Algorithm 4 describes the broadcast consistency check. To check the consistency of a ciphertext c, U_i computes a digest of the ciphertext along with the string "accept" or "reject". This digest is then signed with the ephemeral signing key sk_i and shared to all participants over the secure p2p channels. The message is only displayed for U_i if all participants notified U_i that they have "accept"ed the broadcast as valid from U_j. If U_j did not author c, U_j should "reject" the ciphertext. If there is a malicious participant and they try to convince U_i to display the message and another user to not display the message, the malicious participant will be identified in the next phase.

Following the broadcast consistency check a signature consistency check is performed to identify misbehaving participants or confirm the message was displayed to all users. The simplest method would be to share all of the received signatures $sigs$, but this is expensive as it grows with the size of the group. We perform an optimistic optimization by hashing all of the entries of $sigs$ that "accept" the broadcast, then share all entries that do not. Under benign conditions this optimization shares a small constant sized message. This method also distributes new ephemeral group-wide secrets (s_i, vk_i) to provide forward and backward secrecy of group messages.

Algorithm 5 describes the signature consistency check. Algorithm 6 describes how the optimization is computed.

Algorithm 4 Broadcast Consistency Check

1: **function** BcastConCheck($Sid, U_j, c, m, (S, VK, sk_i)$)
2: $valid = []$
3: $invalid = []$
4: **if** $U_i = U_j$ and U_i did not send c **then**
5: $d_i \leftarrow H_3(\text{“}reject\text{”}||c)$
6: $sig_i \leftarrow Sign_{sk_i}(d_i)$
7: $invalid[i] \leftarrow (U_i, d_i, sig_i)$
8: $m \leftarrow \perp$
9: **else**
10: $d_i \leftarrow H_3(\text{“}accept\text{”}||c)$
11: $sig_i \leftarrow Sign_{sk_i}(d_i)$
12: $valid[i] \leftarrow (U_i, d_i, sig_i)$
13: **for all** $U_k \in U \setminus \{U_i\}$ **do**
14: SecSend(Sid, U_k, d_i, sig_i)
15: $d_k, sig_k \leftarrow$SecRecv(Sid, U_k)
16: **if** $d_k \neq d_i$ **or not** $verify_{VK[k]}(d_k, sig_k)$ **then**
17: $invalid[k] \leftarrow (U_k, d_k, sig_k)$
18: $m \leftarrow \perp$
19: **else**
20: $valid[k] \leftarrow (U_k, d_k, sig_k)$
21: **if** $m \neq \perp$ **then**
22: DisplayBcast(U_j, m)
23: **return** $(valid, invalid)$

Algorithm 5 Signature Consistency Check

function SIGCONCHECK($Sid, U_j, c, (S, VK, sk_i)$)
  $S' \leftarrow [], VK' \leftarrow []$
  $S'[i] \leftarrow \{0,1\}^l$
  $(sk'_i, VK'[i]) \leftarrow KeyGen()$
  $share_i \leftarrow$ OPTIMIZESHARE$(c, sigs)$
  $sucess \leftarrow true$
  for all $U_k \in U \setminus \{U_i\}$ **do**
    SECSEND$(Sid, U_k, share, S'[i], VK'[i])$
    $share_k, S'[k], VK'[k] \leftarrow$SECRECV$(Sid, U_k)$
    if $share_k \neq share_i$ **then**
      $sucess \leftarrow false$
      Warn the user.
      Share inconsistent signatures.
  if $success = true$ **then**
    CONFIRMBCAST(U_j, m)
  return (S', VK', sk'_i)

Algorithm 6 Optimize Share

function OPTIMIZESHARE$(c, (valid, invalid))$
  $input =$ ""
  for all $(U_j, d_j, sig_j) \in valid$ **do**
    $input \leftarrow input||(U_j, sig_j)$
  $h = H_4(input)$
  $d_{valid} \leftarrow H_3(\text{“}accept\text{”}||c)$
  return $(d_{valid}, h, invalid)$

Churn

We describe how SYM-GOTR handles churn in regards to a single user. We discuss handling churn in reference to Algorithm 1. When a new user is added to an existing SYM-GOTR session each existing user simply executes ChannelSetup with the new user and continues execution from line 3. If the session is in the middle of the processing a broadcast, lines 6-8, the existing users finish processing the broadcast before adding the new user.

Removing a user is not as simple, e.g. a user may lose network connectivity while processing a message. If a user U_j is notified of a user leaving the group before line 5, they can simple remove the user from their list and continue execution from line 3. If a user is removed after a broadcast is received but before the broadcast consistency check the broadcast cannot be guaranteed to be seen by all users and must be dropped. If a user is removed while performing the broadcast consistency check or the signature consistency check some users may have already displayed the broadcast, so a warning should be displayed notifying the users that the previous message may not have been seen by all participants. In any case execution should continue from line 3.

P2P Secure Channel

The SYM-GOTR protocol does not require a specific secure p2p channel protocol. It only requires that the p2p channel be encrypted, authenticated, forward and backward secure, deniable, and unlinkable. We could have used OTR as the p2p channel but for efficiency we implemented a simpler secure p2p protocol that meets our needs.

P2P Channel Setup

Set up of the secure p2p channel uses the NAXOS [40] deniable Authenticated Key Exchange. Since we do not assume a Public Key Infrastructure we simply send the long-term key with the ephemeral key in the first message. User identity and key verification are addressed later.

Algorithm 7 describes the deniable AKE between user U_i and U_j where lsk_i is the long-term secret key of user U_i.

After the deniable AKE completes, each user U_i can generate a pair of ephemeral,

Algorithm 7 P2P Secure Channel Setup

function ChannelSetup(Sid, U_j)
  $esk_i \leftarrow \{0,1\}^l$
  $h_i \leftarrow H_1(esk_i, lsk_i)$
  Send($Sid, U_j, g^{h_i}, g^{lski}$)
  $epk_j, lpk_j \leftarrow$ Recv(Sid, U_j)
  $K_{ij} \leftarrow KDF_1(epk_j^{lsk_i}, lpk_j^{h_i}, epk_j^{h_i} U_i, U_j)$
  $K_{ji} \leftarrow KDF_1(epk_j^{lsk_i}, lpk_j^{h_i}, epk_j^{h_i} U_j, U_i)$

deniable, and authenticated sending and receiving keys K_{ij}, K_{ji} with every other participant $\forall U_j \in U \setminus \{U_i\}$

P2P Key Ratcheting

The keys for the p2p channels are ratcheted forward every message to provide forward and backward secrecy. SYM-GOTR ratchets these keys similar to OTR [15]. When Alice sends a message to Bob she includes a new DH public key. To send the next message she uses her last sent DH share and the DH public key she most recently received from Bob to compute a shared secret and encrypt the message. All p2p protocol messages sent in SYM-GOTR expect a response. This allows SYM-GOTR to use the keys generated during the secure p2p channel setup phase as the first sending and receiving keys to bootstrap the DH key ratcheting. Algorithm 8 describes the functions of the secure p2p channel and key ratcheting. SecSend describes U_i sending m to U_j and SecRecv describes U_i receiving m from U_j, Sid is the session id for the SYM-GOTR session. Participants must only maintain a single p2p channel per pair of user per session.

4.2.5 User Authentication

The deniable AKE between all participants does not provide full user authentication. The AKE only guarantees that the remote party knows the long-term secret that corresponds to the long-term public key. To authenticate a remote user and associate that user with their long-term public key we use the Socialist Millionaire Protocol (SMP) [41]. SMP must be executed between every pair of participants with inputs being their long-term public keys and a pre-shared secret. If SMP is successful the pair of users can

Algorithm 8 P2P Key Ratchet

Require: id_i is the id of last sent DH public key and id_j is the id of the most recently received DH public key. S_i contains DH private keys of i for j and S_j contains the DH public keys of j for i.

1: **function** SecSend(Sid, U_j, m)
2: $r \in_R Z_p$
3: $S_i[id_i + 1] \leftarrow r$
4: $priv \leftarrow S_i[id_i]$
5: $pub \leftarrow S_j[id_j]$
6: $K_{ij} \leftarrow KDF_2(pub^{priv}, U_i, U_j)$
7: $c \leftarrow AEnc_{K_{ij}}(m, g^r)$
8: Send(U_j, Sid, id_j, c)
9: **function** SecRecv(Sid, U_j)
10: $id_i, c \leftarrow$ Recv(U_j, Sid)
11: $priv \leftarrow S_i[id_i]$
12: $pub \leftarrow S_j[id_j]$
13: $K_{ji} \leftarrow KDF_2(pub^{priv}, U_j, U_i)$
14: $m, pub_{new} \leftarrow ADec_{K_{ji}}(c)$
15: $S_j[id_j + 1] \leftarrow pub_{new}$
16: Delete old key ratchet material.
17: **return** m

trust the remote user is the expected party. This trust is linked to the long-term public key of a user and may be applied in future SYM-GOTR sessions. That is the SMP need only be executed once between a pair of users to build trust in a user seen in multiple sessions.

4.3 Security

In this section we discuss the security properties of SYM-GOTR under the threat models described in Section 4.1.

In terms of our goals stated earlier, we achieve *confidentiality*, *integrity*, and *message authentication* through the underlying authenticated encryption scheme. *Forward and backward secrecy* are provided by the key ratcheting of the p2p channels and the new group keys generated before every broadcast message.

SYM-GOTR provides *participant consistency* with the simple consistency check on line 10 of Algorithm 2. *Participant Authentication* is enforced by the NAXOS AKE executed between all pairs of participants.

The *anonymity preserving* property is provided because SYM-GOTR does not incorporate any information about the transport channels and a participant may generate fresh long-term keys and pseudonyms for every session. To avoid generating fresh long-term keys and pseudonyms the two party secure channel setup may be modified to provide anonymity preservation. As described in Section 4.2 the first step is to send an ephemeral DH public key and the long-term DH public key in plaintext. To preserve anonymity an anonymous DH key exchange may be executed first, then the long-term public key can be hidden within the resulting anonymous encrypted channel.

Destination validation is implied by the broadcast sending key derivation requiring input from all intended recipients.

Since the consistency checks are performed after every broadcast message and only on a single message at a time, SYM-GOTR is *speaker consistent*, *causality preserving*, and maintains a *global transcript*. These consistency checks only contain information about a single broadcast, thus providing *message unlinkability*.

SYM-GOTR can be simulated by anyone who possess the long-term public keys of the participants, all encryption and signing keys used for communication are ephemeral.

Because of this the sessions have *participant repudiation* and *message repudiation*.

We now sketch in more detail proofs of the security properties specific to SYM-GOTR. Due to space constraints full proofs of the security properties are deferred to Appendix 4.6.

Our properties rely on standard security assumptions and we prove them in a series of games. We assume our authenticated encryption scheme is IND-CPA and INT-PTXT secure. We also assume NAXOS is secure, that is, an adversary cannot distinguish between a valid symmetric key generated with NAXOS and a random bit string. The last assumption we make is the Decisional Diffie-Hellman problem is hard. An adversary given (g^x, g^y, g^z) cannot distinguish between $g^z = g^{xy}$ or $g^z \leftarrow^R G$.

4.3.1 Confidentiality

Confidentiality is similar to the IND-CPA of symmetric encryption. To break the confidentiality of SYM-GOTR an adversary M must either learn the group sending key of a user or break the confidentiality of our authenticated encryption scheme. To learn the group sending key M must learn all of the inputs to KDF_4. These inputs are sent over the secure p2p channels, so to learn these inputs M must either learn the p2p channel keys(breaking NAXOS) or break the confidentiality of the p2p channels. If M cannot break the confidentiality of the p2p channels M must break the confidentiality of the broadcast channel encryption.

4.3.2 Message Integrity and Authentication

Message Integrity and Authentication are similar to the INT-PTXT game for symmetric encryption and integrity. If an adversary M can cause an honest party to accept a forged message from another honest party, M must be able to inject forged consistency checks into the secure p2p channel between the two honest parties. For M to forge p2p messages, M must either learn the p2p sending keys(by breaking NAXOS) or break the integrity of the authenticated encryption scheme.

4.3.3 Participant Consistency

Participant consistency relies on the integrity of the p2p channels. If two honest parties complete the setup phase with differing participant lists for the same session, an adversary M must have forged their p2p communication that sends the group view key. Similarly to message authentication, for M to forge p2p channel message M must learn the p2p sending keys or break the integrity of the authenticated encryption scheme.

4.3.4 Forward and Backward Secrecy

Forward and Backward Secrecy are the properties that if the state of an honest participant is leaked a limited number of messages are compromised. In SYM-GOTR forward and backward secrecy rely on the confidentiality of our authenticated encryption scheme and the Decisional Diffie-Hellman Assumption(DDH). To break forward and backward secrecy an adversary must either break the confidentiality of the symmetric encryption scheme or learn the Diffie-Hellman shared secrets used to compute the symmetric keys of the p2p channels.

4.3.5 Participant Repudiation

We describe participant and message repudiation in a similar manner to that of Raimondo, Gennaro, and Krawczyk [42] deniable authentication and key exchange. We show that SYM-GOTR is deniable because a session can be simulated to produce a transcript that is identically distributed to a transcript of a real SYM-GOTR session. All simulators are simple extensions to the simulator for the p2p deniable AKE.

We define participant repudiation under two threat models. One where the adversary has knowledge of the long-term secret key of corrupt participants and one where the judge only knows the public keys. The adversary produces a protocol transcript T by executing SYM-GOTR with a single honest party. A simulator S takes as input T and the adversary's long-term secrets s for session T. S generates a protocol transcript T' that includes the honest party where $T \neq T'$. A judge that takes as inputs T^* and the adversary's secrets s outputs a single bit guess on the input being T or T'. Participant repudiation requires that for every adversary there is a simulator such that no judge can distinguish between T and T'.

A sketch of the simple simulator S follows. The simulator runs the SYM-GOTR protocol using the secrets of the corrupt parties and generating random values for the honest party as described in the protocol. The simulator uses the user identifier and long-term public key of the honest user. Since NAXOS is a deniable AKE it can be simulated. We omit the details here, but it is a simple 3-DHE protocol. The output T' is identically distributed to T, so the judge has no advantage.

4.3.6 Message Repudiation

Message repudiation is similar to participant repudiation, except that the adversary produces a chat transcript τ. The protocol transcript T is produced by running a SYM-GOTR session with an honest party. The simulator's input is the same as the participant repudiation simulator and also includes the chat transcript and produces T' to be a protocol transcript of τ. The judge takes τ and T^* as input and guesses if $T^* = T$ or T'.

The simulator is almost identical to the simulator for participant repudiation. The message repudiation simulator executes the participant repudiation simulator then follows the protocol as described in Section 4.2 to produce a transcript for τ that is identically distributed to T. It is a simple extension to the participant repudiation simulator because all inputs of the message and signature consistency checks are ephemeral values generated during the SYM-GOTR session. Message repudiation is provided for the group conversation since the underlying p2p protocol provides message repudiation.

4.3.7 Message Unlinkability

Message unlinkability is the property that proving authorship of a message to a third party does not prove authorship of any other message to the third party. It is similar to message repudiation and is achieved in SYM-GOTR due to the fact that any communication in SYM-GOTR relates to only a single broadcast. The message consistency check only includes the digest of the current broadcast and no information about the rest of the chat transcript. SYM-GOTR also maintains forward and backward secrecy for every chat message, so no broadcast messages are encrypted or signed with the same keys.

More formally assume an adversary can convince a judge that a message was authored by a user. This provides the judge with knowledge of the ephemeral symmetric key inputs and ephemeral public verification keys used to encrypt and sign the message and the consistency checks, i.e. the state of the adversary. Message unlinkability is provided if the adversary cannot prove these keys are linked to another message and the messages consistency checks. To show an adversary cannot prove linkability two simulators are needed. One that produces a transcript up until and including the compromised message that is identically distributed to that of the real transcript. This simulator provides unlinkability for all previous messages. The other simulator produces a transcript that is identically distributed to that of the real transcript including and after the compromised message. This simulator provides unlinkability for all messages after the compromise. We quickly sketch the simulators as they are almost identical to the simulator for message repudiation.

Both simulators take as input a chat transcript τ and protocol transcript T. For the first simulator the transcripts are all messages before the compromised message. The second simulator transcripts are for all messages after the compromised message. The simulators also takes as input c the ciphertext of the message and *state* the state of the adversary including all $s_i, vk_i, K_{ij}, K_{ji}$ and keys used to ratchet forward all of the p2p channels.

The first simulator simply executes the message repudiation simulator for all messages before the compromise and uses the adversary's state for all ephemeral keys shared in the last consistency check before the compromise. The second simulator executes the message repudiation simulator using the compromised state as the keys for the first message. Both simulators simply execute the protocol with the adversary's provided inputs for the single compromised message. These simulators produce protocol transcripts T' that are identically distributed to that of a real transcript and contain the compromised message and state.

Message repudiation and unlinkability is trivial when all operations on messages use ephemeral keys distributed over deniable and unlinkable p2p channels and all protocol operations only operate on inputs provided by a single message.

4.3.8 Global Transcript

We define global transcript as all participants agreeing on the order of all broadcast messages sent during a chat, and the guarantee that all participants see every broadcast. An adversary cannot convince a subset of the participants to accept a message without being caught.

A global transcript is provided by construction of the protocol only processing a single message at a time. The two part consistency check and the ephemeral signing keys guarantee message receipt and display. A message is not accepted until it has been approved by all participants, indicating all participants have received the message. For a message to be rejected by a subset of participants, an adversary would have to send a reject message to them signed with the adversary's ephemeral signing key. The signature consistency check performed next would reveal the adversary has signed two messages, one accepting and one rejecting the broadcast. Our protocol does not guarantee all participants have seen a broadcast under strong adversarial conditions but does allow us to identify the adversary and notify the users that not all participants have seen the previous broadcast. We can guarantee that all users have seen the broadcast under the covert model.

4.3.9 GOTR Improvements

We quickly discuss the differences between SYM-GOTR, GOTR, and Signal in terms of security properties.

Participant Consistency exists in SYM-GOTR due to the participant consistency check during group setup. GOTR and Signal do not provide a similar mechanism and are vulnerable to participants not sharing the same view of the group.

Participant Repudiation is not provided by GOTR as described in [17]. The authors claim the p2p channels do not need to be deniable. Only that they are confidential and authenticated channels. If GOTR is implemented with deniable p2p channels then the protocol provides participant repudiation.

Message Unlinkability only partially exists in GOTR. The broadcast messages of GOTR contain the digest of the chat transcript thus far. This builds a chain of all the broadcast messages. Once authorship of a single message is proven acceptance of all

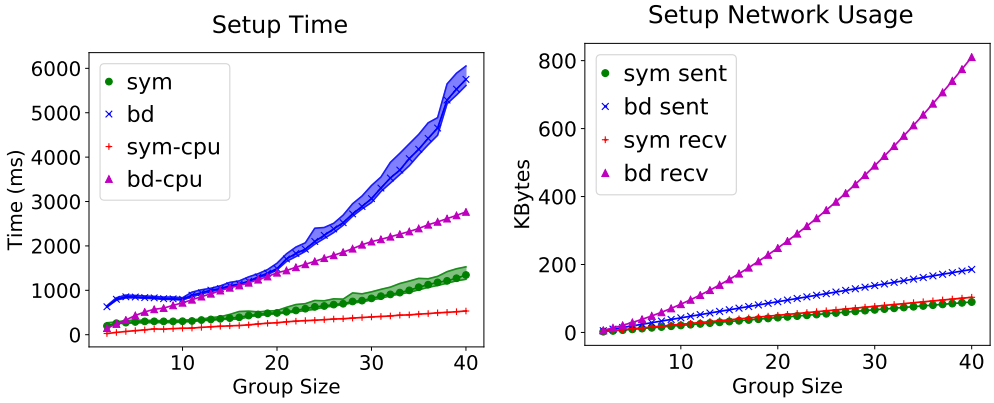


Figure 4.1: The time (25^{th}, 50^{th}, and 90^{th} percentile) and network traffic to set up a secure chat room with SYM-GOTR and GOTR.

previous messages is proven as well as authorship of all messages using the same keys.

Global Transcript is not clear in GOTR. The GOTR protocol relies on a transcript consistency check but does not describe how it should be implemented or the properties it provides. SYM-GOTR provides a global transcript due to our two part consistency check and can identify an adversary via the signatures on the consistency check message. Signal does not make any guarantees about a global transcript.

Forward and Backward Secrecy is only partially provided by GOTR. GOTR only ratchets keys when requested. In comparison SYM-GOTR ratchets all keys on every message with little additional overhead. Due to Signal assuming an asynchronous model it cannot provide forward and backward secrecy for every message.

4.4 Performance Evaluation

We implemented the SYM-GOTR protocol as a Java library and a plugin for the Jitsi IM client. We measured the performance using the library and a $10/month virtual private server hosted on linode [11]. Our clients ran across a cluster of ten machines each with an Intel i7 3.4GHz Quad-Core processor and 32 GB of RAM. Our server was an ejabberd XMPP server. The server had a 1 Gb network connection and the clients shared a 1 Gb connection. The round trip time between the clients and server was approximately 100 ms. Care needs to be taken when choosing a communication service

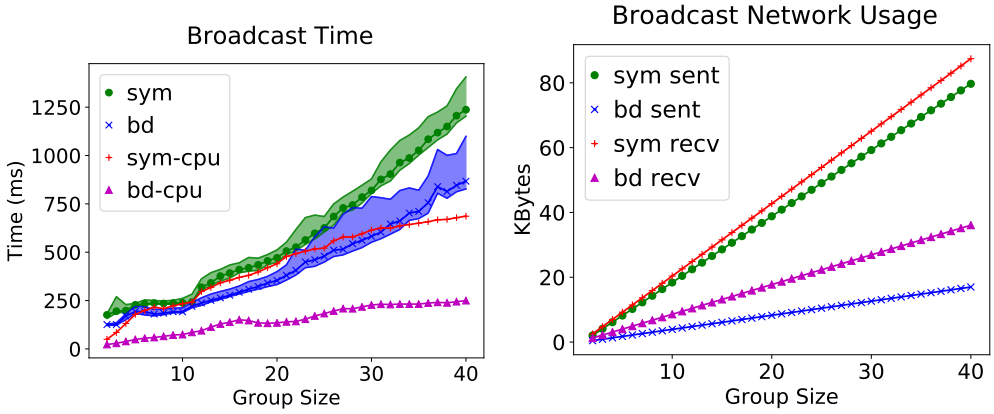


Figure 4.2: The time (25^{th}, 50^{th}, and 90^{th} percentile) and network traffic to broadcast a message to a secure group chat.

due to SYM-GOTR benefiting from consistent ordering of broadcast messages. The two common group messaging standards; XMPP and IRC do not enforce this property, but some server implementations provide consistent ordering.

For performance analysis we consider groups of a practical size to enforce authentication between all participants. We base our groups sizes on analysis of social graphs in the Facebook social network [43]. We would like to know the average clique size for a set of friends. If the participants did not form a clique they would not be able to pairwise authenticate, negating the security provided by SYM-GOTR. The results of the study did not reveal the clique size but did discuss the median degree of a user and a local clustering coefficient. The median degree of a user is 100 with a local clustering coefficient of 14%. This implies that the maximum clique size for an average user is $\sim$37.

For comparison we also implemented GOTR with a single round consistency check executed after every broadcast. The GOTR paper does not describe the details of the consistency check or how often to perform it. We executed 100 runs of each operation for both SYM-GOTR and GOTR. To coexist with current XMPP servers and clients when SYM-GOTR users would like to start or join a SYM-GOTR session they send an insecure broadcast message to inform the group. Clients not supporting SYM-GOTR will simply display the message while clients supporting SYM-GOTR will broadcast their support and proceed to set up secure p2p channels with other supported clients.

This adds a small amount of overhead which describes the lack of symmetry between received and sent network traffic during setup and participant changes.

4.4.1 Setup

For each round of secure chat setup we created a Multi-User-Chat (MUC) with n participants then initiated the GOTR protocol. After the protocol reached the secure state we shutdown the protocol. We measured the cost as the time it takes from initiating the protocol to reaching the secure state.

Figure 4.1 compares the cost of setting up a secure chat session with n users. The number of messages a user sends during the setup phase is $O(n)$ for both SYM-GOTR and GOTR. SYM-GOTR sees a significant improvement in time and network traffic as a result of constant sized messages compared to message sizes of $O(n)$ with GOTR.

We also analyzed the performance of participant modification events in established SYM-GOTR conversation. For 10, 20, 30, and 40 participants SYM-GOTR required approximately 275, 420, 660, and 980 ms to add a participant and 150, 305, 545, and 780 ms to remove a participant. The network overhead is similar to that of setting put the conversation.

4.4.2 Broadcast

We measured the cost of a broadcast message to be sent and a consistency check performed by all participants in the group. We wait for the signature consistency check to complete providing a full representation of the time and network overhead of SYM-GOTR. We first set up a secure chat of n users, then we start our measurements and instruct a single client to broadcast a message. Once the message has been received and both consistency checks have completed we end our measurement. We perform this 100 times for groups of each size. The same measurement is performed for GOTR but there is only a single consistency check.

Figure 4.2 shows a group of 30 users takes 820ms and 65 KB of network traffic down to perform a secure broadcast with SYM-GOTR. GOTR is faster for broadcast operations due to a weaker single round consistency check that does not offer message unlinkability or digest accountability. It requires only sending $n-1$ p2p messages where

as SYM-GOTR requires $2(n-1)$. The difference is small enough to be worth the additional properties. This difference is also not noticeable to the user due to broadcasts being displayed after the message consistency check. The signature consistency check only warns the user on failures which should not normally occur. The message consistency check of SYM-GOTR and GOTR have the same size and complete in the same time. The network overhead of GOTR comes from broadcast messages being $O(n)$ in size.

4.4.3 CPU Usage

The plots also include the median CPU time of each operation. Finally we analyze the CPU time of each operation. All operations have p2p pairwise communication that execute in parallel but the total CPU time is represented in the plots. The "Setup" and "Add" operations are cheaper for SYM-GOTR due to the symmetric group key agreement. The main computation expense for SYM-GOTR "Broadcast" is due to the signing, verification, and signature key generation. The "Remove" CPU cost is less for GOTR since existing users do not need to generate new group keys.

