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

Michael Schliep
University of Minnesota
schli116@umn.edu

Nicholas Hopper
University of Minnesota
hoppernj@umn.edu

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.

I. 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 [24], Google Allo [15], Facebook [14], and Signal [21], 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 example of such a property we focus on in this work 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 [18], [19] 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 [3], [8], [13], [20], [5] make assumptions that are not realistic in the modern mobile internet which makes them unrealistic for practical deployments. Most of these works require synchronous communication or provide little to no guarantees about conversation integrity.

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 II).
- We describe a relatively simple and provably secure protocol for Mobile Conversations With Privacy and Integrity (Mobile CoWPI) in Section III. We show in Section IV 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 VI). 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 V).

II. Background

In this section we layout 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 all of the security properties we provide in our protocol along with the threat model used for each property.

A. 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 and send messages 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. This single service provider can break the conversation integrity property of a protocol that allows a conversation to progress while some participants are offline. The service provider simply needs to fork the conversation 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.

To avoid this conversation integrity attack the system model of Mobile CoWPI consists of a routing/caching service provider with multiple mirror service providers. Users register with the service providers out-of-band. The users register long-term and single use pre-keys with the providers. The single use pre-keys allow for conversation setup while some users are offline. When sending a message in a conversation the user uploads it to the routing service provider. The routing provider sends the message to the mirror providers. After that, the routing and mirror providers send the message to the participants. The participants wait until they have received the message from the routing provider and all mirror providers before processing it and only process messages if it has been received in the same order from all providers. The routing provider is asked to send protocol messages in the same order to all mirrors and all providers are asked to send all messages to the participants in the same order. The protocol enforce that messages are handled in an order that preserves the integrity of the conversation. It is only important that at least one provider send all messages in the same order to all participants. Since this is an any trust model we believe a single routing provider and a single mirror provider is sufficient to provide practical conversation integrity. We discuss some limitations of this model in Section VI.

B. Service Availability

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

Denial of Service protection is also a non-goal of Mobile CoWPI. It is trivial for a service provider to deny service to a client by not forwarding messages to the victim. It is possible for a malicious provide 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. We discuss this in more detail in Section VI. 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.

C. Security Properties

Besides the system goals of offline users and message receipts we now discuss the security goals of secure mobile messaging. Unger et. al. [23] 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 IV we provide sketches of the security proofs for these properties and provide the full proofs in Appendix A.

Message Confidentiality: Only conversation participants can read a message.

Message Integrity: Messages are guaranteed to not have been modified in transit.

Message Authentication: Conversation participants can verify the author of a message. Message authentication implies message integrity.

Forward Secrecy: Past messages are confidential even if future key material is revealed.

Backward Secrecy: Future messages are confidential if past key material is revealed, also know as future secrecy or post-compromise secrecy.

Participant Authentication: Participants can verify other participants are really who they claim to be.

Participant Consistency: All participants of a conversation agree on all the participants of the conversation.

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

Deniability: Participants must be able to 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. Participant repudiation implies message 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.

Anonymity Preserving: The protocol should not undermine the anonymity features of the underlying transport.

Computation and Trust Equality: All users perform similar computations and no user is trusted more than any other.

Untrusted Service Provider: Any individual service provider is not trusted to provide any of the security properties.

Dynamic Groups: Participants can be added and removed from conversations without restarting the protocol.

D. 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, and reorder the network traffic. Each security property is provided under the strongest adversary that cannot trivially break the property. We now detail the exact capabilities of the adversary for each security property.

Message Confidentiality: The adversary has full network control and can insert, drop, and reorder network messages. The adversary is allowed to compromise any or all of the service providers. The adversary may compromise any participant in a non-target session. The adversary is not allowed to compromise any participant in the target session as that would trivially allow them to trivially reveal the plaintext of the message.

Message Integrity and Authentication: The adversary considered for message integrity and authentication is the same as for message confidentiality.

Forward and Backward Secrecy: The adversary for forward and backward secrecy is similar to the message confidentiality adversary. This adversary is also allowed to compromise any or all participant in the target conversation but is not allowed to impersonate a participant in the target conversation. The adversary is also not allowed to compromise a participant at the time a target message is being processed as this would trivially break the secrecy.

Participant Authentication: The adversary is allowed to compromise the service providers and has full network control. The adversary is allowed to compromise non-target participants. We assume there is an authenticated side-channel between the target participants. This may be pre-shared secrets or in person face-to-face communication.

