End-to-End Secure Mobile Group Messaging with Conversation Integrity and Deniability

Michael Schliep
schli116@umn.edu
University of Minnesota

Nicholas Hopper
hoppernj@umn.edu
University of Minnesota

ABSTRACT

In this paper, we describe Mobile CoWPI, a deployable, end-to-end secure mobile group messaging application with proofs of security. Mobile CoWPI allows dynamic groups of users to participate in, join, and leave private, authenticated conversations without requiring the participants to be simultaneously online or maintain reliable network connectivity. We identify the limitations of mobile messaging and how they affect conversational integrity and deniability. We define strong models of these security properties, prove that Mobile CoWPI satisfies these properties, and argue that no protocol that satisfies these properties can be more scalable than Mobile CoWPI. We also describe an implementation of Mobile CoWPI and show through experiments that it is suitable for use in real-world messaging conditions.

CCS CONCEPTS

• **Security and privacy** → **Security protocols**.

KEYWORDS

end-to-end encryption, secure messaging

ACM Reference Format:
Michael Schliep and Nicholas Hopper. 2019. End-to-End Secure Mobile Group Messaging with Conversation Integrity and Deniability. In *18th Workshop on Privacy in the Electronic Society (WPES'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3338498.3358644

1 INTRODUCTION

Texting and social media-based messaging applications have become nearly as common as face-to-face communications for conversation between individuals and groups. The popularity of these messaging applications stems in part from their convenience, allowing users to communicate even in a mobile and asynchronous setting, where their network availability may be unreliable and they may come online and go offline at different times. In response to increasing privacy concerns, some of the most widely deployed messaging applications, including WhatsApp [29], Google Allo [19], Facebook [18], and Signal [27], have been deploying end-to-end encryption to protect the confidentiality and integrity of messages in users' conversations.

However, message confidentiality and integrity are not sufficient to protect a conversation. While current applications protect the integrity of individual *messages* — an adversary cannot modify a message while in transit from Alice to Bob — they do not protect the integrity of the *conversation.* Consider the following conversation between Alice and Bob, in which the order that messages are displayed can drastically affect the meaning of the conversation, even if the individual messages cannot be modified:

Alice's View:

Alice: Are you going to the protests?
Alice: Have you had lunch yet?
Bob: No... Yes.

Bob's View:

Alice: Have you had lunch yet?
Alice: Are you going to the protests?
Bob: No... Yes.

We refer to the security property that a conversation must be displayed consistently to all participants as *conversation integrity.* This is an example of an additional security property we deem necessary for any future protocols to achieve end-to-end secure messaging.

Another property provided by some end-to-end encryption protocols is deniability. Consider the following conversation:

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

Message deniability guarantees there is no cryptographic proof to a third party that the whistleblower authored the message. Now consider the following conversation:

Whistleblower: My SSN is 123-45-6789.
Reporter: What is your company doing illegally?
Whistleblower: They are dumping poison into the water.

A protocol that provides message deniability allows the whistleblower to argue that they did not author the messages. But only the whistleblower knows their social security number so a protocol must also provide message unlinkability, guaranteeing there is no cryptographic proof to a third party that both messages were authored by the same participant.

Finally, most deployed secure messaging applications are based on the Signal two-party protocol, which is non-trivial to extend to group settings. Recently, multiple vulnerabilities [23, 25] have been discovered in the way these applications implement end-to-end secure messaging for groups. These vulnerabilities allow an adversary to drop or reorder messages in two-party and group conversations. Other messaging applications ignore end-to-end security of group conversations entirely. We consider group conversations just as important as two-party conversations and future deployable protocols must be designed with that in mind.

On the other hand, secure messaging protocols appearing in the research literature [6, 8, 10, 16, 26] make assumptions that do not fit the modern mobile internet which makes them unrealistic for practical mobile deployments. Most of these works require synchronous communication, and provide little to no guarantees about conversation integrity. Moreover, many of these protocols provide deniability but not unlinkability.

Another related effort is the recent IETF Working Group on Message Layer Security (MLS) [4, 21]. MLS is focused on improving the scalability of end-to-end encryption to support thousands of users while explicitly not supporting conversation integrity or deniability. While this is one possible tradeoff, we argue that it is equally useful to support stronger security properties for smaller groups of people who can mutually authenticate each other.