4.4.4 Complexity

Due to lack of control of the Signal network we cannot produce a meaningful comparison of SYM-GOTR and Signal but we do provide an asymptotic comparison of each operation. In comparison to Signal, SYM-GOTR requires sending more p2p messages to offer the additional consistency properties. Signal does not attempt to offer group consistency which allows for efficient setup and participant changes. Signal also does not guarantee message consistency or ordering allowing for cheaper message broadcasts. Table 4.1 shows the asymptotic complexity for each operation under SYM-GOTR, GOTR, and Signal.

4.4.5 Practical Example

To show SYM-GOTR performs well in practice, we simulated one year's worth of IRC meetings from the OpenStack High Availability group [44]. We simulated all meetings from 2016. There are 38 meetings between January 04, 2016 and December 21, 2016.

Table 4.1: Asymptotic complexity for each operation. The top line is the size of the broadcast message and the bottom is the maximum number of p2p messages sent by an individual. All p2p messages are of constant size. The last two columns represent the computational complexity of the operation.

	Setup	**Broadcast**	**Add**	**Remove**	**Sending**	**Receiving**
SYM-GOTR	0 $2(n-1)$	1 $2(n-1)$	0 $2(n-1)$	0 $(n-1)$	$O(n)$	$O(n)$
GOTR	$8n$ $5(n-1)$	$8n$ $(n-1)$	$8n$ $5(n-1)$	$8n$ 0	$O(n)$	$O(n)$
Signal	0 $(n-1)$	0 $(n-1)$	0 $(n-1)$	0 $(n-1)$	$O(n)$	$O(1)$

Each meeting had on average 5 participants with 127 messages and lasted 35 minutes. We first replayed the meeting log without SYM-GOTR to record statistics under our network conditions. Then we replayed the meeting logs using SYM-GOTR. SYM-GOTR introduced an average delay of 127 milliseconds to display a message which extended the conversation by 16 seconds on average. This experiment demonstrates a realistic deployment of SYM-GOTR and shows it is practical for day-to-day use.

As expected, participants did not join or leave during the meetings. Churn is an uncommon occurrence in these synchronized messaging applications. We analyzed the churn of the OpenStack Ansible channel. We choose the Ansible channel because outside of meetings the High Availability channel is generally idle. For the year of 2016 there was on average 539 join and leave events per day in the channel with 137 seconds before and after each event and the next event or message. This shows churn is relatively uncommon and our performance is practical.

4.5 Discussion

Implementing a secure messaging application is difficult to do in practice even if the underlying protocol is secure. We quickly discuss the challenges faced by these applications and layout future work.

4.5.1 Usability

From a usability perspective developers attempt to accommodate unmotivated users. These users impose many challenges to secure software. Thinking only of user authentication there does not exist a consistent user interface to inform a local user that a remote user(friend) is who they claim to be. The problem becomes even more difficult for messages. A user can receive a message from an unauthenticated user or an authenticated user. The distinction must be apparent in the user interface and also consistent for messages received prior to authentication. In group communication sessions not all users may have seen a given message. This is also important information that may be relevant to the participants. A developer must consider these security concerns when designing the user interface but must also not add a bright red error for every situation. This will only cause warning fatigue and encourage users to ignore the security concerns the application intends to address.

4.5.2 Key Verification

Key verification is another challenge without a clear solution. In terms of SYM-GOTR we choose to utilize the Socialist Millionaire Protocol allowing users to authenticate each other using a pre-shared secret. We do not claim this is the best or only mechanism for key verification. Any other technique could be deployed with SYM-GOTR. Another existing system attempting to address this problem is keybase.io which links a users online identity to a long-term public key by connecting it to a users Twitter [45], reddit [46], facebook [47], or a handful of other online accounts. Other solutions include a web-of-trust, Public Key Infrastructure, or a Password Authenticated Key Exchanges (PAKE).

4.6 Proofs of Security

We prove the properties of SYM-GOTR with a series of games approach. All games have an Initialize function that sets up the game and a Finalize function that returns the result of the game. All other functions of the games perform a specific operation.

Figure 4.3: IND-CPA Game

function INITIALIZE(l)
    $k \leftarrow^R \{0,1\}^l$
    $b \leftarrow^R \{0,1\}$
function LR(M_0, M_1)
    $c \leftarrow Enc_k(M_b)$
    return c
function FINALIZE(d)
    return $(d = b)$

Figure 4.4: INT-PTXT Game

function INITIALIZE(l)
    $k \leftarrow^R \{0,1\}^l$
    $S \leftarrow^R \{\}$
function ENC(m)
    $c \leftarrow Enc_k(m)$
    $S\ gets S \cup \{m\}$
    return c
function VF(c)
    $M \leftarrow Dec_k(c)$
    if $m \neq \perp$ and $m \notin S$ **then**
        $win \leftarrow true$
    return $(m \neq \perp)$
function FINALIZE(d)
    return win

Figure 4.5: NAXOS Game

function INITIALIZE(U)
  $b \leftarrow^R \{0,1\}$
  Initialize PKI for all users in U.
function SEND($A, B, comm$)
  Send $comm$ to A on behalf of B
  return A's response
function LONG-TERM KEY REVEAL(A)
  return Long-term key of A
function EPHEMERAL KEY REVEAL(sid)
  return Returns the ephemeral key of a possibly incomplete session si.
function REVEAL(sid)
  return Session key of completed session sid
function TEST(sid)
  if $b = 1$ **then**
    $C \leftarrow$REVEAL(sid)
  else
    $C \leftarrow^R \{0,1\}^l$
  return C
function FINALIZE(d)
  return $(d = b)$

Figure 4.6: DDH Game

function INITIALIZE
  $b \leftarrow^R \{0,1\}$
  $x \leftarrow^R Z_p*$
  $y \leftarrow^R Z_p*$
  if $b = 0$ **then**
    $z \leftarrow x * y$
  else
    $z \leftarrow^R Z_p*$
  return (g^x, g^y, g^z)
function FINALIZE(d)
  return $(d = b)$

4.6.1 Assumptions

We base our proofs of the security properties of SYM-GOTR on standard security assumptions under the Random Oracle Model.

First we assume our authenticated encryption scheme (Encrypt-Then-MAC) is IND-CPA and INT-PTXT secure as shown by Bellare and Chanathip [39].

Figure 4.3 details the IND-CPA game. An adversary M is said to win the game if M can determine which of two chosen plaintext has been encrypted. The advantage of M is defined as $Adv^{IND-CPA}(M) = Pr[Mwins] - \frac{1}{2}$.

Figure 4.4 details the INT-PTXT game. An adversary M is said to win the game if M can forge a valid ciphertext of a plaintext that has not been queried. The advantage of M is defined as $Adv^{IND-PTXT}(M) = Pr[Mwins]$.

We also assume the NAXOS AKE is secure in the game detailed in [40]. The authors of NAXOS define an extended Canetti-Krawczyk [48] model (eCK) to prove NAXOS secure under a stronger adversary.

Figure 4.5 describes the NAXOS AKE game. An adversary M is allowed to create multiple sessions and reveal the long-term, ephemeral, and session keys of users. Defining Session IDs is a important part of key agreement models. Our definition is consistent with the eCK model. That is the Session ID of a p2p session is defined as the transcript of messages sent between the parties. This p2p Session ID is different from SYM-GOTR group Session IDs. The adversary wins the game if M can distinguish between a valid NAXOS session key and a random bit string if both parties are not compromised. The advantage of M is defined as $Adv^{NAXOS}(M) = Pr[Mwins] - \frac{1}{2}$.

We assume the Decisional Diffie-Hellman(DDH) problem is hard. Figure 4.6 describes the DDH problem in terms of a game. An adversary wins the DDH game if given (g^x, g^y, g^z) the adversary can distinguish between $g^z = g^{xy}$ and $g^z \leftarrow^R G$. The advantage of an adversary M is defined as $Adv^{DDH} = Pr[Mwins] - \frac{1}{2}$.

Finally, since SYM-GOTR does not require a specific key verification mechanism we assume a trusted PKI in the proofs. We assume all users have secure access to the PKI and verify the long-term public key of remote users for every secure p2p session. If the long-term key sent during channel setup is incorrect the session terminates. In all of our games we allow the adversary to corrupt a user and reveal the long-term private key.

Figure 4.7: SYM-GOTR Game Functions

function SetupSession(Sid, U_i, U)
Setup session Sid with U_i for group U
return U_i's response to session setup

function AddUser(Sid, U_i, U_j)
Add user U_j to session Sid of user U_i
return U_i's response to adding U_j.

function RemoveUser(Sid, U_i, U_j)
Remove user U_j to session Sid of user U_i
return U_i's response to removing U_j.

function Broadcast(Sid, U_i, m)
U_i sends m securely in session Sid.
return U_i's network traffic to send the message

function RevealLongTermKey(U_i)
return U_i's long-term key

function RevealSessionState(Sid, U_i)
return U_i's session state for session Sid.

function Send($U_i, U_j, comm$)
U_i sends $comm$ to U_j
return U_j's response to $comm$

4.6.2 Model

The SYM-GOTR security properties are proved with the following games. The games contain common operations setup, adding/removing users, and sending messages. These functions all return the network traffic generated by the session. In an active network adversary game these functions do not send the traffic to the users, instead they return the traffic and await for it to be sent by the adversary with the Send function. Send will return any new network traffic generated from receiving the input. Under a passive adversary the SYM-GOTR operation functions perform the entire operation and return all network traffic.

These games also have reveal functions when appropriate. These functions reveal information to the adversary e.g. long-term keys or session state. We define a corrupt user to be a user whom has had the long-term secret key revealed by the adversary. When revealing session state for U_i in session Sid the adversary learns all session state from line 3 of Algorithm 1 along with all of the p2p ephemeral key material in use, that is U_i's current ephemeral secret key and the remote parties ephemeral public key.

Figure 4.8: SYM-GOTR Confidentiality Game

function INITIALIZE(U)
  $b \leftarrow^R \{0,1\}$
  Initialize PKI for all users in U.
function TEST(Sid, U_i, M_0, M_1)
  U_i sends M_b in session Sid.
  return U_i's network traffic to send the message
function FINALIZE(d)
  return $(d = b)$

Figure 4.7 describes the functions that are consistent between all SYM-GOTR games.

4.6.3 Confidentiality

The confidentiality property states a message may only be read by a conversation participant. This is equivalent to the indistinguishable chosen plaintext game. Figure 4.8 describes the additional functions of the confidentiality game for SYM-GOTR. The TEST function allows the adversary to force U_i to send a test secure broadcast in session Sid. The adversary may then guess the bit b by calling FINALIZE. A session is defined as clean if the adversary has not revealed the long-term key or the session state for any participant in the session. The adversary wins if the session is clean and the guess is correct. The advantage of adversary M is defined as $Adv^{conf}(M) = Pr[Mwins] - \frac{1}{2}$.

SYM-GOTR is confidential if all hash and key derivation functions are modeled as random oracles. For any confidentiality adversary M that runs in time at most t, establishes at most s sessions with at most w users per session and establishes at most n p2p channels. We show that there exists a NAXOS adversary N, an IND-CPA adversary P_0, and an IND-CPA adversary B such that

$$Adv^{conf}(M) \leq n \cdot Adv^{NAXOS}(N) + n \cdot Adv^{IND-CPA}(P_0) + sw \cdot Adv^{IND-CPA}(B)$$

Where N, P_0, and B run in time $O(t)$.

Proof. The adversary M can win the confidentiality game in two ways. They can learn the group sending key of the target user or win the IND-CPA game against our authenticated encryption. Since KDF_3 is modeled as a random oracle to learn the group

sending key, M must query the random oracle with the same inputs as the sending user. These inputs are shared over the secure p2p channels. For M to learn these inputs M must either brake the NAXOS protocol or win the IND-CPA game against our authenticated encryption. We construct an adversary N that can win the NAXOS game and an adversary P_0 that can win the IND-CPA game of the p2p channels and an adversary B that can win the IND-CPA game against the broadcast ciphertext given an adversary M that can win the confidentiality game.

First, we assume M wins by computing the p2p keys. M can then decrypt the p2p traffic and compute the group sending key. Using the group sending key M can trivially win the confidentiality game. For M to compute the p2p keys M must break NAXOS. We can use M to construct an adversary N that wins the NAXOS game. N is defined as follows, N behaves as a normal SYM-GOTR confidential challenger except during p2p channel setup. During initialization N initializes a NAXOS game. When N would normally setup a NAXOS session at line 2 of Algorithm 1, N sends the start communication to the NAXOS game between U_i and U_j. When M SENDs p2p channel setup communication to N, N forwards it along to the Naxos game. N chooses one session at random invokes $K_{ij} \leftarrow$ TEST on NAXOS session between U_i and U_j to retrieve a test p2p key for the session. When M calls REVEALLONGTERMKEY N returns LONG-TERM KEY REVEAL. When M calls REVEALSESSIONSTATE N returns both EPHEMERAL KEY REVEAL and REVEAL of the NAXOS game. N executes REVEAL on the other NAXOS sessions to learn their p2p keys. N continues the SYM-GOTR protocol as normal. For M to compute the p2p key for a pair of user (U_a, U_b M must have queried the KDF_1 random oracle for $(epk_b^{lsk_a}, lpk_b^{h_a}, epk_b^{h_a}, U_a, U_b)$. N watches M's random oracle queries and checks if any of the outputs match K_{ij} if so N guesses 1 else N guesses 0. N has probability $\frac{1}{n}$ of guessing the correct p2p session. The advantage is $Adv^{conf}(M) \leq n \cdot Adv^{NAXOS}(N)$.

The second game assumes M cannot break NAXOS but instead can learn the inputs to the group sending key derivation function. We now describe a challenger P_0, given M, that acts as an adversary to the IND-CPA game. P_0 behaves as a normal confidentiality challenger and INITIALIZEs an IND-CPA game for a random p2p session between U_i and U_j. P_0 generates two sets of group secrets$(lsecrets, rsecrets)$ for U_i. When U_i shares the secrets over the p2p channel to U_j, P_0 makes a LR$(lsecrets, rsecrets)$ query to

Figure 4.9: SYM-GOTR Authentication Game

function INITIALIZE(U)
$b \leftarrow^R \{0,1\}$
Initialize PKI for all users in U.
function FINALIZE()
return *true* if a message m' was displayed by U'_j with author U'_i in session Sid' that was not broadcast with BROADCAST(Sid', U'_i, m') and U'_i and U'_j are honest.

the IND-CPA game and sends the response to the remote peer. P_0 then watches M's queries to the KDF_3 random oracle. M must query the output on either the left or right secrets. P_0 then guesses left or right based on M's oracle query and wins with $Adv^{conf}(M) \leq n \cdot Adv^{IND-CPA}(P_0)$. P_0 has probability $\frac{1}{n}$ of guessing the right p2p session.

If all p2p traffic appears random to M, he must be able to win the IND-CPA game with our authenticated encryption scheme over the broadcast channel. We construct a challenger B that plays the confidentiality game with M and wins the IND-CPA game. B behaves as a normal SYM-GOTR challenger and initializes an IND-CPA game during initialization for a random user U_i of a random session Sid. When M calls TEST(Sid, U_i, M_0, M_1) B executes $c \leftarrow \text{LR}(M_0, M_1)$. B uses c as the ciphertext to broadcast M_b. When M guesses d, B guesses d to the IND-CPA challenger. B will win with advantage $Adv^{conf}(M) \leq sw \cdot Adv^{IND-CPA}(B)$. B has probability $\frac{1}{sw}$ of guessing the correct user and session.

If M cannot distinguish between the p2p messages or the broadcast messages and random. The network traffic must be independent of b. □

4.6.4 Integrity and Authentication

Integrity and message authentication are captured in the same game. Integrity is the property that all messages displayed were not modified in transit and message authentication is the property that participants agree on the authorship of a message.

Figure 4.9 describes the additional functions of the game that captures the integrity and authentication properties. The game is similar to the game for confidentiality but differs in the BROADCAST function. The adversary can ask user U_i to broadcast a message securely to session Sid. The adversary wins the game if an honest user U'_j displays a message m' from an honest user U'_i in session Sid' where BROADCAST(Sid', U'_i, m')

was not invoked by the adversary. Users are said to be honest if they have not had their long-term key or session state revealed for session Sid'. The advantage of adversary M at wining the message authentication game is defined as $Adv^{int}(M) = Pr[Mwins]$.

SYM-GOTR provides integrity and message authentication if all hash and key derivation functions are modeled as random oracles. For any authentication adversary M that runs in time at most t, establishes at most s SYM-GOTR sessions, and at most n p2p channels, we show that there exists a NAXOS adversary N and an INT-PTXT adversary P_1 such that

$$Adv^{auth}(M) \leq n \cdot Adv^{NAXOS}(N) + n \cdot Adv^{INT-PTXT}(P_1)$$

Where N and P_1 run in time $O(t)$.

Proof. Since all users perform a message consistency check over the p2p channels (line 6, Alg. 1 for M to win the authentication game M must learn U_i' and U_j''s p2p sending keys or win the INT-PTXT game against the p2p authenticated encryption scheme. Similarly to confidential for M to learn a p2p channels sending keys M must be able to win the NAXOS game. Given M, the NAXOS adversary N described above can win the NAXOS AKE game.

If M cannot break NAXOS M must produce a valid ciphertext under (U_i', U_j') p2p sending keys. We construct P_1 an authentication challenger that can win the INT-PTXT game given M. P_1 acts as a normal SYM-GOTR challenger. When setup is complete P_1 initializes an INT-PTXT for a random p2p session between U_i and U_j. When a protocol message m is to be sent over the p2p channel from U_i to U_j P_1 invokes $c \leftarrow \text{ENC}(m)$ of the INT-PTXT game and returns c as the ciphertext. For M to win M must send a ciphertext c' from U_i to U_j where $c' \neq c$. P_1 submits this to $\text{VF}(c')$ of the p2p INT-PTXT game. P_1 wins the INT-PTXT game if M wins the authentication game by forging valid secure p2p messages between U_i and U_j. The advantage of N is $Adv^{int}(M) \leq n \cdot Adv^{INT-PTXT}(P_1)$. If M cannot create a valid p2p message SYM-GOTR must provide integrity and authentication. □

Figure 4.10: SYM-GOTR Participant Consistency Game

function INITIALIZE(U)
$b \leftarrow^R \{0,1\}$
Initialize PKI for all users in U.
function FINALIZE(Sid, U_i, U_j)
return *true* if U_i and U_j are honest and have completed the setup phase with differing views of participants for session Sid.

4.6.5 Participant Consistency

The participant consistency property ensures that all users agree on the set of participants in a group conversation. Figure 4.10 describes the additional functions for the participant consistency game. The participant consistency game is similar to the previous games. The adversary M wins the game by producing a (Sid, U_i, U_j) 3-tuple where U_i and U_j are honest parties that have completed the setup phase(line 3 of Alg. 1) for session Sid with different sets of users. Users a honest if they have not had their long-term key revealed or session state for session Sid. The advantage of M is defined as $Adv^{part}(M) = Pr[Mwins]$.

SYM-GOTR provides participant consistency if all hash and key derivation functions are modeled as random oracles. For any participant consistency adversary M that runs in time at most t, establishes at most s SYM-GOTR sessions, and at most n p2p channels, we show that there exists a NAXOS adversary N and an INT-PTXT adversary P_1 such that

$$Adv^{part}(M) \leq n \cdot Adv^{NAXOS}(N) + n \cdot Adv^{INT-PTXT}(P_1)$$

Where N and P_1 run in time $O(t)$.

Proof. For M to win U_i must be sent a gv_j (line 9, Alg. 2) value that matches gv_i. M must either learn the p2p sending key of U_j or win the INT-PTXT game against our authenticated encryption scheme for the p2p channel between U_i and U_j. Adversaries N and P_1 described earliers function as valid adversaries given M to win the NAXOS game or INT-PTXT game of a p2p session. If M cannot learn the sending keys or cannot forge valid p2p messages, SYM-GOTR must provide participant consistency.

□

Figure 4.11: SYM-GOTR Perfect Forward Secrecy Game

function INITIALIZE(U)
  $b \leftarrow^R \{0,1\}$
  Initialize PKI for all users in U.
function TEST(Sid, U_i, m_0, m_1)
  U_i securely sends m_b in session Sid
  TEST may only be called once and must be called before REVEALSTATE
  return All network traffic generated for U_i to send m_b to session Sid
function REVEALSTATE(Sid, U_i)
  May only be called after TEST
  return All of U_i's current state for session Sid along with U_i's long-term key
function FINALIZE(d)
  return $(d = b)$

4.6.6 Perfect Forward Secrecy

Forward Secrecy is the property that any message sent prior to an honest user state reveal is secure against a passive adversary. Figure 4.11 details the additional functions of the forward secrecy game. All methods return the network traffic generated by the group to perform an operation. An adversary M may request an honest party to send a message with BROADCAST. When ready M may request U_i to send a test message with TEST to a clean session. After sending a test message M may query for the internal state of a user. M must then guess if the challenger sent m_0 or m_1. An adversary's advantage is defined as $Adv^{pfs}(M) = Pr[Mwins] - \frac{1}{2}$.

SYM-GOTR provides forward secrecy if all hash and key derivation functions are modeled as random oracles. For any adversary M that wins the forward secrecy game and runs in time at most t, establishes at most s sessions with at most w users per session, establishes at most n p2p sessions, and sends at most r p2p messages, we show that there exists a DDH adversary D, an IND-CPA adversary P_0, and an IND-CPA adversary B such that

$$Adv^{fs}(M) \leq r \cdot Adv^{DDH}(D_0) + n \cdot Adv^{IND-CPA}(P_0) + sw \cdot Adv^{IND-CPA}(B)$$

Where D_0, P_0, and B run in time $O(t)$.

Proof. For M to win the game he must either compute a previous sending key which

requires knowledge of previous p2p channel plaintexts. M may either compute previous p2p channel keys or break the confidentiality of our authenticated encryption scheme, M may also break the confidentiality of the authenticated encryption for broadcast messages.

If M computes previous p2p keys we can construct a challenger D_0 for the forward secrecy game that can solve the DDH problem if given an adversary that can win the forward secrecy game. D behaves as a normal SYM-GOTR challenger. For a random p2p message between U_i and U_j before TEST is invoked, D_0 requests a DDH challenge (g^x, g^y, g^z) and uses g^x as U_i's next key ratchet pubic key with g^y as U_j's next key ratchet public key. g^z is the input for the next key derivation function between the pair. For M to compute the p2p key used to share the next sending key inputs, M must query $KDF_2(g^h, U_i, U_j)$. D_0 watches M's random oracle queries, if $g^h = g^z$ D guesses $g^z = g^{xy}$. If M computes the p2p key $Adv^{pfs}(M) = r \cdot Adv^{pfs}(D_0)$. D_0 has probability $\frac{1}{r}$ of guessing the correct p2p message.

If M cannot compute previous keys M may break the authenticated encryption of the p2p channel, in which case adversary P_0 applies. Finally, if the p2p channels are secure M must break the authenticated encryption of the group messages adversary B from earlier can be used to win the IND-CPA game against our authenticated encryption scheme.

If the DDH problem is hard and our authenticated encryption scheme is secure, SYM-GOTR must be forward secure. □

4.6.7 Backward Secrecy

The backward secrecy property guarantees that after a users state is revealed only the next message is compromised by a passive adversary. Figure 4.12 describes the additional functions of the backward secrecy game. An adversary M may reveal the state of a user with the REVEALSTATE query. After revealing a state M may issue TEST to instruct U_i to send a message m followed by test message m_b to a clean session. The adversary then guesses if m_0 or m_1 was sent. The advantage of M is defined as $Adv^{bs}(M) = Pr[Mwins] - \frac{1}{2}$.

SYM-GOTR provides backward secrecy if all hash and key derivation functions are modeled as random oracles. For any adversary M that that wins the backward secrecy

Figure 4.12: SYM-GOTR Backward Secrecy Game

function INITIALIZE(U)
  $b \leftarrow^R \{0,1\}$
  Initialize PKI for all users in U.
function TEST(Sid, U_i, m, m_0, m_1)
  U_i first securely sends m then securely sends m_b in session Sid
  TEST may only be called once
  return All network traffic generated for U_i to send m then m_b to session Sid
function REVEALSTATE(Sid, U_i)
  May only be called before TEST
  return All of U_i's long-term key and current state for session Sid
function FINALIZE(d)
  return $(d = b)$

game and runs in time at most t, establishes at most s sessions with at most w users per session, and establishes at most n p2p sessions, we show that there exists a DDH adversary D_1, an IND-CPA adversary P_0, and an IND-CPA adversary B such that

$$Adv^{bs}(M) \leq n \cdot Adv^{DDH}(D_1) + n \cdot Adv^{IND-CPA}(P_0) + sw \cdot Adv^{IND-CPA}(B)$$

Where D_1, P_0, and B run in time $O(t)$.

Proof. For adversary M to win the backward secrecy game they must be able to compute the next sending key of U_i or win the IND-CPA game of the broadcast channel. If M can compute the next sending key M must either compute the next p2p channel keys or win the IND-CPA game against the p2p channels.

If M can compute the next p2p sending key we construct a challenger D_1 that can win the DDH problem. D_1 acts as a normal SYM-GOTR challenger before TEST is invoked. When TEST is invoked, D_1 securely sends m as normal and queries the DDH challenger for (g^x, g^y, g^z) for a random pair of users (U_i, U_j). D_1 uses g^x as U_i's next p2p channel public key and g^y for U_j. The next p2p channel sending key for U_i is computed as $K_{ij} = KDF_2(g^z, U_i, U_j)$. For M to learn K_{ij}, M must query the random oracle $KDF_2(g^{xy}, U_i, U_j)$. D_1 watches M's random oracle queries and if M queries $KDF_2(g^z, U_i, U_j)$, D_1 guesses $g_{xy} = g^z$ to the DDH challenger. D_0 wins if the correct p2p channel was selected. The advantage of D_1 is $Adv^{bs}(M) = n \cdot Adv^{bs}(M)$. D_1 has probability n of guessing the correct p2p session.

The adversaries P_0 and B discussed previously demonstrate how to construct a challenger that can win the IND-CPA game against the p2p channel and broadcast channel respectively given an adversary that can win the backward secrecy game. If M cannot compute the next p2p sending key or break the IND-CPA game of our authenticated encryption, SYM-GOTR must be backward secure. □

4.7 Usability

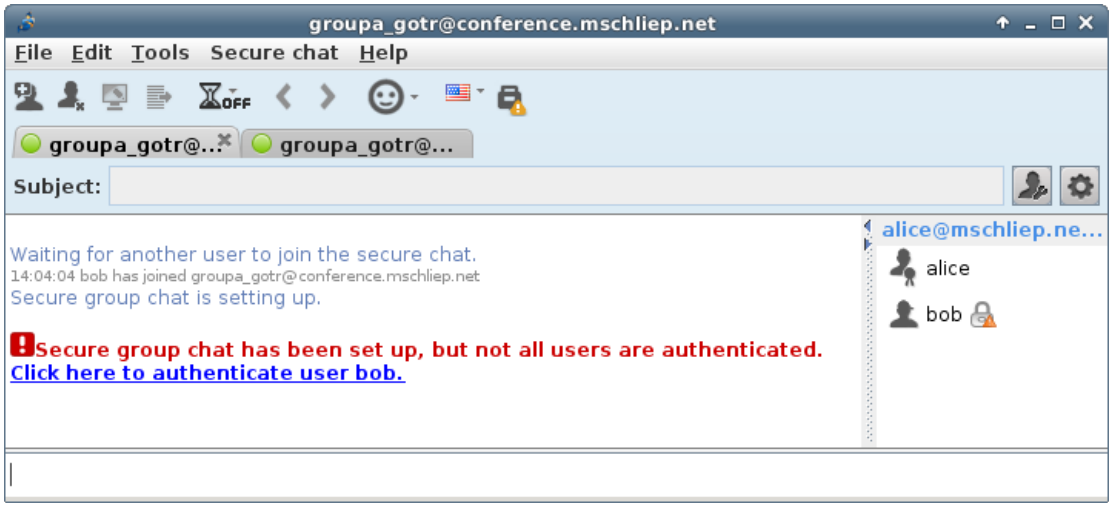


Figure 4.13: Warning displayed when not all users in a secure group chat are authenticated.

We evaluate the usability of our SYM-GOTR plugin. While implementing our plugin we focus on maintaining consistency with the overall Jitsi user experience while focusing on the safe usability of SYM-GOTR. We analyzed our implementation with a cognitive walkthrough [49] and resolved usability issue we discovered during the process. A cognitive walkthrough is a standard first step [50] used to evaluate the usability of software. Gujrati and Vasserman [51] show a cognitive walkthrough is a simple technique that can provide dramatic improvements to the usability of security software that they then validate with a user study.

During the cognitive walkthrough we focused on the four main actions a user will

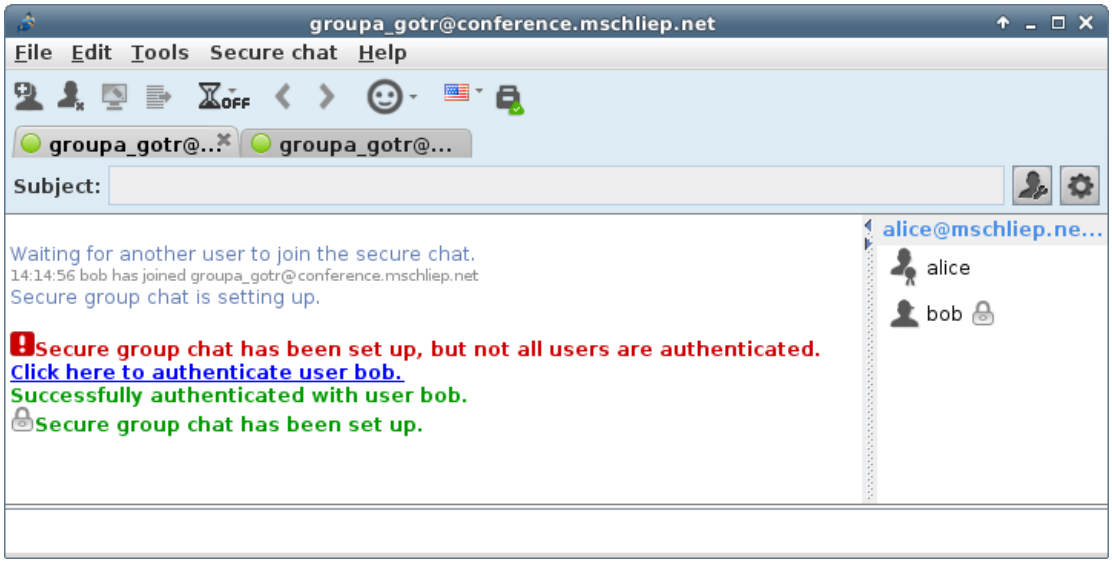


Figure 4.14: Notice of successful participant authentication.

perform with SYM-GOTR; secure group setup, group member authentication, receiving group messages, and sending group messages. In this section we present two key properties that need to consider the user experience when implementing SYM-GOTR. These properties are participant authentication and global transcript consistency.

We consider four usability properties during the walkthrough:

p1 The user should be able to achieve the correct effect.

p2 The user should be able to notice the correct action is available.

p3 The user should be able to associate the correct action with the effect he or she is trying to achieve.

p4 If the users performs the correct action, he or she should be able to see progress is being made towards it.

We also consider the security properties of the secure group communication.

s1 **Unmotivated users property** Users should be able to use the system with minimal additional effort.

Figure 4.15: Warning of participant authentication failure.

s2 Lack of feedback property Lack of feedback from a secure system will leave the user wondering if they have performed the right tasks and performed the task correctly.

s3 Barn door property If a user creates an insecure environment the security of the system may be compromised even if the environment is brought back to a secure state.

s4 Weakest link property Mistakes in the use of security software may lead the user to an insecure environment. Users should be guided to use the software correctly so the chance of failure is minimal.

We have adopted the usability and security properties from the seminal work of Whitten and Tygar [50]. Along with the usability and security properties of the system we want to consider the SYM-GOTR usability goals of the system.

1. Users understand the privacy provided (encryption, authentication).
2. Users understand the privacy of messages(plaintext, encrypted, authenticated).
3. Users understand the warnings when an adversary is present.

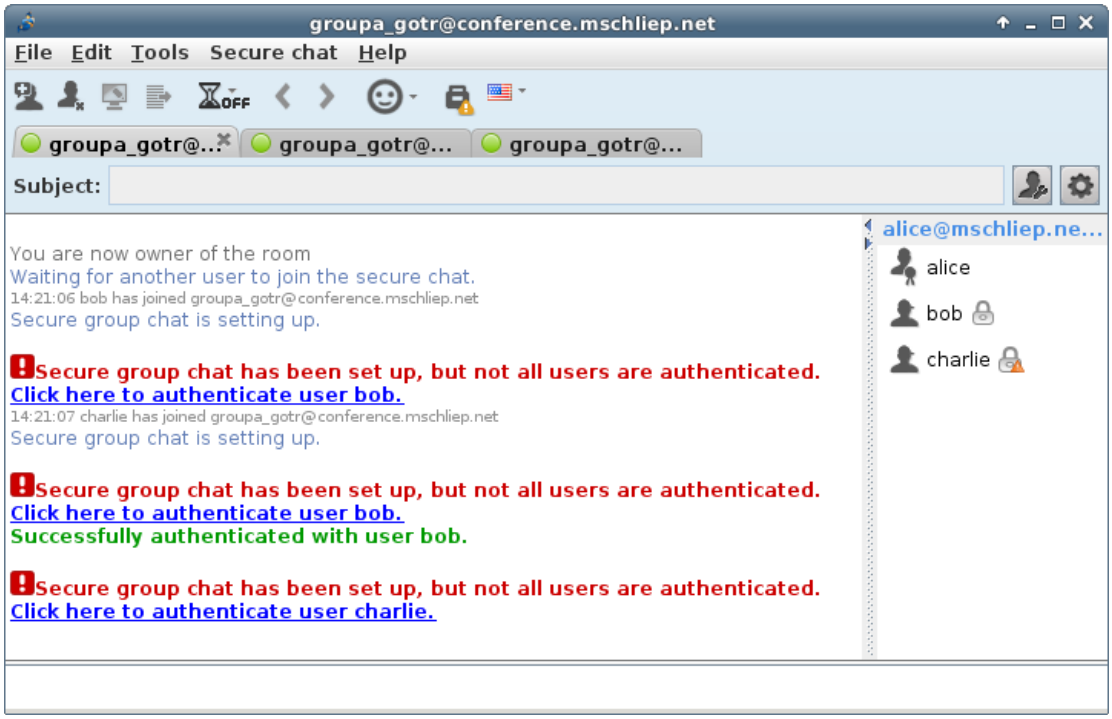


Figure 4.16: Warning that another participant requires authentication.

In our cognitive walkthrough we considered novice users, users with an interest in private communication but unwilling to read a manual. In the expected use case for SYM-GOTR, a user first starts a secure chat room. Then multiple users join the room. Finally secure messages are broadcast to the room. At anytime during the session users may join and leave the secure chat room.

We give users multiple consistent feedback indicators for each action, notice the lock icon on the toolbar and in the participant list and how these icon updates reflect the messages presented during each task.

4.7.1 Secure Group Setup

There are two ways a user can setup a private chat. They can initialize a chatroom that *requires* connected clients to implement SYM-GOTR or they can promote an unsecured chat to a SYM-GOTR chat. For this cognitive walkthrough we only describe our walkthrough of required SYM-GOTR chats and how it relates to participant authentication.

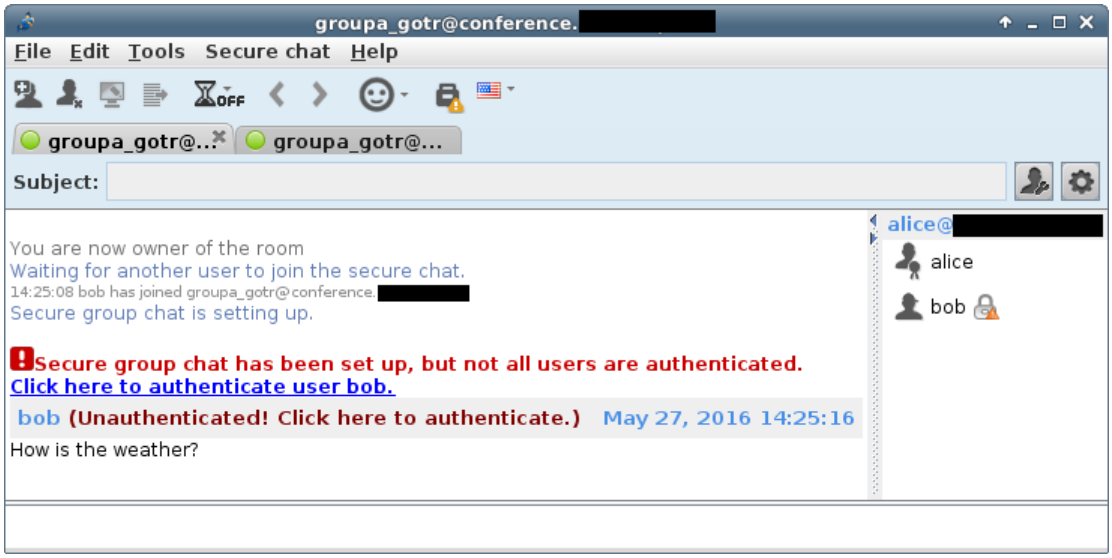


Figure 4.17: Warning shown when a user receives a message from an unauthenticated participant.

After two or more users have joined a secure chat room SYM-GOTR can perform the setup phase but the users remain unauthenticated. In maintaining the usability properties the user is alerted to this and provided with an action to authenticate the participants. Figure 4.13 is a screen shot of this alert.

After authentication the users are notified of the success: Figure 4.14, or failure: Figure 4.15. When more than two participants are in a secure chat the users are prompted to authenticate every user one at a time. Figure 4.16 demonstrates this process. This maintains the usability properties by providing a single correct action at a time.

4.7.2 Receiving Messages

There are three types of messages a user can receive. A plaintext message can only be received when a SYM-GOTR session is in the plaintext state. An unauthenticated message is a message that has been broadcast and been confirmed in the message consistency check but originated from an unauthenticated user. Figure 4.17 is a screen shot showing the warning displayed when an unauthenticated message is received. If a message has been confirmed in the message consistency check and was sent by an

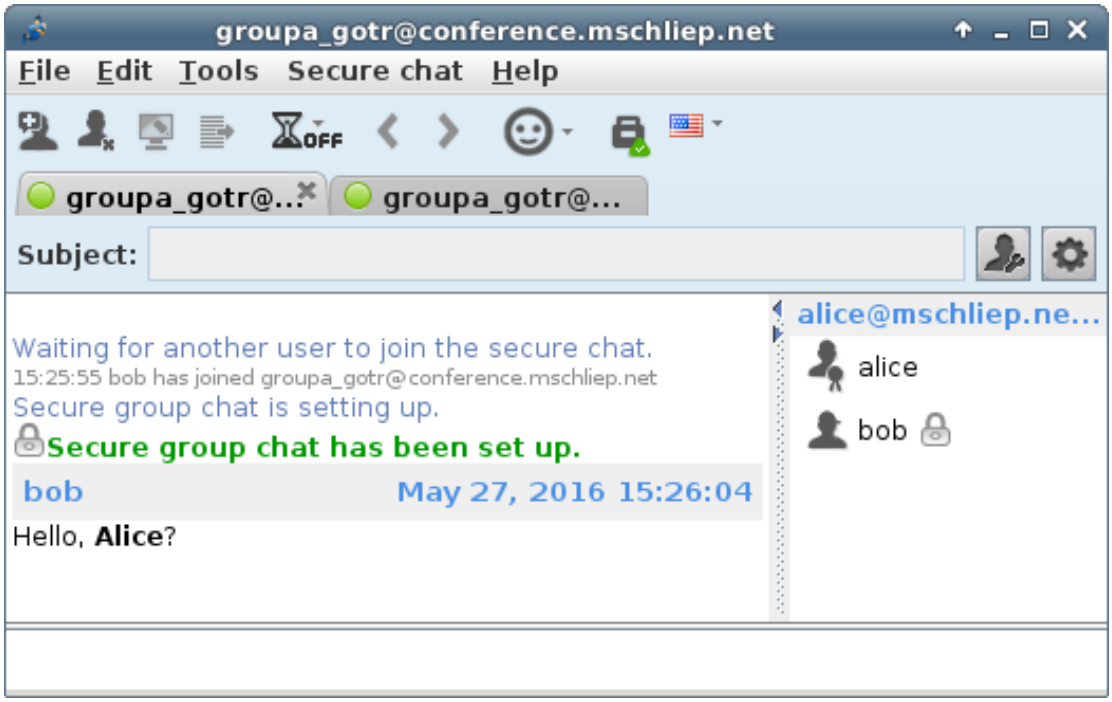


Figure 4.18: Notice when a user receives a secure group message.

authenticated user we consider it an authenticated message. When a user receives a message but it fails the digest exchange we silently drop the message. We choose to drop the message instead of warn the user to maintain the usability properties and a warning of this type is not actionable.

A signature consistency error may be detected in an adversarial environment which would allow an adversary to confirm a message reception with a subset of users only. This attack will be detected by the signature consistency check. The detection of this attack implies the previous message may not have been seen by all participants and the user is warned as shown in Figure 4.19. The adversary in this attack must generate two conflicting signed messages and will be noticed. We warn the user about this adversary so they can decide how to handle them.

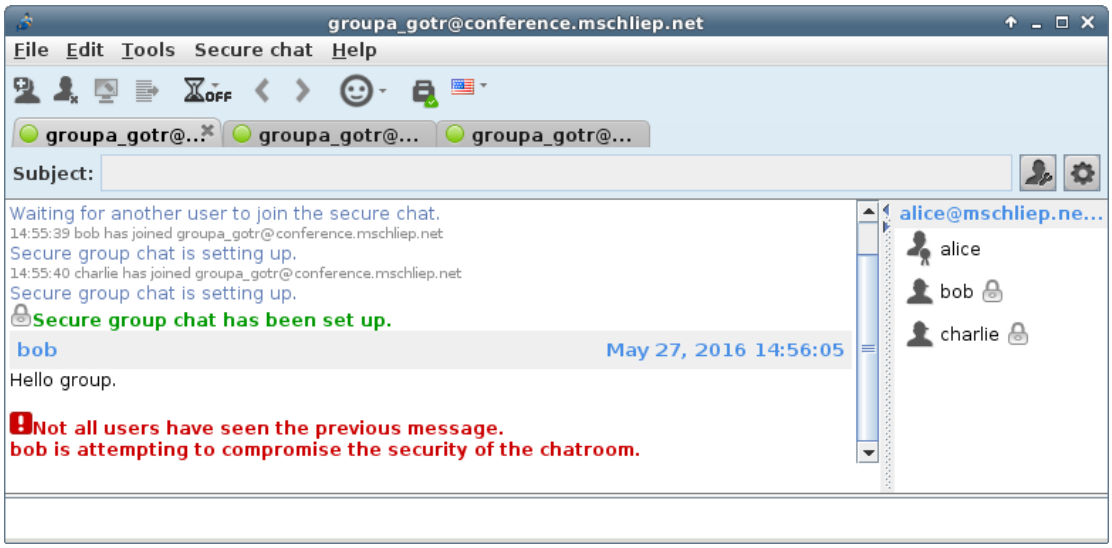


Figure 4.19: Warning displayed when a user detects a participant lied in the last digest exchange.

Chapter 5

Mobile Communication With Privacy and Integrity

In this chapter we address the problem of designing a deployable, end-to-end secure mobile group messaging application. We identify key constraints of the mobile end-to-end secure messaging model as well as describe the security properties a protocol should provide. We also identify a real-world threat model a protocol must provide these properties under (Section 5.1). We describe a relatively simple and provably secure protocol for Mobile Conversations With Privacy and Integrity (Mobile CoWPI) in Section 5.2. We show in Section 5.3 that Mobile CoWPI provides the desired security properties. We then analyze the security properties of our mobile messaging model and show the restrictions they impose on any mobile end-to-end secure messaging protocol (Section 5.5). We argue that under these restrictions, Mobile CoWPI is within a constant factor of optimal in terms of message size. Finally, We implement Mobile CoWPI as a Java server and library and show that it performs well in a realistic internet environment (Section 5.4) deployed on Amazon AWS[10] and Linode [11] with both desktop and Android [52] clients.

5.1 Mobile Secure Messaging

In this section we lay out the system model of modern secure messaging applications and show how this model is insufficient to provide conversation integrity. We then detail our system model and discuss how it enforces conversation integrity. We also overview our strong threat model along with all of the security properties we provide in our protocol.

5.1.1 Mobile Messaging Model

All popular mobile messaging applications provide the same core features using a consistent system model. The key feature is providing a conversation for two or more participants. These applications allow participants to start a new conversation, send messages, and add or remove participants from a conversation even while other participants are offline. When the offline participants return they are updated with all missed messages in the conversation. To improve conversation flow with offline participants the members of the conversation are notified when other participants have received the messages. This informs the author of a message not to expect a response until the recipients have received the message.

To provide these conversation properties the service provider handles routing and caching messages in the conversation. The messages are cached for delivery to offline participants. All popular secure messaging applications rely on a single service provider to perform the message routing and caching.

Unfortunately, if the server providing this service to a particular conversation is compromised, it can break the conversation integrity property[1] of *any* protocol that allows a conversation to progress while some participants are offline. The service provider simply needs to "fork" the conversation (into two separate sub-conversations) after a target message and can partition the group into multiple views of the same conversation. We illustrate this with an example. Consider a conversation between Alice, Bob, Charlie, and Dave. The service provider forks the conversation after Alice's second message. The group is partitioned into two views, one where Alice and Bob believe they are the only participants online and the other where Charlie and Dave believe they are the only participants online.

Alice's and Bob's View:

Alice: Lets go to the protest if 3 people want to?

Alice: I want to go.

Bob: I cannot make it.

Charlie's and Dave's View:

Alice: Lets go to the protest if 3 people want to?

Alice: I want to go.

Charlie: I am in.

Dave: Yes, me too.

5.1.2 Multi-providers for conversation integrity

To avoid this conversation integrity attack the system model of Mobile CoWPI consists of a routing/caching service provider with multiple order-enforcing-service (OES)

[1] Informally, that all participants should have the same view of a conversation.

providers. In this model, users register and communicate directly only with the routing service provider. However, when sending the i^{th} message in a conversation, the user uploads it to the routing service provider, who forwards the message to each OES, receiving a confirmation binding the message to index i. Since service providers should only confirm a single message at each index i, if users only accept messages confirmed by all OES providers, the protocol can ensure that messages are handled in an order that preserves the integrity of the conversation if at least one provider is honest. In this "any trust" model we believe a single routing provider and two OES providers are sufficient to provide practical conversation integrity; we discuss some limitations of this model in Section 5.5.

5.1.3 Service Availability

Service availability is not a security goal of Mobile CoWPI. When discussing the protocol we describe multiple service providers. We do not necessarily expect each service to be provided by a single machine, but require each service to be provided by a separate entity. Standard techniques for achieving high availability can be deployed to ensure the service is reliably available.

Denial of Service protection is also a non-goal of Mobile CoWPI. It is trivial for a service provider to deny service to a client by not processing or forwarding messages. It is possible for a malicious provider to behave incorrectly and send malformed or incorrect messages to a client and cause a denial of service. This is equivalent to not sending the messages at all. All messaging applications that rely on a service provider are vulnerable to this type of denial of service. Additionally, if any participants are offline or cannot process a message, all other participants can still progress the conversation. They are not blocked on the offline/denial of serviced participants.

5.1.4 Threat Model

In relation to the threat model described in Section 2.2. The security provided by Mobile CoWPI needs to withstand strong adversaries. We consider an adversary that may compromise multiple service providers and multiple users. The adversary also has full network control and may drop, modify, reorder, and add network traffic. In effect

the adversary can control the routing service provider and all OES providers as well as any number of participants, unless it would trivially allow the adversary to compromise a target security property. Each security property is provided under the strongest adversary that cannot trivially break the property.

5.1.5 Security Properties

Besides the system goals of offline users and message receipts we now quickly restate the security goals of Mobile CoWPI as they relate to the goals introduced in Section 2.3. In Section 5.3 we provide sketches of the security proofs for these properties and provide both the formal definitions of these properties and the full proofs that Mobile CoWPI achieves these properties in Section 5.6.

Message Confidentiality is the property that only conversation participants can read a message.

Forward Secrecy is the property that, after users have ratcheted their key material all messages sent prior to the key ratchet are confidential.

Post-Compromise Secrecy is provided if an adversary is allowed to reveal the long-term and session state of any or all of the users in a conversation, after the key healing, all future message remain confidential.

Message Authentication is the security property that all participants can verify the author of a message and that a message has not been modified in transit.

Participant Authentication is the property that all honest participants can verify all other honest participants are really who they claim to be.

Conversation Integrity is the property that all participants see the same conversation. This includes the order of messages in a conversation and the order of participant changes in a conversation. As we showed earlier, conversation integrity cannot be achieved if the adversary controls all of the routing and OES providers. Thus, the adversary is allowed to control all but one of the OES providers.

Also recall, post-compromise conversation integrity which introduces key healing. A protocol provides post-compromise conversation integrity if after a key healing process, the conversation integrity of future message is not compromised by an adversary that may have revealed the long-term and session state of users or OES providers.

Participant Consistency guarantees all participants of a conversation agree on the set

of all participants in the conversation. In Mobile CoWPI, setup and participant change messages are handled in the same manner as conversation messages. Thus, conversation integrity implies participant consistency.

Deniability is the property that participants may deny taking part in a conversation and authoring any particular message.

5.2 Design

5.2.1 Overview

At a high level Mobile CoWPI is designed as follows. Users register with the routing service provider out-of-band. This registration links a user identity, a long-term public key, and multiple single use pre-keys. When messages are sent as part of a conversation they are uploaded to the routing provider. The routing service then forwards the message to all of the OES providers, each OES returns a confirmation binding the message to its index in the conversation. The routing server then delivers the message and OES confirmations to the clients. The participants do not process a message until it has been received from the routing service provider and has an order confirmation from all of the OES providers. As long as a single OES provider is honest conversation integrity and participant consistency are enforced.

There are 4 types of protocol messages in Mobile CoWPI; *setup*, *conversation*, *participant update*, and *receipt*. Setup messages are used to instantiate a new Mobile CoWPI session and are detailed in Section 5.2.7. Conversation messages contain a message to be displayed to the participants of the session, detailed in Section 5.2.9. Participant Update message allow adding and removing participants from a conversation, detailed in Section 5.2.10. Finally, Receipt messages indicate a participant has received, accepted, and processed all prior messages, detailed in Section 5.2.8.

For a conversation between Alice, Bob, and Charlie. All messages sent by Alice are of the form:

$$Sid, \text{“}TYPE\text{”}, Alice, idx, P, c_{ab}, c_{ac}, auth_{as_1}, \ldots, auth_{as_m}$$

All messages received by Bob from Alice will be of the form:

$$Sid, \text{“}TYPE\text{”}, Alice, idx, P, c_{ab}, auth_{bs_1}, \ldots, auth_{bs_m}$$

Where Sid is the session identifier, idx is the index of the message, c_{a*} is a pairwise ciphertext block between Alice and each participant detailed in Section 5.2.5 and $auth_{as_*}$, $auth_{bs_*}$ are pairwise authentication blocks between Alice or Bob and each OES provider detailed in Section 5.2.6. Sending a message is linear in the number of participants plus the number of OES providers, while receiving a message is constant in the number of participants and linear only in the number of OES providers. This linear size does not limit the scalability of the Mobile CoWPI as Snapchat's end-to-end encrypted snaps are also linear in size and more than a Billion are sent a day [53].