In this paper we address the problem of designing a deployable, end-to-end secure mobile group messaging application. Our contributions include:

- 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 2).
- We describe a relatively simple and provably secure protocol for Mobile Conversations With Privacy and Integrity (Mobile CoWPI) in Section 9. We show in Section 4 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 6). We argue that under these restrictions, Mobile CoWPI is within a constant factor of optimal in terms of message size.
- We implement Mobile CoWPI as a Java server and library and show that it performs well in a realistic internet environment (Section 5) deployed on Amazon AWS[3] and Linode [15] with both desktop and Android [1] clients.

2 BACKGROUND

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.

2.1 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.

2.2 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) 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 6.

2.3 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

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

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.

2.4 Threat Model

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. This strong threat model is consistent with modern secure messaging threat models.

2.5 Security Properties

Besides the system goals of offline users and message receipts we now informally discuss the security goals of secure mobile messaging. Unger et. al. [28] provide a comparison of security goals of different secure messaging applications. We relate our security goals to the goals of their work where appropriate. In Section 4 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 Appendix A.

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 confidentiality but introduces the concept of key healing. 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. 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. 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. 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 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. 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.

3 DESIGN

3.1 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 3.7. Conversation messages contain a message to be displayed to the participants of the session, detailed in Section 3.9. Participant Update message allow adding and removing participants from a conversation, detailed in Section 3.10. Finally, Receipt messages indicate a participant has received, accepted, and processed all prior messages, detailed in Section 3.8.

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 3.5 and $auth_{as_*}$, $auth_{bs_*}$ are pairwise authentication blocks between Alice or Bob and each OES provider detailed in Section 3.6. 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 [24].

3.2 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 4.

3.3 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:

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

Where H and KDF_1 are used for two-party NAXOS [14] 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 ($IND\$ - CPA$) [22] and integrity of ciphertext security ($INT - CTXT$) [5]. We choose AES-GCM with random IVs for our AEAD scheme.

3.4 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.

3.5 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:

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

Then Alice computes a symmetric key:

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

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:

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

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.

3.6 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.

3.7 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 $i \in P \backslash \{Alice\}$ as described in Section 3.5, 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 3.6 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 3.6 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.

3.8 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 $i \in P \backslash \{Alice\}$. 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 3.5 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 3.6 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.

3.9 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 $i \in P \backslash \{Alice\}$ as detailed in Section 3.5 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 3.6 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.

3.10 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 paper. 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 $i \in (P \cup P') \backslash \{Alice\}$ as described in Section 3.5 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 3.6 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.

3.11 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 2. 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 1 Client To Provider Channel Setup

```
function C2SChannelSetup($S, S, lpk_s, lsk_c$)
    $esk_c[0] \leftarrow_R Z_p^*, epk_c[0] \leftarrow g^{H(esk_c[0], lsk_c)}$
    Send($S, C, epk_c[0]$)
    $epk_s[0], c_1 \leftarrow$ Recv($S$)
    $km_1 \leftarrow lpk_s^{H(esk_c[0], lsk_c)}$
    $km_2 \leftarrow epk_s[0]^{lsk_c}$
    $km_3 \leftarrow epk_s[0]^{H(esk_c[0], lsk_c)}$
    $k_1 \leftarrow KDF_3(km_1, km_2, km_3, S, C)$
    $t, epk_s[1] \leftarrow Dec_{k_1}(c_1)$
    $esk_c[1] \leftarrow_R Z_p^*, epk_c[1] \leftarrow g^{H(esk_c[1], lsk_c)}$
    $km_4 \leftarrow epk_s[1]^{lsk_c}$
    $km_5 \leftarrow lpk_s^{H(esk_c[0], lsk_c)}$
    $km_6 \leftarrow epk_s[1]^{H(esk_c[0], lsk_c)}$
    $k_2 \leftarrow KDF_3(km_4, km_5, km_6, C, S)$
    $c_0 = Enc_{k_2}(t, epk_c[1])$
    Send($S, c_0$)
    return $esk_c, epk_s$
```

Algorithm 1 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 2 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 3 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 4 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 and line 5 decrypts the message and the senders next ephemeral public key.

Algorithm 2 Provider To Client Channel Setup

```
function S2CChannelSetup($S, C, lsk_s$)
    $C, epk_c[0] \leftarrow$ Recv($C$)
    $lpk_c \leftarrow$LookupUser($C$)
    $esk_s[0] \leftarrow 0, 1^l, epk_s[0] \leftarrow g^{H(esk_s[0], lsk_s)}$
    $esk_s[1] \leftarrow Z_p^*, epk_s[1] \leftarrow g^{esk_s[1]}$
    $km_1 \leftarrow epk_c[0]^{lsk_s}$
    $km_2 \leftarrow lpk_c^{H(esk_s[0], lsk_s)}$
    $km_3 \leftarrow epk_c[0]^{H(esk_s[0], lsk_s)}$
    $k_1 \leftarrow KDF_3(km_1, km_2, km_3, S, C)$
    $t_s \leftarrow_R \{0, 1\}^l$
    $c_1 \leftarrow Enc_{k_1}(t_s, epk_{s[1]})$
    Send($C, epk_s[0], c_1$)
    $c_2 \leftarrow$ Recv($C$)
    $km_4 \leftarrow lpk_c^{H(esk_s[1], lsk_s)}$
    $km_5 \leftarrow epk_c[0]^{lsk_s}$
    $km_6 \leftarrow epk_c[0]^{H(esk_s[1], lsk_s)}$
    $k_3 \leftarrow KDF_3(km_4, km_5, km_6, C, S)$
    $t_c, epk_c[1] \leftarrow Dec_{k_2}(c_1)$
    if $t_s = t_c$ then
        return ($C, esk_s, epk_c$)
    else
        return $\perp$
```