5.2.2 Message Order

To enforce conversation integrity there are seven rules to message ordering.

1. An OES confirmation must be received from every OES provider for a protocol message before processing the message. The protocol messages must also be processed in the order they are received and confirmed.

2. All conversations start with a setup message.

3. When Alice sends a receipt, it must acknowledge all setup, conversation, and participant update messages prior to the receipt that she has not yet acknowledged. Typically they are sent shortly after every message is received.

4. Prior to Alice sending a conversation or participant update message, Alice must have sent a receipt.

5. When Alice sends a receipt, she acknowledges messages with every participant separately. If Bob has just joined the conversation she only acknowledges the messages that she and Bob have in common.

6. When Alice sends a conversation or participant update message, she must acknowledge the most recent prior setup, conversation or participant update message. She must also acknowledge all receipts received after that prior message in order.

7. If Alice receives an invalid protocol message from the routing server she terminates the conversation on her client and does not process any future messages.

Rule (1) implies that even the author of a message must wait until they have received confirmation of the message order from all OES providers before processing it. Otherwise, if two users sent a message at the same time, both users would think their message would come first, causing an order inconsistency.

Rule (6) implies strong ordering of setup, conversation, and participant update messages but not receipts. This was a design choice as requiring receipts to acknowledge receipts would cause significant overhead and excess network traffic when every client sends a receipt at the same time, forcing $n-1$ receipts to be outdated and resent.

Rules (3) and (4) restrict the amount of time a message is vulnerable if the keys used to encrypt it are compromised. We discuss this more as it relates to forward and post-compromise secrecy in Section 5.3.

5.2.3 Primitives

We assume standard cryptographic primitives. Let l be the security level in bits of Mobile CoWPI. All primitives are assumed to provide at least l bits of security. Let G be a group of prime order p generated by g where the decisional Diffie-Hellman assumption is hard.

We assume a hash function and three key derivation functions:

$$
\begin{aligned}
&H : \{0,1\}^l \times Z_p \mapsto Z_p^* \\
&KDF_1 : S \times G \times G \times G \times U \times U \mapsto \{0,1\}^l \\
&KDF_2 : \{0,1\}^l \mapsto \{0,1\}^l \\
&KDF_3 : G \times G \times G \times U \times U \mapsto \{0,1\}^l
\end{aligned}
$$

Where H and KDF_1 are used for two-party NAXOS [40] key agreements, KDF_2 is used to produce a random symmetric key from an input string, and KDF_3 is used for the secure channel between the clients and routing service provider. S is the set of possible session identifiers and U is the set of possible participant identifiers. That is, KDF_1 takes as input a session identifier, three group elements and two user identities, the sender and receiver. These functions are modeled as random oracles. We choose NAXOS as it has the property that to distinguish between a random key and a NAXOS key the distinguisher must know both the long-term and ephemeral secret keys of one

of the participants. KDF_1 is a minor modification of the NAXOS KDF that also includes the session identifier of the current Mobile CoWPI session, where as, KDF_3 is the original NAXOS key agreement.

We assume a symmetric authenticated encryption with associated data (AEAD) scheme. AEAD consists of two functions, $Enc_k(m, d) \mapsto c$, and $Dec_k(c, d) \mapsto m$, or $\perp$ if c and d do not authenticate with key k. The AEAD scheme must provide indistinguishable from random chosen-plaintext-attack security ($IND\$ - CPA$) [54] and integrity of ciphertext security ($INT - CTXT$) [39]. We choose AES-GCM with random IVs for our AEAD scheme.

5.2.4 Registration

To register with the providers Alice generates a long-term public private key pair:

$$lsk_a \leftarrow_R Z_p^*, \qquad lpk_a \leftarrow g^{lsk_a}$$

She also generates a list of ephemeral pre-keys where i is the id of the pre-key:

$$esk_a[i] \leftarrow_R \{0,1\}^l, \qquad epk_a[i] \leftarrow g^{H(esk_a[i], lsk_a)}$$

Alice registers her identity, public long-term key lpk_a and public ephemeral pre-keys epk_a with the providers out-of-band. Alice should generate enough pre-keys to support as many conversations as she expects to start while she is offline. She can always upload new pre-keys in the future. Each pre-key may only be used once. The participants must enforce this rule.

5.2.5 Two Party Ciphertext Blocks

All protocol messages contain pairwise ciphertext blocks c_{ab} where a is the sender and b is the receiving participant. These blocks are used to send additional key information and authenticate the protocol message. They are computed using a simple key ratchet where the initial block uses a pre-key to perform a NAXOS authenticated key agreement and then utilizes AEAD to encrypt and authenticate the message. All subsequent blocks after the initial block use ephemeral keys sent in the previous block to derive a new NAXOS key and then encrypt with AEAD as in the initial block. In this section we describe how to compute these ciphertext blocks in terms of Alice sending to Bob.

Here we describe how Alice computes the initial ciphertext block c_{ab} to send to Bob in session Sid. This ciphertext block encrypts message m and authenticates associated data d. m is only used when sending conversation messages, in which case it is random symmetric key material. When sending setup, receipt, and participant update message m is empty.

First, Alice fetches Bob's long-term public key lpk_b and an ephemeral pre-key epk_b from the routing service provider where id_b is the id of epk_b. Alice generates a new ephemeral key:

$$esk_{ab} \leftarrow \{0,1\}^l, \qquad epk_{ab} \leftarrow g^{H(esk_{ab}, lsk_a)}$$

Then Alice computes a symmetric key:

$$\begin{aligned} ki_1 &\leftarrow epk_b^{lsk_a} \\ ki_2 &\leftarrow lpk_b^{H(esk_{ab}, lsk_a)} \\ ki_3 &\leftarrow epk_b^{H(esk_{ab}, lsk_a)} \\ k &\leftarrow KDF_1(Sid, ki_1, ki_2, ki_3, a, b) \end{aligned}$$

Alice generates her next ephemeral key pair:

$$\begin{aligned} id'_{ab} &\leftarrow 1 \\ esk'_{ab} &\leftarrow_R Z_p^* \\ epk'_{ab} &\leftarrow g^{H(esk'_{ab}, lsk_a)} \end{aligned}$$

She computes the ciphertext block as:

$$c_{ab} \leftarrow epk_{ab}, id_b, Enc_k((m, id'_{ab}, epk'_{ab}), d)$$

When Bob receives $c_{ab} = epk_{ab}, id_b, c$ from the providers he first fetches Alice's long-term public key lpk_a and looks up the ephemeral secret key esk_b associated with id_b and computes the symmetric key as:

$$\begin{aligned} ki_1 &\leftarrow lpk_a^{H(esk_b, lsk_b)} \\ ki_2 &\leftarrow epk_{ab}^{lsk_b} \\ ki_3 &\leftarrow epk_{ab}^{H(esk_b, lsk_b)} \\ k &\leftarrow KDF_1(Sid, ki_1, ki_2, ki_3, a, b) \end{aligned}$$

Then he verifies c and d with k and decrypts:

$$(m, id'_{ab}, epk'_{ab}) \leftarrow Dec_k(c, d)$$

and stores id'_{ab} and epk'_{ab} for latter use. Note that the implicit authentication of NAXOS key exchange authenticates that the message originated from someone with knowledge of Alice's long-term secret key.

All subsequent ciphertext blocks are generated and processed in the same manner as the initial ciphertext block replacing the pre-keys with the ephemeral keys received in the previous block. This key ratcheting provides the self healing necessary for forward and post-compromise secrecy. The users do not send the ephemeral public keys in the clear in subsequent ciphertext blocks. That is the ciphertext block has the form:

$$c_{ab} \leftarrow id'_b, Enc_{k'}((m, id''_{ab}, epk''_{ab}), d)$$

Alice and Bob may try to initialize the two-party key ratchet at the same time. If this happens the providers will enforce an order to the messages and future ciphertext blocks should use the most recently initialized key ratchet.

These ciphertext blocks are what provide message integrity and authentication. This is due to the NAXOS key agreement implicitly authenticating the symmetric keys.

5.2.6 OES Authentication Block

Every protocol message that Alice sends contains an OES authentication block $auth_{aj}$ for every OES provider $j \in S$ where S is the set of OES providers. The authentication blocks are necessary since Alice only uploads the message to the routing service provider. The routing service provider then forwards the message to the OES providers. The authentication blocks allow the OES providers to verify that the message is from Alice and for Alice to verify the index a message she receives.

These OES authentication blocks are generated and handled in the same way as the two-party ciphertext blocks discussed earlier. The key ratcheting provides self healing for post-compromise conversation integrity.

5.2.7 Setup Message

All conversation messages are similar in format. For Alice to setup a conversation she first fetches ephemeral pre-keys for every other particpant and each OES provider. Then she generates a random Sid and computes the setup message:

$$data_0 \leftarrow Sid, Alice, \text{“}SETUP\text{”}, idx, P$$

where idx is the index of the message in the session. For setup messages the index is always 0. Next, Alice computes the two party ciphertext block c_{ai} for every participant $i \in P \setminus \{Alice\}$ as described in Section 5.2.5, where $data_0$ is the associated data to authenticate in those ciphertext blocks. Let $n = |P|$ and :

$$data_1 \leftarrow data_0, c_{a0}, \ldots, c_{an-1}$$

Next, Alice computes the OES authentication block $auth_{aj}$ for every OES provider $j \in S$ as described in Section 5.2.6 where $data_1$ is the associated data to authenticate.

Alice then sends to the routing service provider:

$$data_1, auth_{a0}, \ldots, auth_{as}$$

where $s = |S|$.

The routing provider sends to each OES provider j the message $data_1, auth_{aj}$ along with an ephemeral pre-key for every participant except for Alice. Each OES provider verifies the message $data_1$ is from Alice. Then every provider for every participant $i \in P$ generates an $auth_{ji}$ as described in Section 5.2.6 with $data_0, c_{ai}$ as the associated data. Each OES provider then returns all of the $auth_{j*}$ blocks back to the routing service.

The routing service forwards $data_0, c_{ai}, auth_{*i}$ to every user $i \in P$. Every user verifies the $auth_{*i}$ blocks for every OES provider and that that $data_0, c_{ai}$ is from Alice. The routing service only send $data_0, auth_{*a}$ to Alice as there is not a ciphertext block for herself.

Once a participant has received the setup message along with an OES authentication block from the routing service provider and verified the message, they setup a new Mobile CoWPI session with session identifier Sid. All providers must verify Sid is not used for any existing session before processing the message.

5.2.8 Receipt Message

Participants send receipts after they have accepted any setup, conversation, or participant update message. If multiple messages are sent while Alice is offline she sends a single receipt that acknowledges all messages m_i with participant $i \in P \setminus \{Alice\}$. The messages to acknowledge depend on the participant they are being acknowledged to. m_i is composed of all protocol message, excluding receipts more recent than the last setup, conversation, or participant update message, that have not been acknowledge previously and have been sent after participant i has been added to the conversation. This is because i cannot acknowledge messages they have not seen. m_i should be a list of all $data_0$ blocks from the messages to acknowledge in order.

A receipt is similar to a setup message. When Alice generates a receipt for messages she computes:

$$data_0 \leftarrow Sid, Alice, \text{“}RCPT\text{”}, pidx$$

where $pidx$ is the index of the previous setup, conversation, or participant update message. Then she computes the two party ciphertext block c_{ai} for every participant $i \in P \ \{Alice\}$ as detailed in Section 5.2.5 with the associated data to authenticated as $data_0, m_i$. Let

$$data_1 \leftarrow data_0, c_{a0}, \ldots, c_{an-1}$$

She then computes the OES authentication blocks as detailed in Section 5.2.6 with the associated data as $data_1$. Finally, she sends $data_1$ and the authentication blocks to the routing provider.

The routing service provender and OES providers handle the message in the same way as a setup message detailed earlier. Except this the routing server sends the index of the receipt (idx) to the OES providers. They verify that $pidx$ and idx are correct and generate the OES block for user i with $data_0, c_{ai}, idx$ as the associated data.

When a participant receives a receipt they first verify the OES authentication blocks then verify that the receipt authenticates the correct messages. If anything does not verify, they terminate the session.

5.2.9 Conversation Message

Conversation messages are similar to receipts except they contain a ciphertext. Let idx be the index of the next message in the session Sid. When Alice wants to send the conversation message m. She first generates a random symmetric key input $k_a \leftarrow_R \{0,1\}^l$ then computes the symmetric key $k \leftarrow KDF_2(k_a)$. Let

$$data_0 \leftarrow Sid, Alice, \text{“}MSG\text{”}, idx, Enc_k(m)$$

She then generates the ciphertext block c_{ai} for every $i \in P \setminus \{Alice\}$ as detailed in Section 5.2.5 with k_a as the data to encrypt in the ciphertext block and $data_0$ as the associated data. Let

$$data_1 \leftarrow data_0, c_{a0}, \ldots, c_{an-1}$$

She then computes the OES authentication blocks as detailed in Section 5.2.6 with the associated data as $data_1$. Finally, she sends $data_1$ and the authentication blocks to the routing provider.

The routing service and OES providers handle the message in the same manner as receipt messages, verifying the OES authentication blocks and index. After receiving the message, each participant verifies the OES authentication blocks and index and displays the message. If the message does not verify the session is terminated.

5.2.10 Participant Update Message

To change the set of participants in a conversation a member of the conversation can send a participant update message. Who is allowed to send the messages as well as what modifications they are allowed to make are out-of-scope of this work. However, participant modifications must be enforceable by the providers since they need to forward and authenticate messages.

Participant update messages are similar to conversation messages except that the conversation message ciphertext is replaced with a list of participants. Again let idx be index of the next message in session Sid. When Alice wishes to change the participants of a conversation to P' she creates a message:

$$data_0 \leftarrow Sid, Alice, \text{“}UPDT\text{”}, idx, P'$$

She then creates the ciphertext block c_{ai} for participant $i \in (P \cup P') \setminus \{Alice\}$ as described in Section 5.2.5 where $data_0$ is the associated data for the ciphertext blocks. Let

$$data_1 \leftarrow data_0, c_{a0}, \ldots, c_{an-1}$$

Alice then creates the provider authentication block $auth_{aj}$ for provider $j \in S$ as detailed in Section 5.2.6 with $data_1$ as the associated data. She uploads $data_1$ along with the provider authentication blocks to the routing provider.

The routing services and OES providers handle the update message in the same way as a setup message. Each provider checks that Alice is allowed to make the desired group modification and verifies the index is correct. Each participant verifies the messages is authentic from Alice and updates their participant list after they have received the message from every provider. If the message does not verify the session is terminated.

This message authenticates the group change to all old and new participants which leaks any new participants to participants that have been removed. To avoid this leakage it is up to the implementation to send a separate group update message removing users before sending a message adding the new users.

5.2.11 Two Party Channels

All communication between the clients, OES providers, and the routing service is performed over a two-party channel that supplies all of the security properties discussed in Section 5.1. The OES providers act as clients when communicating with the routing provider. This is a synchronous channel that is setup by performing a NAXOS key agreement to provide authentication to the channel. Then all messages are secured by using a NAXOS key agreement with keys being ratcheted on every message to provide forward and post-compromise secrecy.

Algorithm 9 details the algorithm for setting up the channel from the initiator. Line 2 generates the clients first ephemeral NAXOS keys. Lines 3 sends the clients identity and NAXOS ephemeral public key to the provider. Line 4 receives the provider's response and line 5-9 compute the shared NAXOS key and decrypt the provider's next ephemeral public key and a challenge. Line 10 generates the clients next ephemeral keys. Finally, Lines 11-16 ratchets the channel keys and sends the challenge back encrypted.

Algorithm 9 Client To Provider Channel Setup

1: **function** C2SChannelSetup(C, S, lpk_s, lsk_c)
2: $esk_c[0] \leftarrow_R Z_p^*$, $epk_c[0] \leftarrow g^{H(esk_c[0], lsk_c)}$
3: Send($S, C, epk_c[0]$)
4: $epk_s[0], c_1 \leftarrow$ Recv(S)
5: $km_1 \leftarrow lpk_s^{H(esk_c[0], lsk_c)}$
6: $km_2 \leftarrow epk_s[0]^{lsk_c}$
7: $km_3 \leftarrow epk_s[0]^{H(esk_c[0], lsk_c)}$
8: $k_1 \leftarrow KDF_3(km_1, km_2, km_3, S, C)$
9: $t, epk_s[1] \leftarrow Dec_{k_1}(c_1)$
10: $esk_c[1] \leftarrow_R Z_p^*$, $epk_c[1] \leftarrow g^{H(esk_c[1], lsk_c)}$
11: $km_4 \leftarrow epk_s[1]^{lsk_c}$
12: $km_5 \leftarrow lpk_s^{H(esk_c[0], lsk_c)}$
13: $km_6 \leftarrow epk_s[1]^{H(esk_c[0], lsk_c)}$
14: $k_2 \leftarrow KDF_3(km_4, km_5, km_6, C, S)$
15: $c_0 = Enc_{k_2}(t, epk_c[1])$
16: Send(S, c_0)
17: **return** esk_c, epk_s

Algorithm 10 details setting up the channel from the provider. Lines 2 receives the clients identity and NAXOS ephemeral public key. Line 3 looks up the long-term public key of the client. Lines 4-5 compute the next two ephemeral NAXOS keys of the provider. Line 6-8 compute the NAXOS shared key. Lines 9-12 encrypt the challenge and the providers next ephemeral DH key and send it to the client. Lines 13-22 decrypt the clients response and check that the client's response matches the challenge, storing the clients next ephemeral key.

Algorithm 11 details how a message is sent using the two-party channel. Lines 2-3 find the id of the senders last sent ephemeral DH key and the receivers last seen ephemeral public key. Line 4 computes the shared secret from the two keys and line 5 generates the senders next ephemeral DH keys. Line 6 encrypts the message and the next ephemeral public key. Finally, line 7 sends the encrypted message along with the id of the receivers public key used to encrypt it.

Algorithm 12 details receiving a message from the channel. Line 2 finds the id of the sender's last ephemeral public key. Line 3 reads the id of the receiver's ephemeral key used to encrypt the message and the ciphertext. Line 4 computes the shared key

Algorithm 10 Provider To Client Channel Setup

1: **function** S2CChannelSetup(S, C, lsk_s)
2: $C, epk_c[0] \leftarrow$ Recv(C)
3: $lpk_c \leftarrow$LookupUser(C)
4: $esk_s[0] \leftarrow 0, 1^l, epk_s[0] \leftarrow g^{H(esk_s[0], lsk_s)}$
5: $esk_s[1] \leftarrow Z_p^*, epk_s[1] \leftarrow g^{esk_s[1]}$
6: $km_1 \leftarrow epk_c[0]^{lsk_s}$
7: $km_2 \leftarrow lpk_c^{H(esk_s[0], lsk_s)}$
8: $km_3 \leftarrow epk_c[0]^{H(esk_s[0], lsk_s)}$
9: $k_1 \leftarrow KDF_3(km_1, km_2, km_3, S, C)$
10: $t_s \leftarrow_R \{0,1\}^l$
11: $c_1 \leftarrow Enc_{k_1}(t_s, epk_{s[1]})$
12: Send($C, epk_s[0], c_1$)
13: $c_2 \leftarrow$ Recv(C)
14: $km_4 \leftarrow lpk_c^{H(esk_s[1], lsk_s)}$
15: $km_5 \leftarrow epk_c[0]^{lsk_s}$
16: $km_6 \leftarrow epk_c[0]^{H(esk_s[1], lsk_s)}$
17: $k_3 \leftarrow KDF_3(km_4, km_5, km_6, C, S)$
18: $t_c, epk_c[1] \leftarrow Dec_{k_2}(c_1)$
19: **if** $t_s = t_c$ **then**
20: **return** (C, esk_s, epk_c)
21: **else**
22: **return** $\perp$

and line 5 decrypts the message and the senders next ephemeral public key.

5.2.12 Long-term Key Verification

The ability for Alice to verify that Bob is actually Bob is a challenging problem in messaging systems. This is enforced in Mobile CoWPI by verifying the real Bob knows the private key associated with the long-term public key Alice retrieves from the providers. Mobile CoWPI does not necessitate a specific mechanism for verifying these keys and identities but some such mechanism is required to provide participant authentication. In practice key fingerprints can be compared in person or with an interactive scheme such as the Socialist Millionaire Protocol (SMP) as applied by Alexander and Goldberg [41].

Algorithm 11 Channel Send

function SECURESEND($S, R, m, lsk_s, esk_s, lpk_r, epk_r$)
  $n_s \leftarrow |esk_r|$
  $n_r \leftarrow |epk_r|$
  $km_1 \leftarrow epk_r[n_r - 1]^{lsk_s}$
  $km_2 \leftarrow lpk_r[n_r - 1]^{H(esk_s[n_s-1], lsk_s)}$
  $km_3 \leftarrow epk_r[n_r - 1]^{H(esk_s[n_s-1], lsk_s)}$
  $k \leftarrow KDF_3(km_1, km_2, km_3, S, R)$
  $esk_s[n_s] \leftarrow Z_p^*, epk_s[n_s] \leftarrow g^{esk_s[n_s]}$
  $c \leftarrow Enc_k(m, epk_s[n_s])$
  SEND($R, n_r - 1, c$)
  return (esk_s, epk_r)

Algorithm 12 Channel Receive

function SECURERECV(R, S, esk_r, epk_s)
  $n_s \leftarrow |epk_s|$
  $n_r, c \leftarrow$RECV(C)
  $km_1 \leftarrow lpk_s^{H(esk_r[n_r-1], lsk_r)}$
  $km_2 \leftarrow epk_s[n_s - 1]^{lsk_r}$
  $km_3 \leftarrow epk_s[n_s - 1]^{H(esk_r[n_r-1], lsk_r)}$
  $k \leftarrow KDF_3(km_1, km_2, km_3, S, R)$
  $m, epk_s[n_s] \leftarrow Dec_k(c)$
  return (esk_r, epk_s)

5.3 Security

In this section we discuss the security provided by Mobile CoWPI. We argue that it provides all of the desired security properties discussed in Section 5.1. We provide full proofs in Appendix 5.6. We model our hash function (H) and key derivation functions (KDF_1, KDF_2) as random oracles. We also assume the decisional Diffie-Hellman problem is hard. We utilize the fact distinguishing between a random key and a key generated with the NAXOS key agreement is hard if the adversary does not know the long-term and ephemeral secret keys of one of the parties in the key agreement as shown by the NAXOS authors. We assume our AEAD scheme provides $IND\$ - CPA$ and $INT - CTXT$ security. Finally, we assume all participants in a conversation have verified their long-term keys either manually or with SMP.

5.3.1 Message Confidentiality

Message confidentiality is the property that only participants of a conversation can read a message. We provide message confidentiality against a powerful adversary that may corrupt any or all of the providers, may control any user that is not a participant in the target conversation, and may reveal the long-term and ephemeral keys of any participant on any non-target message.

To compromise the confidentiality of a message:

$$Sid, \text{“}MSG\text{”}, A, idx, Enc_{KDF_2(k_a)}(m), c_{a1}, \ldots, auth_{a1}, \ldots$$

The adversary must be able to distinguish between $Enc_{KDF_2(k_a)}(m)$ and a random string. If an adversary can make this distinction they must be able to do one of the following:

1. Compute a two-party NAXOS key without being one of the parties allowing them to decrypt one of the ciphertext blocks c_* and retrieve the key input k_a, thus decrypting the m.

2. Decrypt one of the c_* ciphertext blocks without knowing the symmetric key and learn k_a, thus breaking the $IND\$ - CPA$ security of the AEAD scheme.

3. Distinguish the ciphertext $ENC_{KDF_2(k_a)}(m)$ from random without knowing k_a, thus breaking the $IND\$ - CPA$ security of the AEAD scheme.

5.3.2 Message Authentication and Integrity

Message authentication provides the property that when Bob receives a message from Alice in session Sid, Alice must have sent that message. Mobile CoWPI provides message authentication against a strong adversary that may control any or all of the providers and any users in any session. As long as Alice and Bob have not had their long-term keys and ephemeral keys of session Sid compromised, all messages received by Bob from Alice are authentic.

For an adversary to forge a message from Alice to Bob the adversary must create a message:

$$Sid, \text{“}MSG\text{”}, A, idx, Enc_{KDF_2(k_a)}(m), c_{ab}, \ldots, auth_{a1}, \ldots$$

If the adversary can forge the message they must be able to do one of the following:

1. Compute a two party NAXOS key without knowing Alice's or Bob's long-term and ephemeral keys, allowing the adversary to create the ciphertext block c_{ab}.

2. Forge a valid ciphertext block c_{ab} from Alice to Bob without knowing the symmetric key, thus breaking the $INT - CTXT$ security of the AEAD scheme.

5.3.3 Forward Secrecy

Forward secrecy is the property that past messages are confidential even if future key material is revealed. Mobile CoWPI provides forward secrecy of a message m after every user $i \in P$ has processed the receipt of every user $j \in P$ acknowledging m. Forward secrecy assumes the same adversary as message confidentiality.

Let P be the set of participants in session Sid and let m_a be the message:

$$Sid, \text{“}MSG\text{”}, A, idx, Enc_{KDF_2(k_a)}(m), c_{a1}, \ldots, auth_{a1}, \ldots$$

be a message sent from user $A \in P$. The adversary cannot distinguish $Enc_{KDF_2(k_a)}(m)$ from random after every participant $i \in P$ has processed a receipt from A, acknowledging m_a, and A has processed a receipt form i acknowledging m_a. First we show that every ephemeral private key esk_{ia} used to compute ciphertext block c_{ai} will never be used again and thus can be deleted. Then we show that without esk_{ia} the adversary cannot distinguish $Enc_{KDF_2(k_a)}(m)$ from random similar to message confidentiality.

The ciphertext block c_{ai} is computed using a's ephemeral private key esk_{ai} and i's ephemeral public key epk_{ia}. In c_{ai}, a distributes a new ephemeral public key epk'_{ai} and can safely delete esk_{ai}, so all esk_{ai} have been deleted after sending m_a.

Now we show esk_{ia} can be deleted after i has sent a receipt that acknowledges m_a and processed a receipt from a acknowledging m_a. Let the receipt from i be:

$$r_i \leftarrow Sid, \text{“}RCPT\text{”}, I, pidx, c_{i1}, \ldots, auth_{i1}, \ldots$$

Ciphertext block c_{ia} is generated using ephemeral private key esk_{ia} and ephemeral public key epk'_{ai}. In c_{ia}, i distributes a new ephemeral public key epk'_{ia}. Let r_a be the receipt from a acknowledging m_a. The ephemeral private key esk_{ia} can be deleted after i processes both r_i and r_a. Since receipts do not enforce an order, a may use esk_{ia} when

sending r_a. After a sends r_a she may only send a conversation message or group update message, which acknowledges r_i and thus uses epk'_{ia}. This shows that esk_{ai} and esk_{ia} can be deleted after a and i process the receipts r_a and r_i.

After keys have been ratcheted Mobile CoWPI provides the same message confidentially property as discussed previously.

5.3.4 Post-Compromise Secrecy

Post-Compromise secrecy is the property that compromising prior long-term and ephemeral key material does not break the confidentiality of future messages. If Alice's long-term or ephemeral state are revealed, all conversation messages following Alice's next receipt provide post-compromise secrecy. Similar to forward secrecy we need to show that Alice's compromised ephemeral private keys are not used in the next conversation message.

Let esk_{ai} be Alice's compromised ephemeral key used for the two-party ciphertext block with user i. We show that after Alice's next receipt, the following conversation message does not use esk_{ai}. Let Alice's receipt be:

$$r_a \leftarrow Sid, \text{“}RCPT\text{”}, Alice, pidx, c_{a1}, \ldots, auth_{a1}, \ldots$$

Recall that the ciphertext block c_{ai} is encrypted with a key generated from esk_{ai} and contains a new ephemeral key esk'_{ai}. Let c'_{ai} be the ciphertext block of the conversation message. Since all messages must acknowledge all prior receipts, c'_{ia} must use Alice's ephemeral key epk'_{ai} from her receipt.

Similar to forward secrecy, after keys have been ratcheted Mobile CoWPI provides message confidentiality as discussed previously.

5.3.5 Conversation Integrity

Conversation integrity is the property that all honest participant in a conversation see the same conversation. That is all honest participants agree on the order of all setup, conversation, and participant update messages. Conversation integrity considers an adversary that controls the network, can compromise all but one OES provider, and can compromise participants in the conversation. The adversary is not allowed to

compromise all the OES providers, otherwise breaking conversation integrity is trivial, regardless of the protocol. If all of the providers are compromised the adversary can simply partition the group.

Consider a conversation between Alice, Bob, and Charlie. After Alice sets up the conversation the adversary can partition the conversation by never forwarding messages from Charlie to Alice or Bob, and similarly never forwarding any messages, after the setup message, from Alice or Bob to Charlie. Alice and Bob will believe Charlie has never come online and continue the conversation, while Charlie will believe Alice and Bob are always offline and continue the conversation alone. Thus, at least one provider must be honest.

If at least one provider is honest, to break conversation integrity the adversary must send a message:

$$Sid, \text{“}MSG\text{”}, A, idx, Enc_{KDF_2(k_a)}(m), c_*, \ldots, auth_*, \ldots$$

where two honest users (Alice and Bob), decrypt different key inputs values from their respective ciphertext that both decrypt $Enc_{KDF_2 k^*}(m)$ to different valid plaintext. Let c, d be arbitrary strings; then the probability ϵ_{int} that $Dec_k(c, d) \neq \perp$ for a random key k must be negligible, since an adversary can win the $INT - CTXT$ game by simply submitting c, d as a ciphertext query. This holds even when $c = Enc_{k'}(m, d)$ for some fixed k'. Thus if the adversary makes at most q queries to KDF_2, the probability of finding a $k' = KDF_2(k)$ breaking conversation integrity in this way is at most $q\epsilon_{int}$.

If the adversary cannot find a valid ciphertext under two random keys, to break conversation integrity the adversary must convince two participants to accept different messages as the i^{th} message of conversation Sid. The honest participants only accept a message after verifying all the OES authentication blocks bind the message to the specific index. An honest OES provider will authenticate all messages in a consistent order to all participants. The adversary must be able to forge an OES authentication block for a message to an honest participant A as if it came from honest OES provider S. If the adversary can forge such a message, it must be able to do one of the following:

1. Compute a two party NAXOS key without knowing A's or S's long-term and ephemeral keys, allowing the adversary to create the authentication block $auth_{sa}$.

2. Forge a valid authentication block $auth_{sa}$ from S to A without knowing the symmetric key, thus breaking the $INT - CTXT$ security of the AEAD scheme.

5.3.6 Participant Consistency

Participant consistency is the property that all users agree on the set of participants in a conversation. We provide participant consistency under a strong adversarial model. The adversary controls the network and may compromise all but one OES provider and any participants. The adversary wins if she can cause two honest users to have different sets of users for session Sid after processing a setup or participant update message and not terminating. Since setup and participant update messages in Mobile CoWPI are part of the protocol transcript and Mobile CoWPI provides conversation integrity, Mobile CoWPI also provides participant consistency.

5.3.7 Deniability

Recall deniability as discussed in Section 5.1. Deniability is provided if a single user can run a simulator and produce a simulated transcript that is indistinguishable from a transcript of a real protocol execution. The simulator must only take as input information that is known to a single user. That is, only a single users view of the conversation, which is simply a sequence of two-party messages. The distinguisher is given all of the long-term secret information and any secret state information of the simulating user. This requires the simulator to also output any state information of the user.

We now detail the simulator. Let Alice be the party running the simulator. She acts as all parties in the conversation and behaves as normal expect when performing NAXOS key agreements. The NAXOS key agreements are the only part of the Mobile CoWPI protocol that Alice cannot perform honestly as she does not have the secret key material of all participants. Their are two cases of the NAXOS key agreement she needs to simulate:

1. When she is a participant of the NAXOS key agreement.

2. When she is not a participant of the NAXOS key agreement.

In the first case let Bob be the other participant. Alice may have a valid ephemeral public key of the other participant if she is sending the SETUP message. Otherwise she

generates an ephemeral key epk_b for the other participant as a random group element. She then computes the NAXOS key as she normally would.

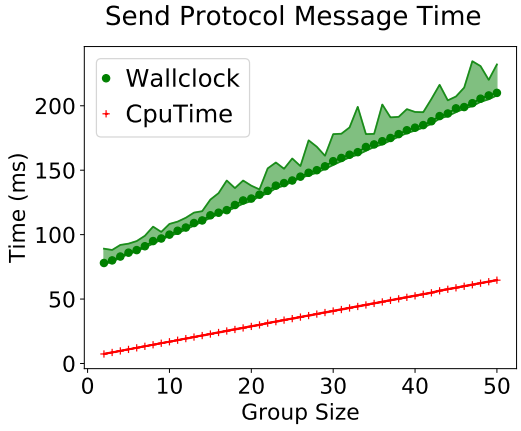
If she has a valid ephemeral key for Bob the NAXOS key agreement is a real key agreement. If she generates a random key from Bob the distinguisher must distinguish between the random key and a real NAXOS ephemeral key $epk_b \leftarrow g^{H(\{0,1\}^l, lsk_b)}$. Since H is modeled as a random oracle the distinguisher can only win if it queries the random oracle on all 2^l possible ephemeral secret keys with Bob's long-term secret key. Thus the adversary cannot tell epk_b apart from a random group element with less than 2^l oracle queries.

In the second case let Bob and Charlie be the two participants. Alice will have a valid ephemeral public key for one of them if they are sending a SETUP message. As before, Alice will generate any ephemeral keys she does not have as random group elements and then generates the NAXOS key as a random symmetric key; the distinguisher cannot tell if the randomly generated ephemeral keys are real with less than 2^l oracle queries. Since the distinguisher does not know the ephemeral secret key of either party it cannot distinguish between a random key and a real NAXOS key.

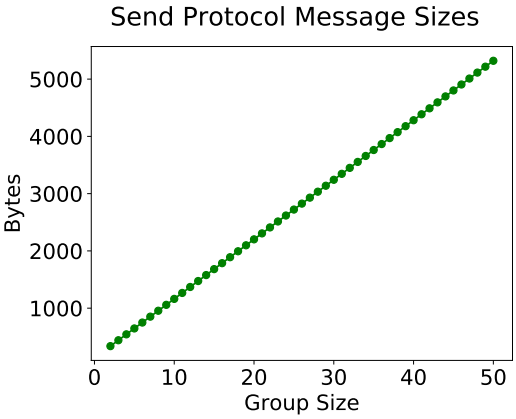
Using these NAXOS simulators, Alice can simulate all parties of a Mobile CoWPI protocol session and produce a simulated transcript that is indistinguishable from a real transcript. Thus, Mobile CoWPI provides message and participant deniability.

Message Unlinkability

Message unlinkability is the property that proving authorship of any one message does not prove authorship of any additional message. This property has not been formally defined previously. It was first discussed in relation to mpOTR [16], as mpOTR is considered not to provide message unlinkability. This is due to mpOTR using the same ephemeral signing key to sign every message. Thus, the distinguisher having knowledge of the ephemeral verification key can verify every message sent by a user. Since Mobile CoWPI does not use signatures and all authentication material is only used for a single message Mobile CoWPI provides message unlinkability. In Appendix 5.6, we prove a stronger version of message unlinkability that provides the distinguisher with a protocol message from a real transcript but can still not distinguish the full transcript from a simulated transcript.



(a) The wallclock (25^{th}, 50^{th}, and 90^{th} percentile) and CPU time to send a protocol message.



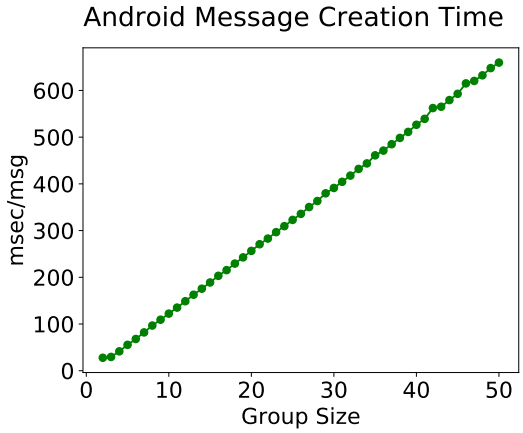
(b) The message size in bytes to send a message.

5.4 Evaluation

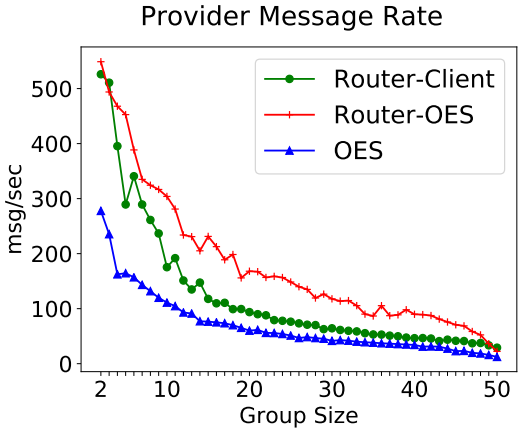
We implemented Mobile CoWPI as a Java server and client library. Since all protocol messages can be processed without interaction between clients the overhead of Mobile CoWPI is low. To measure the run time overhead we deployed Mobile CoWPI in an inexpensive deployment with a routing service and one OES on an AWS [10] free tier t2.micro EC2 instance in Ohio and a second OES hosted on a $5/month Linode [11] virtual private server located in New Jersey. Since Mobile CoWPI uses an any trust model two OES providers is sufficient. We ran all of the client measurements from a personal desktop machine over a home internet connection. The client machine contains an AMD FX 8300 CPU. The network round-trip-time between the client and the router is $\approx 30ms$ and between the router and Linode OES is $\approx 20ms$. The network round trip of a message is from client to routing server to OES to routing server then back to the client. This introduces an $\approx 50ms$ latency for messaging in our measurements.

We ran the measurements with 2 to 50 participants in a conversation and sent 100 messages for each size of conversation. These measurements show Mobile CoWPI is practical for real world deployments.

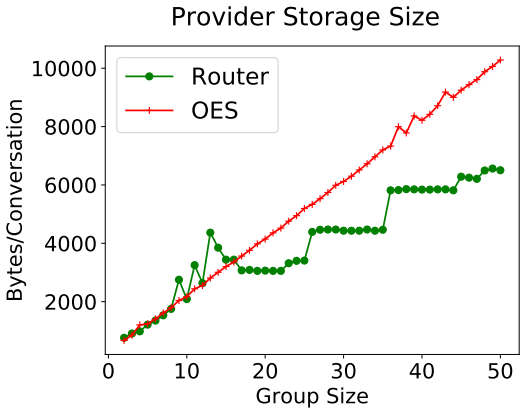
Figure 5.1a shows the time in milliseconds that it takes for a user to send a protocol message and receive a protocol message. This represents the time it takes to display the message. Figure 5.1b shows the outgoing message size in bytes when sending a message.



(a) Time to build a message on an Android device.



(b) Rate at which the Router can process a message from a client and OES and rate at which an OES can process a message.



(c) Provider storage cost per conversation.

All outgoing messages are $O(n+s)$ in size where n is the number of participants and s is the number of OES providers. This is due to the authentication being pairwise with all receivers. We discuss why this overhead is necessary in Section 5.5. Pairwise ciphertext blocks does allow for very little overhead to receive a message. Conversation and receipt messages are $O(s)$ while setup and participant update message must be $O(n+s)$ in size to distribute the list of participants. The overhead of Mobile CoWPI for incoming messages is less than 300 bytes.

5.4.1 Scalability

We evaluated the scalability of our deployment by measuring the message throughput and storage costs of our AWS t2.micro routing server provider and OES provider along with the performance of our Java implementation on a Motorola G3 [55] Android [52] phone.

Figure 5.2a shows the time in milliseconds to create a protocol message on the Android device. Figure 5.2b shows the maximum throughput of the router for processing messages from a client and an OES for each group size. It also shows the maximum throughput of an OES provider. Figure 5.2c shows the storage space required per conversation of each size for both the router and OES. Our implementation uses a PostgreSQL [56] database which causes the steps seen in Figure 5.2c.

Mobile CoWPI can easily scale horizontally by sharding across multiple servers by the conversation SID. For example, for a router service provider to support processing 1 Billion messages a day for groups of size 5 and (10). One t2.micro can support $\approx$ 289(165) messages per second, it would require 40(66) instances. The t2.micro instances are priced at \$0.0116/ per hour under burstable workloads and are charged an addition \$0.05 per hour under sustained high workloads. Thus the cost would range from \$11.14(\$18.38) to \$59.14(\$97.58) per day.

5.5 Discussion

In this section we detail the limitations of Mobile CoWPI along with restrictions enforced by the system model.

5.5.1 Limitations of Group Key Agreements

The MLS draft protocol scales to larger group sizes than Mobile CoWPI can support by using a tree-based Group Key Agreement (GKA) scheme. GKAs provide a mechanism for a group of users to compute a symmetric encryption key, which can then be used to encrypt a message and authenticate that the message originated from a member of the group. However, such a scheme by itself does not provide a mechanism to verify that a message came from a specific member of the group.

To provide message sender authentication, another authentication mechanism must be introduced and it must be deniable. Recently MLS has proposed to use a signature scheme to provide sender authentication with a tree-based GKA scheme. However, by its nature a signature on a message is unforgeable and thus not deniable. Another possible approach would be to use a multi-designated verifier signature scheme [57], which provides *source hiding* signatures that can be validated by anyone and can be generated either by the author or by the full group of receivers. This would provide a weaker form of deniability than Mobile CoWPI provides, since simulation requires the cooperation of all users.

The key challenge is to provide message authenticators that can be simulated by any subset of the group, but cannot be forged by any subset of the group. Mobile CoWPI achieves this by using deniable pairwise ciphertext blocks to authenticate every message. While this limits the size of groups that can be supported, in practice this has not been problematic for existing end-to-end encryption schemes; for example, both Signal and Snapchat's end-to-end encryption [53] are linear in the number of receivers and have been deployed to support billions of messages per day.

5.5.2 Multiple Providers

Requiring multiple providers for conversation integrity adds difficulty to deploying Mobile CoWPI. However, if this requirement cannot be met the conversation integrity property could be modified to include a time aspect. Most users are expected to only be offline for short periods of time, for example less than one week. It is also the case that after Alice receives a receipt from Bob, she can be confident that Bob's transcript provides conversation integrity with her transcript. Thus, if every user sends a receipt after every message, we can add a time constraint to the conversational integrity property and warn users after a time limit (e.g. one week) of not having seen a receipt from every other participant. We chose to require multiple providers for Mobile CoWPI as it provides much stronger conversation integrity for every message.

5.5.3 Denial of Service

Mobile CoWPI does not protect against denial of service attacks from compromised servers: a server can simply not forward conversation messages to a participant. Since the participant must receive the message from every server, the participant will simply keep waiting and not make progress. A potential solution to this problem would be to have multiple servers perform a byzantine agreement on the messages of a conversation and then participants could process a message after receiving it from a majority of servers. This changes the trust model from a single honest server to a majority of honest servers and it is not straight forward how this modification would affect the deniability properties of the conversation.

Mobile CoWPI also does not offer denial of service protection against a compromised participant. A compromised participant can send an invalid ciphertext block c_* to a victim. The victim will terminate the session and all non-victims will not know of the attack. The implementation should warn the user of the attack allowing them to notify the other participants out-of-band. It may be possible to mitigate this issue by modifying the ciphertext blocks to provide zero knowledge proofs of correctness that the servers can verify. However, we do not know of an efficient mechanism that would allow for this and also preserve message deniability and unlinkability.

These denial of service limitations are not unique to Mobile CoWPI. All existing protocols in the literature and in wide deployment are also vulnerable to denial of service by both the server and individual participants.

5.6 Formal definitions and proofs

We define all of our security conditions in terms of a game in which a challenger runs a procedure INITIALIZE to set up an initial state, before running an adversary that may access several oracles that can access and modify the game state; the game concludes when the adversary calls the FINALIZE oracle, which determines if the adversary has won the game. For each game, the complete experiment is defined by the initialization procedure, the set of oracles defined for the game, and the finalization function. For all experiments that involve running Mobile CoWPI, we assume that each client c maintains a list M_c of the tuples of the form (sid, s, i, pm) indicating that c *accepted* $pm \neq \perp$ as

Figure 5.3: IND$-CPA Game

function INITIALIZE(l)
  $k \leftarrow^R \{0,1\}^l$
  $b \leftarrow^R \{0,1\}$
function TEST($m, data$)
  $c_0 \leftarrow Enc_k(m, data)$
  $c_1 \leftarrow_R \{0,1\}^{|c_0|}$
  return c_b
function FINALIZE(d)
  return $(d = b)$

the i-th protocol message in session sid, with sender s.

5.6.1 Security Assumptions

We assume our symmetric AEAD scheme ciphertexts are indistinguishable from random bit strings (IND$-CPA) as defined by the game in Figure 5.3 and provides integrity of ciphertexts as defined in Figure 5.4. The advantage of an adversary M winning each of the games is defined as $Adv^{IND-CPA}(M) = Pr[M \text{ wins}] - \frac{1}{2}$, $Adv^{INT-CTXT}(M) = Pr[M\text{wins}]$ respectively.

We assume the NAXOS protocol is a secure authenticated key agreement protocol. Figure 5.5 describes the game used by the original authors [40], modified to include an additional bit string ($\{0,1\}^l$) into the EPHEMERAL KEY REVEAL and TEST queries that is included in the input of KDF_2 of NAXOS. This modification is to allow the Mobile CoWPI session id Sid to be incorporated into the KDF and does not affect the security of NAXOS. The NAXOS session id is

$$sid = (role, ID, ID^*, comm_1, \ldots, comm_n)$$

where ID is the identify of the executing party and ID^* is the identities of the other party, $role \in \{I, R\}$ is the role of initiator or responder, and $comm_i$ is the i^{th} communication sent by the parities. This preserves the session matching of NAXOS.

An adversary wins if it queries TEST on a clean session and guess the correctly in FINALIZE. Let sid be the NAXOS session between parties A and B. Let sid^* be the

Figure 5.4: INT-CTXT Game

function INITIALIZE(l)
  $k \leftarrow^R \{0,1\}^l$
  $S \leftarrow \{\}$
function ENC(m, d)
  $c \leftarrow Enc_k(m, d)$
  $S \leftarrow S \cup \{c\}$
  return c
function VF(c)
  $m \leftarrow Dec_k(c, d)$
  if $m \neq \perp$ and $c \notin S$ **then**
    $win \leftarrow true$
  return $(m \neq \perp)$
function FINALIZE(d)
  return win

matching session of sid executed by B, sid^* may not exist. A session is *not* clean if any of the following hold:

- A or B is an adversary-controlled party
- REVEAL is queried on sid or sid^*
- sid^* exists and both the long-term and ephemeral key of A or B are revealed
- sid^* does not exist and the long-term key of B was reveled or both the long-term and ephemeral key of A was revealed

An adversary M's advantage at winning the NAXOS game is defined as $Adv^{NAXOS}(M) = Pr[M \text{ wins}] - \frac{1}{2}$.

5.6.2 Message Confidentiality

Message Confidentiality is the property that only conversation participants can read a message. The adversary we consider controls the network and is allowed to register malicious users and reveal the long-term keys and ephemeral keys of users. When discussing message confidentiality we consider the confidentiality of individual (target)

Figure 5.5: NAXOS Game

function INITIALIZE(U)
Initialize PKI for all users in U.

function SEND($A, B, comm$)
Send $comm$ to A on behalf of B
This query allows A to start a NAXOS AKE with B.
return A's communication to B

function LONG-TERM KEY REVEAL(A)
return Long-term private key of A

function EPHEMERAL KEY REVEAL(sid)
return Returns the ephemeral private key of a possibly incomplete session sid.

function REVEAL(sid, Sid)
return Session key of completed NAXOS session sid with Mobile CoWPI session id Sid

function TEST(sid, Sid)
$b \leftarrow^R \{0,1\}$
if $b = 0$ **then**
$C \leftarrow$REVEAL(sid, Sid)
else
$C \leftarrow^R \{0,1\}^l$
return C

function FINALIZE(d)
return $(d = b)$

Figure 5.6: Message Confidentiality Game G_0

function INITIALIZE(U)
$b \leftarrow_R \{0,1\}$
Initialize PKI for all users in and servers U.

function SEND(R, S, m)
Send m to R from S where R and S may be participants or servers.
return Network output of R after processing m

function SETUPGROUP(Sid, P, U)
Setup session Sid, as participant P for users U.
return Network output of P

function SENDGROUPMESSAGE(Sid, P, m)
Send message m from P to group Sid.
return Network output of P.

function UPDATEPARTICPANTS(Sid, P, U)
Send participant update message as P for participants U in session Sid.
return Network output of P.

function REVEALEPHEMERALKEYS(Sid, A, B)
return The ephemeral secret keys of A that A uses for communication with B in session Sid. A or B may be users or servers. If A or B is a server, Sid is ignored.

function REVEALLONGTERMKEYS(T)
return The Long-term keys of T where T may be a server or participant.

function TEST(Sid, P, m)
if $b = 0$ **then**
P sends protocol broadcast message of m in session Sid
else
P send a random bit string in Sid
return P's network traffic to send the message

function FINALIZE(d)
return $(d = b)$

messages in a session. The adversary is only limited to avoid trivially breaking message confidentially. Message confidentiality is captured by the game in Figure 5.6.

First the adversary INITIALIZEs with a set of honest user identities. The challenger sets up the public key infrastructure (PKI) and generates long-term keys for the honest users. The adversary is allowed to register additional users and long-term keys with the PKI. SEND is called by the adversary to send network messages from entity S to entity R. The adversary is also allowed to instruct users to SETUP, SENDGROUPMESSAGE, and UPDATEPARTICIPANTS, to setup a session, send group messages, and update the set of participants in a session. Additionally, the adversary is allowed to reveal the long-term and ephemeral secret keys of any participant or server with REVEALLONGTERMKEYS and REVEALEPHEMALKEYS. The adversary may issue a single TEST query where the challenger flips a coin and sends either the encrypted message or a random ciphertext. Finally, the adversary calls FINALIZES providing its guess of the bit. The adversary wins if it guesses correctly.

To prevent the adversary from trivially wining it is not allowed to:

- Control a participant in the target session at the time of the target message.
- Call REVEALLONGTERMKEYS **and** REVEALEPHEMERALKEYS of the sender P and a receiving participant $R \neq P$ in session Sid. This does allow the adversary to compromise the long-term and ephemeral keys between receivers.

The advantage of adversary M is defined as $Adv^{conf}(M) = Pr[M\text{wins}] - \frac{1}{2}$.

Mobile CoWPI provides message confidentiality if all hash and key derivation functions are modeled as random oracles.

For any message confidentiality adversary M that runs in time at most t and creates sessions with at most w users. We show that there exists a NAXOS adversary M_0, an IND$-CPA adversary M_1, and an IND$-CPA adversary M_3 such that

$$\begin{aligned} Adv^{conf}(M) \leq & w-1 \cdot Adv^{NAXOS}(M_0) \\ & + w-1 \cdot Adv^{IND-CPA}(M_1) \\ & + Adv^{IND-CPA}(M_3) \end{aligned}$$

Where M_1, M_0, and M_3 run in time $O(t)$.

Proof. We prove Mobile CoWPI provides message confidentiality in a sequence of games:

G_0 The challenger behaves correctly.

$G_1.i$ The challenger replaces the NAXOS key exchange in the ciphertext block between the sender and the i^{th} receiver of the test message.

$G_2.i$ The challenger replaces the first ciphertext block between the sender and the i^{th} receiver of the test message with a random bit string.

G_3 The challenger replaces the ciphertext block of the test message with a random bit string.

The first games show the adversary can not learn the NAXOS keys of the ciphertext block of the test message, the second games show the adversary can not learn the key used to encrypt the test message, and the final game shows the adversary cannot distinguish the test message from random. Thus the protocol transcript is effectively random.

Let $G_1.0 = G_0$. We now construct a challenger M_0 that given a distinguisher D_0 that can distinguish between playing $G_1.i$ and $G_1.i+1$ with probability S_0, M_0 can win the NAXOS game.