Algorithm 3 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 4 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 ($esk_r, epk_s$)
```

3.12 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 [2].

4 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 2. We provide full proofs in Appendix A. 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.

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.5 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.

4.6 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.

4.7 Deniability

Recall deniability as discussed in Section 2. 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.

4.7.1 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 [10], 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 A, 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.

5 EVALUATION

We implemented Mobile CoWPI as a Java server and client library[2]. 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 [3] free tier t2.micro EC2 instance in Ohio and a second OES hosted on a $5/month Linode [15] 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 ≈ 30ms and between the router and Linode OES is ≈ 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 ≈ 50ms latency for messaging.

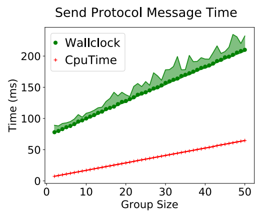


Figure 1: The wallclock (25th, 50th, and 90th percentile) and CPU time to send a protocol message.

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

Figure 1 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 2 shows the outgoing message size in bytes when sending a message. 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 6. Pairwise ciphertext blocks allow for very little overhead to receive a message. Conversation and receipt messages are $O(s)$ while setup and participant update messages 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.

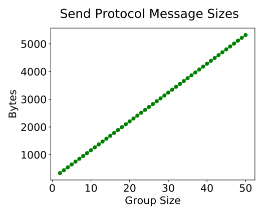


Figure 2: Outgoing message size in bytes.

5.1 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 [17] Android [1] phone.

Figure 3a shows the time in milliseconds to create a protocol message on the Android device. Figure 3b 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 3c shows the storage space required per conversation of each size for both the router and OES. Our implementation uses a PostgreSQL [11] database which causes the steps seen in Figure 3c.

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 ≈ 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.

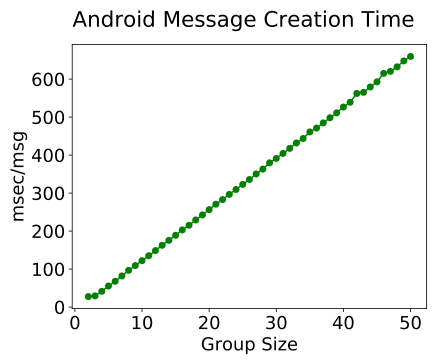
6 DISCUSSION

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

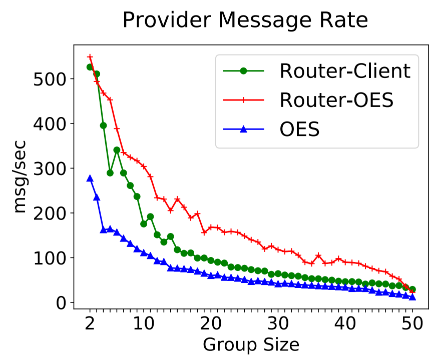
6.1 Limitations of Group Key Agreements

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

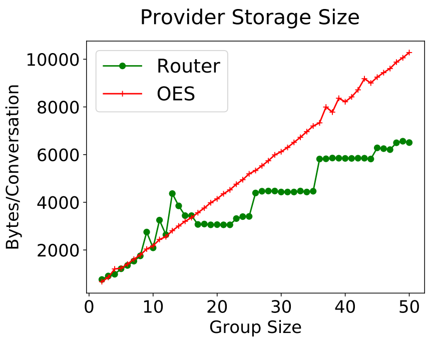
[2]https://github.com/mschliep/cowpi



(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.

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 [13], 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 [24] are linear in the number of receivers and have been deployed to support billions of messages per day.

6.2 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.

6.3 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.

7 RELATED WORK

Off-The-Record (OTR) [6] 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) [10] 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 distinguisher 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 non-mobile model.

Group Off-The-Record (GOTR)[16] 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 synchronous model as all users must be online to execute the GKA and consistency checks, making it not suitable for mobile communication.

SYM-GOTR [26] is a recent proposal for synchronous end-to-end secure group conversations with the same properties as our work. SYM-GOTR works with existing XMPP servers and a client plugin. Similar to GOTR, participants first setup pairwise secure channels between all participants. Then the participants share symmetric key inputs and verification keys. When Alice sends a message she first computes a symmetric encryption key by hashing all of the symmetric key input material from all the other participants and encrypts the message. She broadcast the ciphertext to all participants. After receiving a ciphertext all participants perform a two phase consistency check of the ciphertext over the pairwise secure channels. The first phase verifies all users have received the ciphertext and the second phase identifies any users who have misbehaved. Modifying the participants of the conversation is as simple as distributing new symmetric key inputs and verification keys. The main limitations of SYM-GOTR is that it requires all participants to be online at the same time and the two phase interactive consistency check causes additional delay in message processing.

Signal [27] (formerly TextSecure) is the most widely deployed protocol for secure mobile messaging. However it has only recently received formal analysis of its security properties [7, 9, 12]. With [23, 25] 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 [20] 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.

Asynchronous Ratcheting Trees (ART) [8] 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 like Mobile CoWPI. 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) [4, 21]. The focus of the working group has been on improving scalabity to thousands of users with limited security trade offs. The two major trade offs compared to Mobile CoWPI is the lack of conversation integrity and deniability of MLS. These security properties are currently considered OPEN ISSUES by the working group.

8 CONCLUSION

In this work we addressed the problem of practical end-to-end secure mobile messaging with support for group conversations. We identified a mobile messaging model and showed that (1) multiple service providers are required to provide strong conversation integrity and (2) to provide deniable message sender authentication, messages must be $O(n)$ in size. We then showed that given an any-trust model, a relatively simple protocol, Mobile CoWPI, can achieve these strong security properties while being practically efficient. We provide proofs of the security of Mobile CoWPI, and analyze the performance of a Java implementation with groups of varying size to show the protocol performs well with realistic internet latencies.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. This work was supported by the NSF under grant 1814753.

REFERENCES

[1] [n.d.]. Android | The World's Most Populare Mobile Platform. https://www.android.com/

[2] Chris Alexander and Ian Goldberg. 2007. Improved user authentication in off-the-record messaging. In *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 41–47.

[3] Inc. Amazon Web Services. 2019. Amazon Web Service (AWS) - Cloud Compute Services. https://aws.amazon.com/

[4] Richard Barnes, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. 2019. *The Messaging Layer Security (MLS) Protocol*. Internet-Draft draft-ietf-mls-protocol-06. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-mls-protocol-06 Work in Progress.

[5] Mihir Bellare and Chanathip Namprempre. 2008. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *J. Cryptol.* 21, 4 (Sept. 2008), 469–491. https://doi.org/10.1007/s00145-008-9026-x

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

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

[8] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. 2018. 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*. ACM, 1802–1819.

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

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

[11] The PostgreSQL Global Development Group. 2019. PostgreSQL: The world's most advanced open source database. https://www.postgresql.org/

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

[13] Fabien Laguillaumie and Damien Vergnaud. 2004. Multi-designated verifiers signatures. In *International Conference on Information and Communications Security*. Springer, 495–507.

[14] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. 2007. Stronger security of authenticated key exchange. In *Provable Security*. Springer, 1–16.

[15] linode. [n.d.]. *linode*. https://linode.com/.

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

[17] Motorola Mobility LLC. 2019. Moto G3 - Motorola. https://www.motorola.com/us/products/moto-g-gen-3

[18] Moxie Marlinspike. 2016. Facebook Messenger deploys Signal Protocol for end to end encryption. https://whispersystems.org/blog/facebook-messenger/

[19] Moxie Marlinspike. 2016. Open Whisper Systems partners with Google on end-to-end encryption for Allo. https://whispersystems.org/blog/allo/

[20] Moxie Marlinspike and Trevor Perrin. 2016. The X3DH Key Agreement Protocol. https://whispersystems.org/docs/specifications/x3dh/

[21] Emad Omara, Benjamin Beurdouche, Eric Rescorla, Srinivas Inguva, Albert Kwon, and Alan Duric. 2019. *The Messaging Layer Security (MLS) Architecture*. Internet-Draft draft-ietf-mls-architecture-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/draft-ietf-mls-architecture-02 Work in Progress.

[22] Phillip Rogaway. 2004. Nonce-based symmetric encryption. In *International Workshop on Fast Software Encryption*. Springer, 348–358.

[23] Paul Rösler, Christian Mainka, and Jörg Schwenk. 2018. More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema. (2018).

[24] Subhash Sankuratripati, Moti Yung, Anirudh Grag, and Wentao Huang. 2019. Catch Me if You Can: An Account Based End-to-end Encryption for 1/1 Snaps. In *Real World Crypto Symposium*. IACR.

[25] Michael Schliep, Ian Kariniemi, and Nicholas Hopper. 2017. Is Bob Sending Mixed Signals?. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. ACM, 31–40.

[26] Michael Schliep, Eugene Vasserman, and Nicholas Hopper. 2018. Consistent Synchronous Group Off-The-Record Messaging with SYM-GOTR. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018), 181–202.

[27] Open Whisper Systems. [n.d.]. *Open Whisper Systems*. https://whispersystems.org/.

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

Figure 4: 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 $(d = b)$
```