The challenger M_0 plays $G_1.i$ in the following way:

- During INITIALIZE the challenger initializes a NAXOS game and setups the PKI for U.
- When REVEALLONGTERMKEYS(T) is called, M_0 returns LONG-TERM KEY REVEAL(T) of the NAXOS game.
- The challenger plays the NAXOS game replacing all NAXOS keys as detailed next.
- When REVEALEPHEMERALKEYS(Sid, A, B) is called, M_0 returns EPHEMERAL KEY REVEAL((A, B, epk_{ab}) of the NAXOS game for the most recent NAXOS session between A and B in Sid.
- When D_0 finalizes the game and guesses $G_1.i$, M_0 finalizes the NAXOS game and 0. If D_0 guesses $G_1.i + 1$, M_0 guesses 1.

We now describe how M_0 computes the NAXOS key of the ciphertext block between the sender and the receiver.Let A be the sender of the block and B the receiver. Compute the key as follows:

1. $epk_{ab} \leftarrow \textsc{Send}(A, B)$
2. $epk_{ba} \leftarrow \textsc{Send}(A, B, epk_{ab})$, epk_{ba} may be a pre-key of B.
3. When computing a NAXOS key of a ciphertext block not part of the test message, $k_{ab} \leftarrow \textsc{Reveal}(A, B, epk_{ab}, epkba)$ is used as the key.
4. When computing the NAXOS key of the i^{th} ciphertext block of the test message, $k_{ab} \leftarrow \textsc{Test}(A, B, epk_{ab}, epk_{ba})$ and is used by A to encrypted the ciphertext block and B to decrypt it.

M_0 wins the NAXOS game if D_0 guesses correctly. Thus the advantage of M_0 is $Adv^{NAXOS}(M_0) = S_0$. The advantage of distinguishing between $G_1.0$ and $G_1.w - 1$ is at most $Adv^{NAXOS}(M_0) \cdot w - 1$.

Let $G_2.0 = G_1.w - 1$. We now construct a challenger M_1 that given a distinguisher D_1 that can distinguish between playing $G_2.i$ and $G_2.i + 1$ with probability S_1, M_1 can win the IND$-CPA game.

The challenger M_1 plays $G_2.i$ in the following way:

- During Initialize the challenger initializes an IND$-CPA game.
- The challenger replaces the ciphertext block of the test message between the sender and the i^{th} receiver with an IND$-CPA Test query detailed next.
- When D_1 finalizes the game and guesses $G_2.i$, M_1 finalizes the IND$-CPA game and 0. If D_1 guesses $G_2.i + 1$, M_1 guesses 1.

We now detail how the challenger M_1 generates the ciphertext block between the sender and the i^{th} participant. Let A be the sender of the block and B the receiver. Let m be the plaintext to be encrypted by the block, d the associated data, and id_{ba} the id of B's ephemeral public key used to compute the key. The blocks is generated as follows:

1. $c_{ab} \leftarrow id_{ba}$, $\textsc{Test}(m, d)$.

2. When B receives c_{ab}, it uses m and d as the plaintext and associated data respectively.

M_1 wins the IND$-CPA game if D_1 guesses correctly. Thus the advantage of M_1 is $Adv^{IND\$-CPA}(M_1) = S_1$. The advantage of distinguishing between $G_2.0$ and $G_2.w-1$ is at most $Adv^{IND\$-CPA}(M_1) \cdot w - 1$.

We now construct a challenger M_2 that given a distinguisher D_2 that can distinguish between playing $G_2.w-1$ and G_3 with probability S_2, M_2 can win the IND$-CPA game.

The challenger M_2 plays G_3 in the following way:

- During INITIALIZE the challenger initializes an IND$-CPA game.
- The challenger replaces the ciphertext of the test broadcast message an IND$-CPA TEST query detailed next.
- When D_2 finalizes the game and guesses $G_2.w$, M_2 finalizes the IND$-CPA game and 0. If D_2 guesses G_3, M_2 guesses 1.

M_3 constructs the protocol message as follows:

1. $c \leftarrow \text{TEST}(m, \cdot)$.
2. The protocol message is thus Sid, "MSG", $P, c, c_{p*}, \ldots, auth_{p*}, \ldots$.
3. When the participants receive the sent protocol message with c they use m as the plaintext.

M_2 wins the IND$-CPA game if D_2 guesses correctly. Thus the advantage of M_2 is $Adv^{IND\$-CPA}(M_2) = S_2$. We have now shown that the protocol output is indistinguishable from random. □

5.6.3 Message Integrity and Authentication

Message authentication and integrity is the property that receivers can verify the author of a messages and are confident that the messages has not been modified in transit. Message authentication implies message integrity. Mobile CoWPI provides message authentication under an adversary that may compromise the servers or participants as well as control the network. Message authentication is provided as long as the adversary

Figure 5.7: Message Authentication Game

function INITIALIZE(U, C)
Initialize PKI for all users in and servers U.
Initialize $Out[P] \leftarrow \{\}$ for $P \in U$

function SEND(R, S, m)
Send m to R from S where R and S may be participants or servers.
return Network output of R after processing m

function SETUPGROUP(Sid, P, U)
Setup session Sid as participant P for users U.
return Network output of P

function SENDGROUPMESSAGE(Sid, P, m)
Send message m from P to group Sid.
Record the broadcast protocol message pm output of P as $Out[P] \leftarrow Out[P] \cup \{pm\}$.
return Network output of P.

function UPDATEPARTICPANTS(Sid, P, U)
Send participant update message as P for participants U in session Sid.
return Network output of P.

function REVEALEPHEMERALKEYS(Sid, A, B)
return The ephemeral secret keys of A that A uses for communication with B in session Sid. A or B may be users or servers. If A or B is a server, Sid is ignored.

function REVEALLONGTERMKEYS(T)
return The Long-term keys of T where T may be a server or participant.

function FINALIZE
return $True$ iff there exist clients R,P, session id Sid, index i and protocol message pm such that $(Sid, P, i, pm) \in M_R$, $pm \notin Out[P]$, and R and P are clean.

cannot trivially break the authentication. That is the adversary is not allowed to control the sender or have revealed the long-term **and** ephemeral keys for the target message.

Figure 5.7 captures the message authentication and integrity property in a game similar to message confidentiality. The adversary first INITIALIZEs the PKI and can register adversary controlled users and long-term keys. The adversary controls the network and uses the SEND function to send messages between users and servers. The adversary may also instruct honest users to SETUPGROUP, SENDGROUPMESSAGE, and UPDATEPARTICIPANTS as with message confidentiality. The adversary is allowed to REVEALLONGTERMKEYS and REVEALEPHEMERALKEYS of users. Finally, the adversary FINALIZES the game and wins if a participant R accepted protocol broadcast message pm from P in session Sid where the P did not send c and R and P have not had their long-term and ephemeral keys of ciphertext block of pm revealed.

That is R must have received a message:

$$Sid, \text{“}MSG\text{”}, P, c, c_{PR}$$

Where c_{PR} is the ciphertext block used to authenticate pm with AEAD from P.

To avoid trivially winning the game the adversary is not allowed to:

- Control the sender of the winning protocol message.
- Issue REVEALLONGTERMKEYS and REVEALEPHEMERALKEYS of the sender or receiver of the winning protocol message.

The advantage of an adversary M is defined as $Adv^{auth}(M) = Pr[M\text{wins}]$.

Mobile CoWPI provides message authentication and integrity if all hash and key derivation functions are modeled as random oracles.

For any message authentication adversary M that runs in time at most t, w is the maximum number of participants in a session, q is the maximum number of messages received in a session, y is the maximum number of sessions. We show that there exists a NAXOS adversary M_0 and an INT-CTXT adversary M_1 such that

$$\begin{aligned} Adv^{auth}(M) \leq & \frac{1}{(w-1)qy} \cdot Adv^{NAXOS}(M_0) \\ & + Adv^{INT-CTXT}(M_1) \cdot \frac{1}{(w-1)qy} \end{aligned}$$

Where M_0 and M_1 run in time $O(t)$.

Proof. We prove Mobile CoWPI provides message authentication in a sequence of games:

G_0 The challenger behaves correctly.

G_1 The challenger replaces the NAXOS key exchange used to decrypt a random ciphertext block between the sender and a random receiver of a random forged message with a random key.

G_2 The challenger replaces the ciphertext block of a forged message between the sender and a random receiver with an instance of the INT-CTXT game.

Game G_1 shows the adversary can not learn the NAXOS keys between users and is used as a transition to a game that M_1 can play.

We construct a challenger M_0 that given a distinguisher D_0 that can distinguish between playing G_0 and G_1 with probability S_0, M_0 can win the NAXOS game.

The challenger M_0 deviates from G_0 in the following way:

- During INITIALIZE the challenger initializes a NAXOS game and setups the PKI for U.
- When REVEALLONGTERMKEYS(T) is called, M_0 returns LONG-TERM KEY REVEAL(T) of the NAXOS game.
- The challenger plays the NAXOS game replacing all NAXOS keys as detailed next.
- When REVEALEPHEMERALKEYS(Sid, A, B) is called, M_0 returns EPHEMERAL KEY REVEAL((A, B, epk_{ab}) of the NAXOS game for the most recent NAXOS session between A and B in Sid.
- When D_0 finalizes the game and guesses G_0, M_0 finalizes the NAXOS game and 0. If D_0 guesses G_1, M_0 guesses 1.

We now describe how M_0 computes the NAXOS key of the ciphertext block between the sender and the receiver.Let A be the sender of the block and B the receiver. Compute the key as follows:

1. $epk_{ab} \leftarrow$ SEND(A, B)
2. $epk_{ba} \leftarrow$ SEND(A, B, epk_{ab}), epk_{ba} may be a pre-key of B.

3. When computing a NAXOS key of a ciphertext block not part of the test message, $k_{ab} \leftarrow \text{REVEAL}(A, B, epk_{ab}, epkba)$ is used as the key.

4. When computing the NAXOS key of the ciphertext block of the of a received protocol message that was not sent, $k_{ab} \leftarrow \text{TEST}(A, B, epk_{ab}, epk_{ba})$ and is used by B to decrypt the ciphertext block.

M_0 wins the NAXOS game if it guesses the correct forged message, correct receiver, and D_0 guesses correctly. Thus the advantage of M_0 is $Adv^{NAXOS}(M_0) = S_0$. The advantage of distinguishing between G_1 and G_1 is at most $Adv^{NAXOS}(M_0) \cdot \frac{1}{(w-1)qy}$.

We now construct a challenger M_1 that given a an adversary M that can win the authentication game S_1, M_1 can win the INT-CTXT game.

The challenger M_1 behaves as follows:

- During INITIALIZE the challenger initializes an INT-CTXT game.

- The challenger guesses a random sent message in a random session and guesses a random receiver of the message. Then challenger replaces the instance of the ciphertext block with a query to ENC(m, d) of INT-CTXT game.

- When the challenger receives an unsent protocol message in the chosen session from the chosen sender, it submits the ciphertext block between the sender and chosen recipient to VF of the INT-CTXT game.

M_1 wins the INT-CTXT game if it guesses session, protocol message, and receiver of a forged message correctly and M wins. Thus the advantage of M_1 is $Adv^{INT-CTXT}(M_1) = S_1 \cdot \frac{1}{(w-1)qy}$.

□

5.6.4 Conversation Integrity

Conversation integrity is the property that all users see all messages in the same order. Since participant update messages are treated the same as conversation messages, participant consistency is implied. The adversary is allowed to compromise all but one of the OES providers and any of the participants. Conversation integrity is provided between honest participants.

Figure 5.8: Conversation Integrity Game G_0

function INITIALIZE(U)
Initialize infrastructure and PKI for all users and servers in U.

function SEND(R, S, m)
Send m to R from S where R and S may be participants or servers.
return Network output of R after processing m

function SETUPGROUP(Sid, P, U)
Setup session as participant P for users U.
return Network output of P

function SENDGROUPMESSAGE(Sid, P, m)
Send message m from P to group Sid.
return Network output of P.

function UPDATEPARTICPANTS(Sid, P, U)
Send participant update message as P for participants U in session Sid.
return Network output of P.

function REVEALEPHEMERALKEYS(Sid, A, B)
return The ephemeral secret keys of A that A uses for communication with B in session Sid. A or B may be users or servers. If A or B is a server, Sid is ignored.

function REVEALLONGTERMKEYS(T)
return The Long-term keys of T where T may be a server or participant.

function FINALIZE()
return $True$ iff there exist honest users A,B, session Sid, and index i such that $(Sid, s_a, i, pm_a) \in M_A$, $(Sid, s_b, i, pm_b) \in M_B$, and $pm_a \neq pm_b$.

Figure 5.8 details the conversation integrity game. First the adversary INITIALIZES the PKI and registers corrupt users and providers. The adversary may then issue commands instructing participants and providers to execute protocol operations the same way as the previous two games. Finally, the adversary wins the game if he convinces two participants A and B of session Sid to accept different messages as the i^{th} message.

To avoid trivially winning the game the adversary is not allowed to:

- Issue REVEALLONGTERMKEYS and REVEALEPHEMERALKEYS of all the OES providers and one of A or B.

The advantage an adversary M has at winning the game is defined as $Adv^{INT-CONV}(M) = Pr[M \text{ wins}]$.

Recall from Section 5.3 the probability of an adversary finding a protocol ciphertext that successfully decrypts under two separate keys is at most $q\epsilon_{int}$. If an adversary cannot constructs such a message they must be able to forge a message from an honest server to an honest participant indicating that an out-of-order protocol message should be processed.

Mobile CoWPI provides conversation integrity if all hash and key derivation functions are modeled as random oracles.

For any conversation integrity adversary M that runs in time at most t, performs at most q KDF_2 oracle queries and sends at most y messages between honest OES providers and honest participants. We show that there exists a NAXOS adversary M_0 and an INT-CTXT adversary M_1 such that

$$\begin{aligned} Adv^{INT-CONV}(M) \leq & \frac{1}{y} \cdot Adv^{NAXOS}(M_0) \\ & + Adv^{INT-CTXT}(M_1) \cdot \frac{1}{y} \\ & + q\epsilon_{int} \end{aligned}$$

Where M_0 and M_1 run in time $O(t)$.

Proof. We prove Mobile CoWPI provides conversation integrity in a sequence of games:

G_0 The challenger behaves correctly.

G_1 The challenger replaces the NAXOS key exchange used to create a random OES authentication block between an honest server and participant with a random key.

Games G_1 show the adversary can not learn the NAXOS keys used in the OES authentication block and is used as a transition to a game that M_1 can play. If if M can win the conversation integrity game, then M_1 can win the INT-CTXT game.

We construct a challenger M_0 that given a distinguisher D_0 that can distinguish between playing G_0 and G_1 with probability S_0, M_0 can win the NAXOS game.

The challenger M_0 deviates from G_0 in the following way:

- During INITIALIZE the challenger initializes a NAXOS game and sets up the PKI for U.
- When REVEALLONGTERMKEYS(T) is called, M_0 returns LONG-TERM KEY REVEAL(T) of the NAXOS game.
- The challenger plays the NAXOS game replacing all NAXOS keys as detailed next.
- When REVEALEPHEMERALKEYS(Sid, A, B) is called, M_0 returns EPHEMERAL KEY REVEAL((A, B, epk_{ab}) of the NAXOS game for the most recent NAXOS session between A and B in Sid.
- When D_0 finalizes the game and guesses G_0, M_0 finalizes the NAXOS game and 0. If D_0 guesses G_1, M_0 guesses 1.

We now describe how M_0 computes the NAXOS key in the OES authentication block honest servers and participants. Let A be the participant and B the server. Compute the key as follows:

1. $epk_{ab} \leftarrow$ SEND(A, B)
2. Send epk_{ab} to B.
3. Upon B receiving epk_{ab}, $epk_{ba} \leftarrow$ SEND(A, B, epk_{ab}), epk_{ba}.
4. When computing a NAXOS key of a OES authentication block not part of the test message,
$$k_{ab} \leftarrow \text{REVEAL}(A, B, epk_{ab}, epk_{ba})$$
is used as the key.

5. When computing the NAXOS key of the OES authentication block of the of a received protocol message that was not sent, $k_{ab} \leftarrow \textsc{Test}(A, B, epk_{ab}, epk_{ba})$ and is used by B to decrypt the OES authentication block.

M_0 wins the NAXOS game if it guesses the OES authentication block correctly and D_0 guesses correctly. Thus the advantage of M_0 is $Adv^{NAXOS}(M_0) = S_0 \cdot \frac{1}{y}$.

We now construct a challenger M_1 that given an adversary M that can win the conversation integrity game with probability S_1, M_1 can win the INT-CTXT game.

The challenger M_1 behaves as follows:

- During Initialize the challenger initializes an INT-CTXT game.
- The challenger replaces a random OES authentication block between an honest OES provider and participant with an INT-CTXT game detailed next.

We now detail how the challenger M_1 generates the random OES authentication block between an honest OES provider and participant. Let A be the sender of the message and B the receiver. Let m be the plaintext to be encrypted by the block, d the associated data, and id_{ba} the id of B's last received ephemeral public key used to compute the key. The blocks is generated as follows:

1. $c_{ab} \leftarrow id_{ba}, \textsc{Enc}(m, d)$.
2. When B receives the next authentication block $auth'_{ab} \neq auth_{ab}$, the challenger submits $auth'_{ab}$ to VF of the INT-CTXT game.

M_1 wins the INT-CTXT game if it guesses the OES authentication block correctly and M wins the game. Thus the advantage of M_1 is $Adv^{INT-CTXT}(M_1) = S_1 \cdot \frac{1}{y}$. □

5.6.5 Deniability

We capture the deniability property with the general-purpose game detailed in Figure 5.9. The distinguisher Initializes the game with a plaintext transcript τ. Then the challenger executes Mobile CoWPI on τ producing a real protocol transcript T_0 and three outputs $input_d$, $input_s$, $output_0$. The challenger then runs a simulator with inputs τ and $input_s$ producing a forged protocol transcript T_1 and state $output_1$. The

Figure 5.9: Deniability Game G_0

function INITIALIZE(τ,S)
Initialize an PKI and executes the protocol on plaintext transcript τ producing protocol transcript T_0, and state information $output_0$.
Run the protocol simulator S with input τ and $input_s$ to produce protocol transcript T_1 and state information $output_1$
Flip a coin $b \leftarrow_R \{0,1\}$
return $(T_b, output_b, input_d)$

function FINALIZE(d)
return $(d == b)$.

challenger returns a random transcript T_b, output $output_b$, and $input_d$ to the distinguisher. The distinguisher wins the game if it guesses b correctly. The advantage of the distinguisher M is defined as $Adv^{DENY-*}(M) = Pr[M \text{ wins}] - \frac{1}{2}$. The DENY-* game depends on how $input_d$, $input_s$, and $output_*$ are defined.

When proving message deniability and participant deniability it is sufficient to define the inputs and output as follows:

$$\begin{aligned} input_d &= \{(lsk_0, epk_0) \ldots, (lsk_n, epk_n)\} \\ input_s &= \{(lpk_0, epk_0), \ldots, (lpk_n, epk_n)\}, (lsk_a, esk_a) \\ output_b &= \{esk_{a0}, esk_{aw}\} \end{aligned}$$

where n is the number of participants, a is the user running the simulator, and w is the number of ciphertext blocks where a is a participant. In this case the distinguished is provided with long-term secret keys and single use public pre-keys of all users in the transcript. The simulator is only given the public values and the secret values of a single user and must output all of a' ephemeral secret keys.

Mobile CoWPI provides message and participant deniability if all hash and key derivation functions are modeled as random oracles.

For any participant deniability adversary M that runs in time at most t, performs at most q H oracle queries and supplies a transcript that produces at most y ciphertext blocks between participants that are not the simulating participant. We show that there

exists a NAXOS adversary M_0 such that

$$Adv^{DENY-PART}(M) \leq y \cdot Adv^{NAXOS}(M_0) \tag{5.1}$$

Where M_0 runs in time $O(t)$ and $q < 2^l$.

Proof. Recall the Mobile CoWPI simulator discussed in Section 5.3. We prove the simulated transcript is indistinguishable from the real transcript in a sequence of games. In each game we replace an additional NAXOS key agreement, between two parties that are not the simulating party, from the real transcript with a random NAXOS key. In the final game the real transcript is generated in the same way as the simulated one.

Below is the sequence of games:

G_0 The challenger behaves correctly.

$G_1.i$ The challenger replaces the NAXOS key exchange used to encrypt the ith ciphertext block between two user that are not the simulating user.

Game $G_1.i$ shows the adversary cannot distinguish between a simulated and real NAXOS key agreement and is used as a transition to a game that M_1 can play. If M can win the participant deniability game, then M_1 can win the NAXOS game.

Let $G_1.0 = G_0$, we construct a challenger M_0 that given a distinguisher D_0 that can distinguish between playing $G_1.i$ and $G_1.i$ with probability S_0, M_0 can win the NAXOS game.

The challenger M_0 deviates from $G_1.i - 1$ in the following way:

- During INITIALIZE the challenger initializes a NAXOS game and sets up the PKI for U and issues REVEALLONTERMKEYS for all U.
- The challenger replaces the first $i - 1$ NAXOS keys between non-simulating participants with random keys.
- The challenger plays the NAXOS game replacing the ith NAXOS key between non-simulating participants with a NAXOS TEST query detailed next.
- When D_0 finalizes the game and guesses $G_1.i - 1$, M_0 finalizes the NAXOS game and 0. If D_0 guesses $G_1.i$, M_0 guesses 1.

We now describe how M_0 computes the ith NAXOS key. Let B and C be the participants. Compute the key as follows:

1. $epk_{bc} \leftarrow \textsc{Send}(B, C)$
2. Send epk_{bc} to C.
3. Upon C receiving epk_{bc}, $epk_{cb} \leftarrow \textsc{Send}(C, C, epk_{bc})$, epk_{cb}.
4. When computing a NAXOS key of all ciphertext blocks after the ith $k_{bc} \leftarrow \textsc{Reveal}(B, C, epk_{bc}, epkcb)$ is used as the key.
5. When computing the NAXOS key of the ith ciphertext block , $k_{bc} \leftarrow \textsc{Test}(B, C, epk_{bc}, epk_{cb})$.

M_0 wins the NAXOS game if D_0 guesses correctly. Thus the advantage of M_0 is $Adv^{NAXOS}(M_0) = S_0$. There are y ciphertexts blocks the between non-simulating participants. □

5.6.6 Message Unlinkability

We now detail message unlinkability provided by Mobile CoWPI. Compared to participant deniability we consider a stronger definition where the distinguisher is given a real protocol message and the ephemeral public keys of the sender for the message. The simulator is given the ephemeral secret key used to encrypt the message.

$$\begin{aligned} input_d =& lsk_s, (lsk_0, epk_1) \ldots, (lsk_n, epk_n), epk_{si}, i, pm_i \\ input_s =& lpk_s, (lpk_0, epk_1) \ldots, (lpk_n, epk_n), \\ & lsk_a, esk_{ai}, epk_{si}, i, pm_i \\ output_b =& \{esk_{a0}, esk_{aw}\} \end{aligned}$$

Where epk_n is the ephemeral secret key of receiver n shared with the sender, esk_{si} is the ephemeral secret key of the sender shared with the simulating party for the ith message, esk_{ai} is the secret key of the simulation party for the ith message, and i is the index of the protocol message pm_i in the transcript. The simulator must output all of a's ephemeral keys.

This definition provides the distinguisher with knowledge of a non-deniable protocol message. The goals is to simulate a transcript that contains pm_i and is identically

distributed to the real. The message unlinkability simulator behaves as a participant repudiation simulator discussed earlier. When the simulation party sends its' last ciphertext block prior to pm_i the simulator uses $epk_{ai} \leftarrow g^{H(esk_{ai}, lsk_a)}$ as the next ephemeral public to the sender. Similarly, when the sender of pm_i sends it last ciphertext block to the simulating party it uses epk_{si} as it next ephemeral public key. The simulator then sends pm_i as the i^{th} message in the transcript. The simulator then continues to behave the same as the participant deniability simulator from earlier. The simulated transcript is identically distributed to the real transcript and contains the undeniable message pm_i in position i. The proof is identical to the proof of participant deniability.

Chapter 6

Private Presence

While secure messaging hides the content of the conversation from the provider by using strong cryptographic techniques, but the metadata of the conversation is still known to the service provider. A critical part of secure messaging is presence, i.e., knowing when a friend is online. Although secure messaging providers do not have access to the plaintext conversation, they still have access to the set of friends of every user. This means that these services know the social graph of their users as well as the presence status of every user at any given time. To have a truly private communication platform, the metadata of the communication must also be protected.

An existing solution to this problem is DP5—the Daghstul Privacy Preserving Presence Protocol P—proposed by Borisov, Danezis, and Goldberg [12]. DP5 is the first private presence protocol to leak no information about the social graph to third parties and limit the information retained by the service itself. DP5 allows its users to see the online status of their friends in a completely private manner. It utilizes Private Information Retrieval (PIR) [58] for querying the service for a given user's buddy's presence.

The major weakness of DP5 is its *lack of* scalability for a large number of users. The presence database of DP5 grows much more rapidly than the number of users of the service. This results in a very expensive service for even a small number of users. To overcome this bottleneck, we propose MP3—the Minnesota Private Presence Protocol. MP3 maintains the same security goals as DP5. This is elaborated in Section 6.1.3 and Section 6.2.7. Compared to DP5, MP3 makes an assumption that revoking and unrevoking friends is uncommon and thus we are able to significantly reduce the size of the presence database by using a *dynamic broadcast encryption scheme* [13]. As a result, MP3 is significantly cheaper to run for even a relatively modest user base. Additionally, these savings increase as the number of users increase. The *key contribution* of this chapter lies in significantly reducing the size of the presence database compared to DP5; this allows cheaper registration and lookup queries in the context of the bandwidth required. The client-facing latency of MP3 is also an order of magnitude less than that of DP5 due to the smaller presence database.

This chapter is organized as follows. Section 6.1 describes the background, goals, and related work pertaining to the MP3 protocol. Section 6.2 presents a detailed description of the MP3 protocol. Section 6.3 analyzes the performance of MP3 and compares it to

that of DP5.

6.1 Goals

The primary functionality of a private presence protocol is to allow for the registration of one's online presence and to allow for the query of the presence status of one's buddies in a completely private manner.

6.1.1 DP5 Overview

Since our design shares many characteristics with DP5, we give a brief overview of the protocol here. DP5 uses Private Information Retrieval (PIR), in which a database is distributed to several servers so that a user can query the servers to retrieve a specific record without revealing which record they retrieve. Given this functionality, a "trivial" private presence protocol would have each user A with n_A friends encrypt n_A *presence records* recording their status (and possibly other information, such as a contact address), with a shared key for each friend, and periodically upload this information to a presence database. When A's friend B wants to check on A's status, they would query the current database (using PIR) for the presence records encrypted under the symmetric key A shares with B. To hide information about the social graph, each user would need to pad the number of presence records uploaded per period to some maximum value denoted by N_{fmax}. This protocol results in a nearly quadratic-size database in the number of users. Moreover, the server-side computational costs of PIR scale linearly in the size of the database (and the bandwidth costs also increase, though sub-linearly).

To combat this inefficiency, DP5 splits the presence service into two asymmetric services. The primary, short-term, service is used to register and query presence of users with the precision of short windows (on the order of minutes). The secondary service is the long-term service, which is used to provide metadata for querying during the short-term. The long-term service also provides friend registration and revocation with the precision of long windows (on the order of days). As in the "trivial" protocol, it is assumed that users share a unique secret with each of their friends. Additionally, in order to not leak information about how many friends a given user has, DP5 defines a limit of N_{fmax} as the maximum number of friends a user may have.

In each period of the long-term service, a user (let's say Alice) of DP5 will upload her presence to the registration mechanism of DP5. This is referred to as Alice's long-term presence record. This record is actually *several* records, one for each of Alice's friends, padded with random records upto N_{fmax}. If, N_{fmax} is on the order of the number of users of the service, then the long-term presence database scales quadratically with the number of users, which in turn increases the amount of bandwidth and CPU this service requires. These long-term records contain information used by her friends to identify her during the short-term period. Then, during the short-term period, Alice uploads a single record to the short-term presence database in each short-term period she is online. Additionally, a *single* record containing a signature is uploaded to an auditable *signature database* during every short-term period. Thus, the short-term service, which is queried more often, grows only linearly with the number of online users.

To improve the DP5 protocol, we address the issue of scaling in the long-term service of DP5 by reducing the number of records Alice uploads in each long-term period to only a relatively constant number of records, yielding performance closer to that of the short-term service. We leave the short-term service of DP5 unchanged as it is already quite cheap.

6.1.2 Threat Model

We make standard assumptions about the users and adversaries of MP3. They are real world adversaries with common capabilities.

- We assume that honest users' end systems are secure and not compromised. We also assume that honest servers can maintain secrecy and integrity. Our design maintains forward security and does not require servers to store any long-term secrets.

- We allow the adversary to be an observer or a dishonest user of the system, and we assume they have not made any recent breakthrough in computational cryptographic assumptions, and assume that they cannot distinguish between different ciphertexts. More detailed assumptions are described in the protocol description in Section 6.2.

- Our security properties are under the covert model, i.e., adversaries will not act dishonestly if it would cause them to be detected and identified.
- Our protocol maintains availability against malicious parties.

6.1.3 Security Goals

Here, we describe the goals required for a private presence service.

Privacy of presence and auxiliary data. The presence status of a user and their auxiliary data should be available to only that user's explicit friends.

Integrity of authentication and auxiliary data. The friends of Alice should not accept the presence and auxiliary data unless it was submitted by Alice.

Unlinkability of sessions. It should be infeasible for a user to be linked between multiple uses of the service. The infrastructure and non-friend users should not be able to link the presence of another between epochs.

Privacy of the social graph. No information about the social graph of a user should be revealed to any other party of the service. More specifically, friends should not learn about other friends and the infrastructure should not learn any new information about a user.

Forward/Backward secrecy of the infrastructure. Any compromised keys stored in the infrastructure servers should not reveal past or future information that is secured with previous or future keys.

Auditability. All operations performed by the infrastructure should be auditable. A user should detect when their friend registration or presence registration has not been performed honestly by the service provider.

Support for anonymous channels. The protocol should not require any identifying information for operation. The use of an anonymous channel should only enhance the privacy of the system and the service will not compromise the anonymity of the user.

Indistinguishability of revocation, suspension, and offline status. A user is revoked if they are no longer able to query the presence status of the buddy that revoked them. A suspension is a temporary revocation, i.e., for some period of time, a user cannot query the presence status of the buddy that suspended them. This

means a suspended buddy can be "unrevoked." Revocations and suspensions should not be distinguishable from being offline. For example, if Bob's buddy Alice appeared to be offline, Bob would not know if he was revoked or suspended, or if Alice was genuinely offline. MP3 provides *plausibly deniable* revocation and suspension of buddies. Plausibly deniable revocation means the transcript does not prove a user has been revoked. However, If a user does not see their friend come online for an extended period of time they may begin to assume they have been revoked. This is discussed in further detail in Section 6.2.7.

6.1.4 Related Work

DP5 seems to have been the first and only previous work to address social graph privacy in the context of presence services. Similar, but different, related work include Dissent [59] and Riposte [60] which offer anonymous micro-blogging services; these systems are similar to private presence in that posting a micro-blog implies the author was online. Dissent is based on a DC-net with a client-server architecture. Clients in Dissent must form a group to post anonymous messages for each other using distrusted servers. Dissent provides anonymity within a static group. Riposte utilizes a novel private database writing mechanism based on techniques of PIR. Both of these systems have high latency when dealing with large anonymity sets and are not concerned with social graph privacy.

Two other similar and relevant anonymous messaging/microblogging systems that build on PIR techniques include Riffle [61] and Pung [62]. These systems allow for a user to send a message to their friends without revealing the social graph of the users. These message could be used to indicate presence. However, these two systems assume every user uploads a message during every epoch. This implies that all users must be present at all times which is unrealistic and negates the need for a private presence protocol.

6.2 The MP3 Protocol

6.2.1 Overview

In this section, we provide an overview of MP3. MP3 is composed of two databases, a long-term database and a short-term database. The short-term database contains the presence status and information of a user. A new short-term database is generated on the order of minutes (5 minutes), we refer to these as short-term epochs. The long-term database contains information for a user to identify their friends short-term database entries. A new long-term database is generated less frequently (once every 24 hours), these are referred to as long-term epochs. When Bob wants to check if his friend Alice is online, he first queries the long-term database and retrieves Alice's entry. Then he computes her short-term identifier and queries the short-term database for her presence. Each long-term database entry of a user contains information for looking up the next long-term database entry of that user. For this reason, MP3 keeps the most recent long-term databases corresponding to 30 days.

The database entries are simple (key, value) pairs where the key is a unique identifier for a user in that specific epoch. These databases are queried using hash-based PIR protocol of retrieving (key, value) pairs. Since these database a queried with PIR the infrastructure does not learn any information about a user's queries. In our implementation users upload their database entries to a single registration server and use a distributed PIR protocol with N_{lookup} servers for database queries. When querying the long-term database the users must query all long-term databases to avoid revealing which database contained the entry they needed.

The short-term database contains a single entry (presence message) per online user. For Alice (a) we denote presence message as $m_a(j)$ during short-term epoch t_j. This presence message may contain information such as how to contact a or a public key. If $m_a(j)$ is not present in during t_j, Alice is assumed to not be online.

MP3 uses a Dynamic Broadcast Encryption (DBE) [13] scheme for long-term database entries. DBE allows Alice to create a single constant-sized ciphertext that can be decrypted by all of her friends. Before participating in MP3, Alice generates a single DBE encryption key (mk) and a decryption key (dk_{ai}) for each friend (i). During long-term epoch T_j Alice creates a DBE ciphertext with her short-term identity information for

all short-term epoch that occur during T_j.

The dynamic part of DBE allows Alice to revoke a decryption keys so the revoked keys can not decrypt future ciphertexts. We utilize this to allow Alice to revoke up to N_{rev} friends in each long-term epoch. To provide plausible deniability of revocation, Alice distributes new decryption keys to the revoked friends. If Alice wants to truly revoke the friend she gives them a new decryption key generated randomly. If they are revoked for deniability reasons she gives them a new correctly generated decryption key. For the rest of this chaper DBE.Revoke is used to refer to DBE revocations and MP3.Revoke is used to refer to Alice actually revoking a friend.

Alice may wish to unrevoke a friend she previously MP3.Revoked. MP3 allows here to unrevoke N_{unrev} friends per long-term epoch. Alice does this by creating a long-term database entry for the MP3.Revoked user that DBE.Revokes their random decryption key. she gave to them when they were MP3.Revoked. This means each long-term database for Alice consists of $N_{\text{unrev}} + 1$ entries, each of N_{rev} size. One entry to distribute short-term lookup information to her friends and possibly revoke N_{rev} friends. And N_{unrev} entries that allow her to unrevoke friends.

Finally, to prevent forged records, we employ two signature schemes, one for long-term presence records and another for short-term presence records. Thus, all of Alice's friends can be confident that the record they received from MP3's lookup mechanism can only belong to Alice. During long-term epochs, we use a digital signature, specifically using the Elliptic Curve Digital Signature Algorithm (ECDSA) [63]. During short-term epochs, the Boneh-Lynn-Shacham (BLS) [64] signature scheme is used. Moreover, an auditable *signature database* is employed during each short-term epoch.

6.2.2 Cryptographic Primitives

It is assumed that everyone participating in MP3 shares a set of known cryptographic primitives. Let G_1 and G_2 be two cyclic groups of prime order p and let G_T be a cyclic group also of prime order p. Denote $\mathbb{Z}_p$ as the ring of integers modulo p and denote $\mathbb{Z}_p^\times$ as the set of units in $\mathbb{Z}_p$, also denote $g \leftarrow G$ as randomly selecting an element g from a set G. An efficiently computable asymmetric pairing defined by the map $e : G_1 \times G_2 \to G_T$

is known so that for generators $g_1 \in G_1$, $g_2 \in G_2$ and for all $u, v \in \mathbb{Z}_p$

$$e(g_1^u, g_2^v) = e(g_1, g_2)^{uv}$$

The computational Diffie-Hellman problem and the computational co-Diffie-Hellman problem are assumed hard for G_1 and G_2. It is also assumed that our pairing is non-degenerate and that we have a type 3 pairing, i.e., the isomorphisms $G_1 \xrightarrow{\sim} G_2$ and $G_2 \xrightarrow{\sim} G_1$ are not efficiently computable [65].

MP3 utilizes the following:

- PRF_K, a keyed pseudorandom function that maps a short-term epoch timestamp to keys for AEAD described below.
- H_1, an efficiently computable hash function that maps the concatenation of the byte representations of long-term epoch timestamps and elements of G_T to elements of $\mathbb{Z}_p^\times$.
- H_2, an efficiently computable hash function that maps long-term public key to shared identifiers.
- H_3, an efficiently computable hash function that maps elements of G_T to elements of $\mathbb{Z}_p^\times$.
- H_4, an efficiently computable hash function that maps elements of G_1 to keys in the pseudorandom function above.
- H_5, an efficiently computable hash function that maps short-term epoch timestamps to elements of G_2.
- H_6, an efficiently computable hash function that maps elements of G_T to shared identifiers.
- $\mathrm{AEAD}_K^{\mathrm{IV}}(m)$, an authenticated encryption function where m is the message, K is the key, and IV is the initialization vector.

6.2.3 Dynamic Broadcast Encryption

In our construction of MP3, long-term epochs share many characteristics with a broadcast encryption scheme. In the context of MP3's long-term epoch, we use a *dynamic broadcast encryption* scheme with constant-size ciphertexts and decryption keys.

The definition of a Dynamic Broadcast Encryption (DBE) [13] is slightly different from a conventional broadcast encryption scheme in that it involves *two* authoritative parties: a *group-manager* and a *broadcaster*. The job of the group manager is to grant new users access to the group. The job of the broadcaster is to encrypt and transport messages to this group of users. When a message is encrypted, some members of this group can be revoked temporarily (suspended) or permanently (revoked) from decrypting the broadcasted message. Formally, a DBE scheme with revocation is a tuple of probabilistic algorithms (Setup, Join, Encrypt, Decrypt, Revoke, Update). These algorithms rely on an asymmetric pairing as described in Section 6.2.2.

Our design relies on the DBE scheme proposed by Delerablée et al. [13]. In our use case, every user is both a group-manager and a broadcaster. We introduce two additional operations (ShiftMK, ShiftDK) to support MP3's plausibly deniable revocation and suspension. Our modifications are detailed in Appendix 6.4. We provide the relevant components of DBE as it pertains to MP3 below:

- DBE.Setup() - generates a *manager key* as the tuple $mk \coloneqq (G, H, \gamma)$, where $G \leftarrow G_1$ and $H \leftarrow G_2$ are randomly selected generators and $\gamma \leftarrow \mathbb{Z}_p^\times$.

- DBE.Join($mk = (G, H, \gamma)$) - allows a user to friend someone by sharing a *decryption key* derived from their manager key as the tuple $dk \coloneqq (x, A, B)$, where $x \leftarrow \mathbb{Z}_p^\times$ is fresh, $A \coloneqq G^{\frac{x}{\gamma+x}}$, $B \coloneqq H^{\frac{1}{\gamma+x}}$. If x and γ happen to be inverses, resample x. In our construction of MP3, we add an additional component to the decryption key, κ, a shared random symmetric key for AEAD that is persistent even when (x, A, B) is reassigned. Thus, the decryption key derived is $dk \coloneqq (x, A, B, \kappa)$.

- DBE.Encrypt($mk = (G, H, \gamma)$) - generates a shared secret $K \coloneqq e(G, H)^w$ and two ciphertexts: $C_1 \coloneqq G^{w\gamma}$ and $C_2 \coloneqq H^w$, where $w \leftarrow \mathbb{Z}_p^\times$.

- DBE.Decrypt($dk = (x, A, B, \kappa), C_1, C_2$) - takes as input a decryption key and

the ciphertexts and computes the shared secret $K = e(C_1, B) \cdot e(A, C_2)$. Notice that this is the same shared secret computed by the broadcast manager in DBE.Encrypt.

- DBE.Revoke$(mk = (G, H, \gamma), x_r, B_r)$ - revokes the user with decryption key that contains x_r and B_r from the group of the user with manager key mk by updating $H \coloneqq H^{\frac{1}{\gamma + x_r}} = B_r$. The user then advertises x_r and B_r to all their buddies.

- DBE.Update$(dk = (x, A, B, \kappa), x_r, B_r)$ - every non-revoked user with decryption key $dk = (x, A, B, \kappa)$ must update their decryption key via $B \coloneqq \left(\frac{B_r}{B}\right)^{\frac{1}{x - x_r}}$ to revoke the user who owns x_r and B_r. Note that the revoked buddy will not be able to update their B value due to not being able to compute $\frac{1}{x_r - x_r}$. It is important to notice that this revocation is explicit to the revoked buddy, but in our construction of MP3 we add some auxiliary functionality in the long-term epoch to make revocations plausibly deniable as described in Section 6.2.7.

- DBE.ShiftMK$(mk = (G, H, \gamma), \lambda)$ - updates $G \coloneqq G^{\lambda}$ and $H \coloneqq H^{\lambda}$.

- DBE.ShiftDK$(dk = (x, A, B, \kappa), \lambda)$ - updates $A \coloneqq A^{\lambda}$ and $B \coloneqq B^{\lambda}$.

6.2.4 Setup

Initialization.

To participate in MP3, Alice generates a manager key

$$mk_a \coloneqq \text{DBE.Setup}()$$

She also generates a set of private-public base key pairs

$$(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}}) \quad \text{and} \quad (z_{a,\mathsf{init}}, q_{a,\mathsf{init}})$$

where $Y_{a,\mathsf{init}}$ is a randomly generated private key and $P_{a,\mathsf{init}} = g^{Y_{a,\mathsf{init}}}$ is an elliptic curve point, and $z_{a,\mathsf{init}}$ is a random private key randomly selected from $\mathbb{Z}_p^{\times}$ and $q_{a,\mathsf{init}} = g_1^{z_{a,\mathsf{init}}}$ is an element of G_1, $g_1 \in G_1$.

Adding buddies

For every friend f of Alice, she derives a decryption key

$$dk_{af} \coloneqq \text{DBE.}\textsc{Join}(mk_a)$$

and shares it along with her public base keys out-of-band. Assuming that the current long-term epoch is T_j, Alice also shares her most recent shared secrets K, from Encrypt, and R, her most recently generated random bit-string R, i.e., Alice shares[1] $(dk_{af}, P_{a,\mathsf{init}}, q_{a,\mathsf{init}}, K, R)$ with friend f out-of-band.

6.2.5 Long-term Epoch

Registration.

To register for epoch T_j, Alice must register during epoch T_{j-1}. To begin, Alice computes a new long-term private-public key pair for T_j

$$h_j \coloneqq H_1(T_j \parallel K_{j'} \oplus R_{j'}), \quad Y_a(j) \coloneqq Y_{a,\mathsf{init}} \cdot h_j, \quad P_a(j) \coloneqq P_{a,\mathsf{init}}^{h_j}$$

as well as a new short-term private-public key pair for all short-term epochs during T_j

$$z_a(j) \coloneqq z_{a,\mathsf{init}} \cdot h_j, \quad q_a(j) \coloneqq q_{a,\mathsf{init}}^{h_j}$$

where $K_{j'}$ is Alice's shared secret from calling DBE.Encrypt in epoch $T_{j'}$ and $R_{j'}$ is a random bit-string generated in epoch $T_{j'}$. Note that we use j' here since users need not register during *all* long-term epochs as described in Section 6.2.1; thus, $T_{j'}$ was the most recent long-term epoch in which Alice registered. The $\oplus$ between $K_{j'}$ and $R_{j'}$ implicitly converts $K_{j'}$ to a bit-string of some length and the length of $R_{j'}$ is defined to be that length.

During a long-term registration, Alice has the opportunity to revoke or suspend, or unrevoke a certain number of her buddies. Denote the number of buddies she can revoke or suspend at each long-term epoch as N_{rev} and the number of buddies she can unrevoke at each long-term epoch as N_{unrev}.

[1] If this is the very first epoch, Alice must generate an initial $K \coloneqq K_{\mathsf{init}}$ and $R \coloneqq R_{\mathsf{init}}$ to share with her buddies to bootstrap the protocol.

Every long-term epoch registration, Alice will upload $1 + N_{\text{unrev}}$ records to the long-term presence database. A single record is for all her buddies she wishes to continue being buddies with or that she wishes to revoke or suspend. The remaining N_{unrev} records are for buddies she had previously suspended and wishes to "re-friend." Alice constructs the single record according to Algorithm 13 and she constructs the other N_{unrev} records according to Algorithm 14.

All long-term records are of the form

$$\overbrace{P}^{(a)} \| \underbrace{x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}}}_{(b)} \| \overbrace{E_1 \| \cdots \| E_{N_{\text{rev}}}}^{(c)} \| \underbrace{C_1 \| C_2}_{(d)} \| \overbrace{R}^{(e)} \| \underbrace{S}_{(f)}$$

where (a) is a long-term identifier, (b) contains the x and B values of all buddies to revoke or suspend, (c) contains new encrypted decryption keys for all buddies revoked in (b), (d) contains the ciphertext components from DBE, (e) is a random bit string, and (f) is the signature for the record that can be verified with (a). Further details of how (b) and (c) are used to allow *plausibly deniable* revocation and suspension are described in Section 6.2.7.

Upon receiving a record of the form

$$P \| x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}} \| E_1 \| \cdots \| E_{N_{\text{rev}}} \| C_1 \| C_2 \| R \| S$$

from Alice, the registration server verifies the signature S with P. If the signature is valid, it computes $\text{ID}_a(j) \coloneqq H_2(P)$ and stores the $\langle$key, value$\rangle$ pair

$$\langle \text{ID}_a(j), x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}} \| E_1 \| \cdots \| E_{N_{\text{rev}}} \| C_1 \| C_2 \| R \| S \rangle$$

Otherwise, if the signature is invalid, nothing is stored.

Lookup

To look up Alice's presence for epoch T_j, Bob first requests the metadata associated with the databases from each lookup server. Since all lookup servers have the same database, the metadata should be the same, but in the event that some of the servers are dishonest, he takes the majority of the received metadata. The metadata contains information about the number of buckets and size of the buckets. He then computes

Algorithm 13 Alice computing her long-term presence record

Input:
$mk_a = (G_a, H_a, \gamma_a)$, Alice's manager key
$(Y_a(j), P_a(j))$, Alice's private-public key pair for T_j
$\mathcal{R}$, the set of buddies' decryption keys to MP3.REVOKE or suspend
$\mathcal{B}$, the set of buddies' decryption keys to continue being buddies
Output: Long-term presence record

1: **function**
2: $(C_1, C_2, K) \coloneqq \text{DBE.ENCRYPT}(mk_a)$
3: Generate and store a random bit-string R_j
4: $\lambda \coloneqq H_3(K)$
5: $\text{DBE.SHIFTMK}(mk_a, \lambda)$
6: Store $K_j \coloneqq K^\lambda$
7: $record \coloneqq P_a(j)$
8: $n \coloneqq N_{\text{rev}} - |\mathcal{R}|$
9: $\mathcal{B}_{\mathsf{padded}} \coloneqq$ pad $\mathcal{B}$ with random decryption keys upto N_{fmax}
10: $\mathcal{F} \coloneqq n$ decryption keys chosen uniformly from $\mathcal{B}_{\mathsf{padded}}$
11: **for each** $(x, A, B, \kappa) \in \mathcal{R} \cup \mathcal{F}$ **do**
12: $\text{DBE.GEVOKE}(mk_a, x, B)$
13: $record \coloneqq record \,\|\, x \,\|\, B$
14: **for each** $(x, A, B, \kappa) \in \mathcal{R}$ **do**
15: $x' \leftarrow \mathbb{Z}_p^\times$, $A' \leftarrow G_1$, $B' \leftarrow G_2$ ▷ Here, x' is fresh
16: ▷ Store the following in case we want to unrevoke this buddy
17: Store $((x', A', B', \kappa), C_1, C_2, R_j)$ in a global set $\mathcal{U}$
18: $E \coloneqq \text{AEAD}^j_\kappa(x' \,\|\, A' \,\|\, B')$
19: $record \coloneqq record \,\|\, E$
20: **for each** $(x, A, B, \kappa) \in \mathcal{F}$ **do**
21: ▷ generate new and valid x, A, and B
22: $(x', A', B', \_) \coloneqq \text{JOIN}(mk_a)$
23: $E \coloneqq \text{AEAD}^j_\kappa(x' \,\|\, A' \,\|\, B')$
24: $record \coloneqq record \,\|\, E$
25: Shuffle the concatenated encrypted triples $x \,\|\, A \,\|\, B$ in $record$
26: $record \coloneqq record \,\|\, C_1 \,\|\, C_2 \,\|\, R_j$
27: $S \coloneqq \text{ECDSA-SIGN}_{Y_a(j)}(record)$
28: $record \coloneqq record \,\|\, S$
29: **return** $record$

$P_a(j) \coloneqq P_{a,\mathsf{init}}^{H_1(T_j \| K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered, and subsequently computes $\text{ID}_a(j) = H_2(P_a(j))$. Finally, he builds a PIR request using the metadata for $\text{ID}_a(j)$ to retrieve a record of

Algorithm 14 Alice computing N_{unrev} presence records to unrevoke buddies

Input:
$\mathcal{U}$, the global set of buddies to unrevoke
T_j, the current long-term epoch
mk_a, Alice's manager key
$(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}})$, Alice's initial long-term epoch base keys
K_j stored from Algorithm 13
R_j stored from Algorithm 13
Output: list of N_{unrev} long-term presence records

1: **function**
2: $ret \coloneqq$ new list
3: **for each** $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}}) \in \mathcal{U}$ to unrevoke **do**
4: Remove $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}})$ from $\mathcal{U}$
5: $K_{\mathsf{revoked}} \coloneqq \text{DBE.}\textsc{Decrypt}((x, A, B, \kappa), C_1, C_2)$
6: $Y_{\mathsf{revoked}} \coloneqq Y_{a,\mathsf{init}} \cdot H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})$
7: $P_{\mathsf{revoked}} \coloneqq P_{a,\mathsf{init}}^{H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})}$
8: $R_{\mathsf{unrevoked}} \coloneqq K_j \oplus R_j \oplus K_{\mathsf{revoked}}$
9: $record \coloneqq P_{\mathsf{revoked}}$
10: ▷ We concatenate enough bytes so the record is the same length as that of Algorithm 13. We also make sure the byte encodings are of the correct type.
11: $record \coloneqq record \parallel \langle N_{\text{unrev}} - 1$ encrypted random triples $\overline{x} \leftarrow \mathbb{Z}_p^\times \parallel \overline{A} \leftarrow G_1 \parallel \overline{B} \leftarrow G_2 \rangle$
12: ▷ generate a fresh decryption key for the unrevoked buddy
13: $(x', A', B', \_) \coloneqq \textsc{Join}(mk_a)$
14: $record \coloneqq record \parallel \text{AEAD}_\kappa^j(x' \parallel A' \parallel B')$
15: Shuffle the record as in Algorithm 13 line 25
16: $record \coloneqq record \parallel C_1 \parallel C_2 \parallel R_{\mathsf{unrevoked}}$
17: $S_{\mathsf{unrevoked}} \coloneqq \text{ECDSA-}\textsc{Sign}_{Y_{\mathsf{revoked}}}(record)$
18: $record \coloneqq record \parallel S_{\mathsf{unrevoked}}$
19: store $record$ in ret
20: **if** number of buddies unrevoked $< N_{\text{unrev}}$ **then**
21: ▷ these records must be of the correct encoding
22: store random records in ret so that its length is N_{unrev}
23: **return** ret

the form

$$x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

Bob then processes this long-term record according to Algorithm 15 and stores the returned K and R values for computing Alice's long- and short-term identifiers in the

Algorithm 15 Bob processing the long-term record of Alice

Input:
$P_a(j)$, Alice's long-term key for T_j
$x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S$, long-term record
Output: K and R for the next long-term epoch

1: **function**
2: ▷ Only process the record if the signature is valid, otherwise the lookup server was malicious
3: **if** S is valid signature for the record with $P_a(j)$ **then**
4: *plausiblyRevoked* $\coloneqq$ **false**
5: **for** $i = 1$ **to** N_{rev} **do**
6: **if** $x_{ab} \neq x_i$ **then**
7: DBE.Update(dk_{ab}, x_i, B_i)
8: **else**
9: *plausiblyRevoked* $\coloneqq$ **true**
10: **if** *plausiblyRevoked* **then**
11: **for** $i \coloneqq 1$ **to** N_{rev} **do**
12: Try to decrypt E_i with κ_{ab}
13: **if** successfully decrypted E_i **then**
14: $(x', A', B') \coloneqq E_i$ decrypted with κ_{ab}
15: $x_{ab} \coloneqq x'$, $A_{ab} \coloneqq A'$, $B_{ab} \coloneqq B'$
16: ▷ Note that we updated dk_{ab} in line 15
17: $K \coloneqq$ DBE.Decrypt(dk_{ab}, C_1, C_2)
18: **return** K, R
19: **else**
20: $K \coloneqq$ DBE.Decrypt(dk_{ab}, C_1, C_2)
21: $\lambda \coloneqq H_3(K)$
22: DBE.ShiftDK(dk_{ab}, λ)
23: $K \coloneqq K^{\lambda}$
24: **return** K, R

next long-term epochs.