Figure 5: 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 $win$
```

[29] WhatsApp. 2017. https://www.whatsapp.com/security

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

A.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 4 and provides integrity of ciphertexts as defined in Figure 5. 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.

Figure 6: 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 $(d = b)$
```

We assume the NAXOS protocol is a secure authenticated key agreement protocol. Figure 6 describes the game used by the original authors [14], 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

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

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 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 $Adv^{NAXOS}(M) = Pr[M \text{ wins}] - \frac{1}{2}$.

Figure 7: 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 $(d = b)$
```

A.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) messages in a session. The adversary is only limited to avoid trivially breaking message confidentially. Message confidentiality is captured by the game in Figure 7.

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 $Adv^{conf}(M) = Pr[M\text{wins}] - \frac{1}{2}$.

Theorem A.1. *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) $epk_{ab} \leftarrow$ 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, $k_{ab} \leftarrow$ 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, $k_{ab} \leftarrow$ 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) $c_{ab} \leftarrow id_{ba}$, 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.

Figure 8: 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.
```

- 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) $c \leftarrow$ 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. □

A.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 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 8 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 $Adv^{auth}(M) = Pr[M\text{wins}]$.

THEOREM A.2. *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

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

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, $k_{ab} \leftarrow$ 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, $k_{ab} \leftarrow$ 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 $Adv^{INT-CTXT}(M_1) = S_1 \cdot \frac{1}{(w-1)qy}$.
□

Figure 9: 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 $pm_a \neq pm_b$.
```

A.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 9 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 4 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.

Theorem A.3. *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 KDF₂ 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.

Figure 10: Deniability Game G_0

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

(5) When computing the NAXOS key of the OES authentication block of the of a received protocol message that was not sent, $k_{ab} \leftarrow$ 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) $c_{ab} \leftarrow id_{ba}$, 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 $Adv^{INT-CTXT}(M_1) = S_1 \cdot \frac{1}{y}$. □

A.5 Deniability

We capture the deniability property with the general-purpose game detailed in Figure 10. 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 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:

$$input_d = \{(lsk_0, epk_0) \dots, (lsk_n, epk_n)\}$$
$$input_s = \{(lpk_0, epk_0), \dots, (lpk_n, epk_n)\}, (lsk_a, esk_a)$$
$$output_b = \{esk_{a0}, esk_{aw}\}$$

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.

Theorem A.4. *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

$$Adv^{DENY-PART}(M) \leq y \cdot Adv^{NAXOS}(M_0) \tag{1}$$

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

Proof. Recall the Mobile CoWPI simulator discussed in Section 4. 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) $epk_{bc} \leftarrow \text{Send}(B, C)$
(2) Send epk_{bc} to C.
(3) Upon C receiving epk_{bc}, $epk_{cb} \leftarrow \text{Send}(C, C, epk_{bc})$, epk_{cb}.
(4) When computing a NAXOS key of all ciphertext blocks after the ith $k_{bc} \leftarrow \text{Reveal}(B, C, epk_{bc}, epkcb)$ is used as the key.
(5) When computing the NAXOS key of the ith ciphertext block , $k_{bc} \leftarrow \text{Test}(B, C, epk_{bc}, epk_{cb})$.

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. □

A.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.

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

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.