6.2.6 Short-term Epoch

Registration.

To register for epoch t_i, Alice must register during t_{i-1}. Assume that epoch t_i is during epoch T_j. Recall that Alice computed the private-public key pair $(z_a(j), q_a(j))$ during the long-term registration for T_j. To begin, Alice encrypts her presence message, $m_a(i)$

as follows:

$$k_a(i) \coloneqq \mathrm{PRF}_{H_4(q_a(j))}(t_i) \qquad\qquad c_a(i) \coloneqq \mathrm{AEAD}^i_{k_a(i)}(m_a(i))$$

She then computes the unforgeable signature:

$$s_a(i) \coloneqq H_5(t_i)^{z_a(j)}$$

Alice then uploads $c_a(i) \parallel s_a(i)$ to the short-term registration server.

Upon receiving Alice's record, the short-term registration server will compute $\mathrm{id}_a(i) = H_6(e(g_1, s_a(i)))$, and store $\langle \mathrm{id}_a(i), c_a(i) \rangle$. Additionally, $\langle \mathrm{id}_a(i), s_a(i) \rangle$ is stored in a short-term *signature database* to audit.

Lookup

To lookup Alice's presence for epoch t_i, Bob requests the metadata associated with the short-term databases in the same manner as in the long-term epoch. To begin, he first computes $q_a(j) \coloneqq q_{a,\mathsf{init}}^{H_1(T_j \| K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered. Then he computes $\mathrm{id}_a(i) \coloneqq H_6(e(q_a(j), H_5(t_i))$ which is equivalent to the registration server's computation of $H_6(e(g_1, s_a(i)))$ by the properties of pairings. He then builds a PIR request for $\mathrm{id}_a(i)$ to retrieve $c_a(i)$. Bob can be certain that $c_a(i)$ is from Alice due to the unforgeable signature $s_a(i)$ by auditing the signature database.

Bob can then compute $k_a(i) \coloneqq \mathrm{PRF}_{H_4(q_a(j))}(t_i)$ and decrypts $c_a(i)$ retrieving $m_a(i)$. The decryption being successful implies that Alice is online during epoch t_i. As in the long-term epoch, in order to not leak information of how many buddies Bob has, he must pad his lookup to N_{fmax} ids.

6.2.7 Details

Unrevokoe and Unsuspend

For simplicity in this section we assume $N_{\mathrm{rev}} = 1$. The long-term presence record of Alice that MP3.Revokes Bob takes the form

$$P \parallel x_{ab} \parallel B_{ab} \parallel E_{ab} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

That is x_{ab} and B_{ab} are Bob's x and B values and E_{ab} contains a triple of random (x, A, B). encrypted with κ_{ab}, i.e., x, A, and B were *not* generated from Alice's manager key. This means that for future epochs, Bob cannot compute the proper K (Algorithm 15 line 17), and thus can no longer query for Alice. Alice can unrevoke Bob by computing the incorrect K that he computes from when he was revoked (Algorithm 14 line 5) and use this to upload a record that Bob *can* query for. This record will allow Bob to compute the correct K and R values for Alice for future epochs, as well as providing Bob with fresh x, A, and B values that *are* generated from Alice's manager key (Algorithm 14 line 14). This is possible by storing the correct K value within the R value in this "unrevoking" record (Algorithm 14 line 8). This allows Bob to compute the proper identifier for Alice in the next long-term epoch. Thus, Bob can continue to query Alice's presence as before.

Alice's friends must process all of her long-term database entries so they must query all long-term databases. To allow users to be offline for extended periods of time MP3 stores the previous 30 days worth of long-term databases. If a user does not come online for more than 30 days they must share new keys with all of their friends. Revocations in the database entries are plausible deniable but a user may be able to notice if a friend does not come online anymore indicating they may have been revoked. This problem is inherent to presence systems.

Plausible Deniability of Revocation and Suspension

For MP3.Revoke to be deniable a revoked user must not be able to determine they have been revoked. MP3 implements this by DBE.Revoking users that have not been MP3.Revoked and issue these users new DBE decryption keys. Where as friends which are MP3.Revoked are issued a new random decryption key. For a friend to determine if they have been MP3.Revoked they must be able to distinguish between valid and random decryption keys. We introduce DBE.ShiftMK and DBE.ShiftDK to make the decryption keys indistinguishable. Section 6.4 details a distinguisher if DBE.ShitfMK and DBE.ShiftDK are not used.

More formally, If a friend can distinguish a transcript where they have been MP3.Revoked from a transcript where they have been DBE.Revoked but not MP3.Revoked they can be used as a Decisional Diffie-Hellman (DDH) distinguisher. That is, given (g, g^x, g^y, g^z)

determine if $g^{xy} = g^z$. We assume all hash functions are modeled as random oracles.

We now quickly sketch the proof. If given a decryption key (x, A, B) and ciphertext $(C_1 = G^{w_0\gamma}, C_2 = H^{w_0})$ and ciphertext $C_1' = G^{w_1\gamma\lambda}, H = ^{w_1\lambda})$ where w_0, γ, w_1, and λ are random, and given (x', A', B'), determine if (x', A', B') are valid decryption keys or random. Given a DDH challenger we can construct the above problem. Since w_0, γ, w_1, and λ are random and the group is of prime order we define $\lambda = y$ and let $C_1 = g$, $C_2 = g^x$, $C_1' = g^y$, and $C_2' = g^z$. If a friend can distinguish if they have been MP3.Revoked they can win the DDH game.

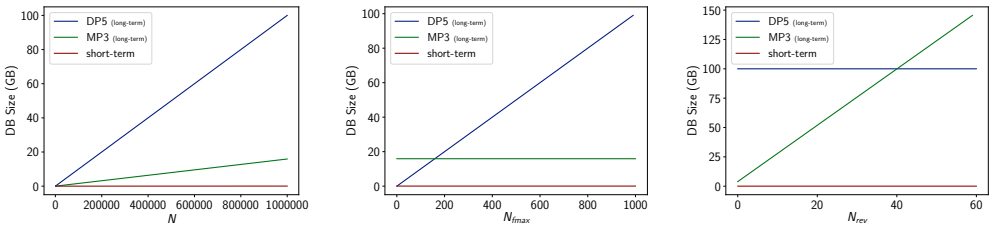
Complexity Comparisons

During each long-term epoch in DP5, $N \cdot N_{\text{fmax}}$ records are stored, where each record is a constant size. Thus, the registration bandwidth is $\Theta(N \cdot N_{\text{fmax}})$. During each long-term epoch in MP3, $N \cdot (N_{\text{unrev}} + 1)$ records are stored, where each record's length scales with N_{rev}. Thus, the registration bandwidth complexity is $\Theta(N \cdot N_{\text{unrev}} \cdot N_{\text{rev}})$. In reality, N_{unrev} and N_{rev} will be relatively constant[2] compared to N. This implies that (as functions of N and N_{fmax}) the registration bandwidth complexities for DP5 and MP3 are $\Theta(N \cdot N_{\text{fmax}})$ and $\Theta(N)$, respectively.

With the PIR protocol used in both DP5 and MP3, the bandwidth cost per query of a single record scales with the square root of the size of the database[3] . Also, recall that each user must query for N_{fmax} buddies to not reveal any information about their number of buddies. This implies that the bandwidth complexities for an entire long-term epoch (assuming all users query) for DP5 and MP3 are $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ and $\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$, respectively. Using the same approximations for N_{unrev} and N_{rev} as above, this results in lookup bandwidth complexities of $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ for DP5 and $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$ for MP3. Similar arguments can be made for the shared short-term epoch of DP5 and MP3 and are summarized in Table 6.1.

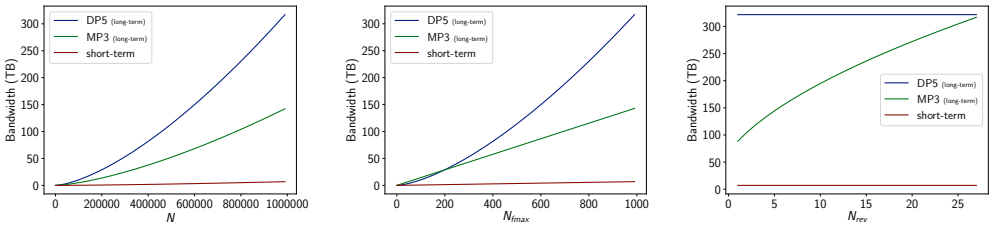
[2] Arguments for why this is a valid assumption are discussed in Section 6.5.

[3] As in the PIR protocol in DP5, we constructed $r = \lceil \sqrt{ns} \rceil$ buckets and we can upper bound the size of each bucket by $\left(\frac{n}{r} + \sqrt{\frac{n}{r}}\right) \cdot s \approx \sqrt{ns} + \sqrt[4]{ns^3}$ (here, s is the size of each record in bytes); since a query results in an entire bucket, this scales with the square root of the size of the database.



(a) $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (b) $N = 1000000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (c) $N = 1000000$, $N_{\text{fmax}} = 1000$, $N_{\text{unrev}} = 5$

Figure 6.1: Presence database sizes



(a) $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (b) $N = 1000000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$ (c) $N = 1000000$, $N_{\text{fmax}} = 1000$, $N_{\text{unrev}} = 5$

Figure 6.2: Lookup server bandwidth

6.3 Experimental Results

Implementation

Our MP3 library is implemented in 350 lines of C and 4000 lines of C++. The core cryptography relies on OpenSSL for AES, SHA-256, and elliptic curve arithmetic and signatures; RELIC [66] for pairing-friendly curves; and Percy++ [67] for PIR. The groups G_1, G_2, and G_T are defined by the Optimal Ate pairing over a 256-bit Barreto-Naehrig curve. We use a 224-bit Elliptic Curve, specifically secp224r1, for ECDSA, though the choice was arbitrary. AEAD is implemented using AES in Galois/Counter Mode (GCM). The PRFs and hash functions are implemented using SHA-256.

Evaluation

To evaluate the performance of MP3 vs. DP5, we simulated both protocols in a "worst-case" scenario with the number of users ranging from 1000 to 1000000 clients. To simulate the worst-case scenario, we had all clients perform registration and lookup for all epochs. All simulations were run on a machine with dual Intel Xeon E5-2630 v3 CPUs and 256GB of RAM. N_{lookup} was held fixed at 3 for all setups and both protocols. The equivalent of 1 year of execution were simulated in all setups. For both MP3 and DP5, the most expensive components are the lookup servers from both CPU and bandwidth perspectives.

Figure 6.1 compares the size of the presence databases for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. If we fix N_{rev} and N_{unrev} to small constants compared to N, it's obvious that the size of the long-term database of MP3 is significantly less than that of DP5. A smaller database implies cheaper lookup costs in the context of both bandwidth and CPU. Additionally, we can raise N_{rev} quite a bit and still maintain a smaller long-term database than that of DP5. It's also important to see how cheap the short-term database is relative to the long-term databases. Also note that the presence database sizes are proportional to the registration bandwidth.

Figure 6.3 compares the client-facing latency of the long-term epochs of MP3 and DP5 as well as the shared short-term epoch, for $N = 100000$ users. The latency of MP3's long-term epoch is smaller than that of DP5 due to

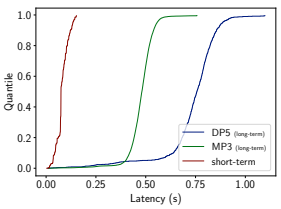


Figure 6.3: Client-facing lookup latency excluding RTT. $N = 100000$, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$

the inherently smaller database. The short-term epoch's latency is even less as the short-term databases are even smaller.

Figure 6.2 compares the bandwidth of a single long-term lookup server for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. The same pattern occurs that we saw in Figure 6.1 - for relatively constant N_{rev} and N_{unrev}, the bandwidth required is significantly less for MP3 than for DP5 and the bandwidth requirements of the short-term epoch are negligible compared to that of the long-term epochs.

6.4 Modifications to Dynamic Broadcast Encryption

Recall the operations of DBE from Section 6.2.3. Our modifications to DBE [13] only add the DBE.ShiftMK and DBE.ShiftDK operations. These operations are required for plausibly deniable revocations and suspensions. Recall that in a long-term database record for Alice in which Bob is actually revoked, Bob's old decryption key (dk_{ab}) is revoked and he is issued a new, but invalid, decryption key (dk'_{ab}). Without DBE.ShiftMK and DBE.ShiftDK, Bob could use dk'_{ab} to invert the Update operation and detect whether he was revoked or not.

The inverse update function is:

- DBE.Update$^{-1}(dk' = (x', A', B', \kappa'), x_r, B_r)$ - takes as input a decryption key and a revocation values and computes a new decryption key $dk := (x', A', B, \kappa')$ where $B := \frac{B_r}{B'^{(x'-x_r)}}$ that *can* decrypt ciphertexts *created before* the revocation.

Assuming DBE.ShiftMK and DBE.ShiftDK are not in place, a revoked user Bob can use DBE.Update^{-1} to *detect* that he has been revoked by Alice. Given two long-term presence records of Alice, where the former has not revoked Bob and the latter has revoked Bob, Bob can apply the Update^{-1} function to his new decryption key and compute a decryption key for the former presence record.

Let Bob's decryption keys for the former presence record be dk_{ab} and let Bob's decryption key for the latter presence record (after calling Update^{-1}) be dk'_{ab}. Also let C_1, C_2 be the ciphertext components from the former presence record. To detect if he has been revoked, all he must do is check if DBE.Decrypt$(dk_{ab}, C_1, C_2) \neq$

DBE.Decrypt(dk'_{ab}, C_1, C_2). If the statement is true, then Bob has been revoked by Alice.

By introducing DBE.ShiftMK and DBE.ShiftDK we create a one-way operation to the revocation process of MP3; thus Bob *cannot* invert the DBE.ShiftMK and DEB.ShiftDK functions without the knowledge of the plaintext of (C_1, C_2) and therefore cannot detect whether or not he was revoked.

6.5 Availability Against Malicious Parties

Some conventional approaches to ensure availability against malicious parties cannot be applied directly to privacy-preserving protocols, as they can leak information. This causes several challenges: keeping the databases small, ensuring the registration server stores all uploaded presence records, ensuring the lookup servers store all presence records and do not modify them.

A malicious client could upload many presence records during a given epoch, causing a denial of service (DoS) for all other clients. If a malicious client were using an anonymous channel, authentication would compromise the anonymity of that client, defeating the purpose of MP3. To eliminate this, k-times anonymous authentication schemes have been proposed [68, 69]. In these schemes, users are guaranteed anonymity up to k times; that is, if a user authenticates $k+1$ times, the identity of the user can be computed. Such private rate-limiting schemes can be used to limit the number of times a client registers during a given epoch without losing anonymity.

In the case that the registration server is dishonest and drops records, a user could "friend themself" to ensure that their presence records are being stored, by looking themself up during every epoch. Note that all presence records are indistinguishable, so the registration server can not target specific records for dropping.

Lastly, in the case that the lookup servers are dishonest and modify the database, Devet et al. propose a robust PIR scheme [70] that allows detection of malicious servers. This detection requires at least $t + 2$ honest servers, where t is the number of servers needed to collude to be able to determine the data in a query. This robust PIR scheme is implemented in MP3.

Discussion of Scalability and Cost Improvements

A primary bottleneck in DP5 is its lack of scaling with large number of users, specifically for long-term epochs. MP3 solves just that. The complexity for bandwidth usage of all operations are summarized in Table 6.1.

Table 6.1: Bandwidth complexities comparing MP3 and DP5 as a function of N, N_{fmax}, N_{rev}, and N_{unrev}.

		Client		Server	
		registration	lookup	registration	lookup
long-term	DP5	$\Theta(N_{\text{fmax}})$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}^{3/2}\right)$	$\Theta(N \cdot N_{\text{fmax}})$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$
	MP3	$\Theta(N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{1/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N \cdot N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$
short-term	DP5 MP3	$\Theta(1)$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N)$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$

The bulk of the cost in running a service such as MP3 or DP5 comes from the bandwidth usage of the given protocol. N_{rev} and N_{unrev} are always less than or equal to N_{fmax} by definition. In reality, with a 24-hour long-term epoch, setting N_{rev} and N_{unrev} to a small constant is very reasonable[4] ; therefore, MP3 is significantly cheaper during long-term epochs, and thus overcomes the scalability bottleneck of DP5.

In Figure 6.2a, with $N = 1000000$ users, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$, we can see that MP3 uses about half the bandwidth of that of DP5 and it's evident that the savings grow as the number of users increases.

[4] Social networks such as Twitter disallow bulk unfollowing [71], making our argument about setting N_{rev} and N_{unrev} to a small/constant value even stronger.

Chapter 7

Future Work and Final Remarks

In this dissertation I detailed strong security properties that are necessary for private group conversation protocols and applications. I detailed how these properties apply in two real-world networking models. I examined the most popular secure communication protocol (Signal) and demonstrated how it does not provide the not only these stronger privacy properties but even does not provide confidentiality for all messages.

Given both networking models of online instant messaging and mobile message; I provide two private group messaging protocols that provably provide the necessary security properties. I implement these protocols and show their performance is practical under real-world networking conditions. Finally, I detail an improved private presence protocol to complement the messaging protocols that reduces the message size to logarithmic via Dynamic Broadcast Encryption$\tilde{(}$DBE).

This is an open research area and future directions for this work include:

Manually proving the security properties of protocols is a painstakingly tedious process that leads to hard to read and reason about proofs. This dissertation is the first for formally examine many of the security properties of private messaging protocols. One future direction would explore how automated provers and proof assistants can increase the confidence and improve the writing and verification of these properties.

The two proposed secure communication protocols only support text messages. However, many popular messaging applications have extensive features to support other interactions, for example sending images, reactions, or external hyper links. As shown in the analysis of Signal, these features have bespoke consequences on the privacy of the conversation and cannot be implemented without due analysis. Another direction for future research would explore the privacy implications of these auxiliary features, the consequence they have on the privacy of the conversation, and how to provide these features without sacrificing the strong privacy guarantees.

This dissertation has focused on private communication between a group of participants that can mutually authenticate one another. However, many conversations happen between people that do not for a clique. It is worth exploring what privacy means for these types of communication and how it may be guaranteed with

provable protocols. Consider a forum or industry IRC channel. The number of participants may be low but they may not be able to mutually authenticate. Can a referral mechanism be introduced to Mobile CoWPI? Would Mobile CoWPI still provide participant consistency? Also consider, How can private communication be provided for mailing lists? Can we achieve deniable message authentication in a broadcast channel?

References

[1] Nicholas Weaver. A close look at the NSA's most powerful internet attack tool. http://www.wired.com/2014/03/quantum/. Accessed: 19 May 2017.

[2] Apple, 2017.

[3] Nadim Kobeissi, 2017.

[4] Andreas Straub. Omemo encryption.

[5] Open Whisper Systems. *Open Whisper Systems*. https://whispersystems.org/.

[6] WhatsApp, 2017.

[7] Moxie Marlinspike. Facebook messenger deploys signal protocol for end to end encryption, 2016.

[8] Moxie Marlinspike. Open whisper systems partners with google on end-to-end encryption for allo, 2016.

[9] Moxie Marlinspike. Signal and GIPHY, jan 2016.

[10] Inc. Amazon Web Services. Amazon web service (aws) - cloud compute services, 2019.

[11] linode. *linode*. https://linode.com/.

[12] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, 2015.

[13] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. pages 39–59, 2007.

[14] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: Secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 232–249. IEEE, 2015.

[15] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84. ACM, 2004.

[16] Ian Goldberg, Berkant Ustaoğlu, Matthew D Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 358–368. ACM, 2009.

[17] Hong Liu, Eugene Y Vasserman, and Nicholas Hopper. Improved group off-the-record messaging. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 249–254. ACM, 2013.

[18] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, Jörg Schwenk, and Thorsten Holz. How secure is textsecure? In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 457–472. IEEE, 2016.

[19] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[20] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[21] Michael Schliep, Ian Kariniemi, and Nicholas Hopper. Is bob sending mixed signals? In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 31–40. ACM, 2017.

[22] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: On the end-to-end security of group chats in signal, whatsapp, and threema. 2018.

[23] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol, 2016.

[24] eQualit.ie. *(n+1)sec protocol specification - draft*. https://equalit.ie/introducing-n1sec-a-protocol-for-distributed-multiparty-chat-encryption/.

[25] Michel Abdalla, Céline Chevalier, Mark Manulis, and David Pointcheval. Flexible group key exchange with on-demand computation of subgroup keys. *Africacrypt*, 10:351–368, 2010.

[26] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1802–1819. ACM, 2018.

[27] Emad Omara, Benjamin Beurdouche, Eric Rescorla, Srinivas Inguva, Albert Kwon, and Alan Duric. The Messaging Layer Security (MLS) Architecture. Internet-Draft draft-ietf-mls-architecture-02, Internet Engineering Task Force, March 2019. Work in Progress.

[28] Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-06, Internet Engineering Task Force, May 2019. Work in Progress.

[29] Dominic Rushe. Lavabit founder refused FBI order to hand over email encryption keys. The Guardian, October 2013.

[30] Moxie Marlinspike and Trevor Perrin. The double ratchet algorithm, 2016.

[31] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[32] Trevor Perrin. [messaging] how secure is textsecure?, Nov 2014.

[33] Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[34] Paul Rösler, Christian Mainka, and Jörg Schwenk. More is less: How group chats weaken the security of instant messengers signal, whatsapp, and threema. 2017.

[35] David Cole. We kill people based on metadata, 2014.

[36] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Technologies*, 2015(2):46–64, 2015.

[37] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 57–69. ACM, 2015.

[38] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system. In *Advances in cryptology EUROCRYPT'94*, pages 275–286. Springer, 1994.

[39] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, September 2008.

[40] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In *Provable Security*, pages 1–16. Springer, 2007.

[41] Chris Alexander and Ian Goldberg. Improved user authentication in off-the-record messaging. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 41–47. ACM, 2007.

[42] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 400–409. ACM, 2006.

[43] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[44] *OpenStack IRC meetings*. http://eavesdrop.openstack.org/.

[45] twitter. *twitter*. https://twitter.com/.

[46] reddit. *reddit*. https://reddit.com/.

[47] Facebook. *Facebook*. https://facebook.com/.

[48] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 453–474, London, UK, UK, 2001. Springer-Verlag.

[49] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. The cognitive walkthrough method: A practitioner's guide. In Jakob Nielsen and Robert L. Mack, editors, *Usability Inspection Methods*, pages 105–140. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[50] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Usenix Security*, volume 1999, 1999.

[51] Sumeet Gujrati and Eugene Y Vasserman. The usability of truecrypt, or how i learned to stop whining and fix an interface. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 83–94. ACM, 2013.

[52] Android — the world's most populare mobile platform.

[53] Subhash Sankuratripati, Moti Yung, Anirudh Grag, and Wentao Huang. Catch me if you can: An account based end-to-end encryption for 1/1 snaps. In *Real World Crypto Symposium*. IACR, 2019.

[54] Phillip Rogaway. Nonce-based symmetric encryption. In *International Workshop on Fast Software Encryption*, pages 348–358. Springer, 2004.

[55] Motorola Mobility LLC. Moto g3 - motorola, 2019.

[56] The PostgreSQL Global Development Group. Postgresql: The world's most advanced open source database, 2019.

[57] Fabien Laguillaumie and Damien Vergnaud. Multi-designated verifiers signatures. In *International Conference on Information and Communications Security*, pages 495–507. Springer, 2004.

[58] B Chor, O Goldreich, E Kushilevitz, and M Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, 1995.

[59] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

[60] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[61] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[62] Sebastian Angel and Srinath TV Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI*, pages 551–569, 2016.

[63] ANSI ANSI. X9. 62-1998: Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ecdsa). *American National Standards Institute (ANSI), Washington, DC*, 1998.

[64] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[65] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Annual International Cryptology Conference*, pages 41–55. Springer, 2004.

[66] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[67] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ project on sourceforge. http://percy.sourceforge.net, 2014.

[68] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

[69] Patrick P Tsang, Man Ho Au, Apu Kapadia, and Sean W Smith. Blacklistable anonymous credentials: blocking misbehaving users without TTPs. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 72–81. ACM, 2007.

[70] Casey Devet, Ian Goldberg, and Nadia Heninger. Optimally robust private information retrieval. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 269–283, 2012.

[71] Suspension - what's the daily/hourly unfollow limit for each user? what is the aggressive behaviour? https://twittercommunity.com/t/suspension-whats-the-daily-hourly-unfollow-limit-for-each-user-what-is-the-aggressive-behaviour/13971.