Participant Consistency: The adversary has full network control and is allowed to compromise all of the service providers. The adversary is allowed to compromise any participant in any session. The adversary is not allowed to compromise all of the service providers and a participant in the target session. This would allow the adversary to forge and deliver inconsistent participant changes to participants.

Conversation Integrity: The adversary has full network control and is allowed to compromise any participant in any session. The adversary is allowed to compromise all but one of the service providers. If the adversary compromised all service providers, the adversary could then fork the conversation as discussed earlier.

Anonymity Preserving: The adversary is a passive network adversary such as an Internet Service Provider. The adversary is not allowed to compromise any of the service providers or users of the messaging service. The goal of the adversary is to link a session to the long-term key or identity of a participant. The goal of Mobile CoWPI is to not reduce the anonymity provided by the underlying transport, e.g. Tor [22].

Deniability: We model deniability similar to Di Raimondo et. al. [6]. A protocol is said to be deniable if a distinguisher can not distinguish between a real protocol transcript and a simulated transcript. We only consider the protocol deniable if any user can produce a simulated transcript that is indistinguishable from a real transcript. The simulator must only take as input information that is known to a user, e.g. identities, long-term public keys, the users secret keys. The distinguisher is given access to the long-term private information of users, i.e. long-term secret keys.

When considering message unlinkability the distinguisher and simulator are also given a part of a real transcript and ephemeral state relating to a single message. The simulated transcript must contain this partial information and still be indistinguishable from the real transcript.

III. DESIGN

A. 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. The routing provider shares this registration information with the mirror service providers. When messages are sent as part of a conversation they are uploaded to the routing provider and distributed to the mirror providers. All service providers then send the messages to the participants. The participants do not process a message until it has been received from the routing service provider and all mirror providers in the same order. As long as a single provider is honest conversation integrity and participant consistency are enforced.

Users of Mobile CoWPI communicate with each provider using a two-party secure channel. The two party channels provide all of the security properties described in the previous section. We describe this channel in Section III-K.

When Alice wishes to set up a new conversation with Bob and Charlie she first fetches Bob's and Charlie's long-term public keys and a single use public pre-key for each of Bob and Charlie. She then uploads a *setup* message of the form:

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

to the routing provider. Sid is a unique session identifier for the session. P is the set of participants. c_{a*} are per-user ciphertext blocks that authenticate the message and contain future key material for Bob and Charlie, described in Section III-E. The $auth_{a*}$ blocks are authentication blocks for the providers detailed in Section III-F. The providers then distribute the following to Alice, Bob and Charlie respectively.

$$Sid, \text{“}SETUP\text{”}, Alice, P$$
$$Sid, \text{“}SETUP\text{”}, Alice, P, c_{ab}$$
$$Sid, \text{“}SETUP\text{”}, Alice, P, c_{ac}$$

Alice, Bob, and Charlie do not process a message until they have received it from every provider. All protocol messages have a similar form, with a data block followed by per-user and per-provider blocks. This allows sending only the data block and a single per-user block to the participants.

After receiving the setup message, Charlie sends a *receipt* acknowledging Charlie has accepted the setup message from Alice. He first fetches the long-term public key and a single ephemeral public pre-key for Bob; he already has the necessary key material for Alice from the setup message. Charlie then uploads a protocol message of the form:

$$Sid, \text{“}RCPT\text{”}, Charlie, c_{ca}, c_{cb}, auth_{cs_1}, \ldots, auth_{cs_m}$$

where c_{c*} and $auth_{c*}$ authenticates both the setup message with Alice, Bob, and the providers respectively.

For Bob to send a *broadcast* message m, Bob uploads a protocol message of the form:

$$Sid, \text{“}MSG\text{”}, Bob, c, c_{ba}, c_{bc}, auth_{bs_1}, \ldots, auth_{bs_m}$$

where c is an authenticated encryption with associated data (AEAD) ciphertext of the message m under a random key. c_{ba} and c_{bc} are AEAD encryptions of random key material that authenticate for Alice and Charlie respectively, and $auth_{b*}$ authenticates the message for the providers. The term broadcast is used to indicate this is a group conversation message that is meant to be displayed to the user, it has no relation to broadcast messaging.

The final message type is *group update* for changing the participants of the conversation. When Alice wishes to add Dave to the conversation she fetches Dave's long-term and ephemeral key material and uploads a message of the form:

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

where P is the new set of participants and c_{a*} and $auth_{a*}$ authenticates the message for every participant and the providers. The rules of who is authorized to make participant changes as well as what kind of changes they are allowed to make are left up to the implementation but must be enforceable by the providers.

B. Message Order

We now detail the rules enforcing message order.

1) A protocol message must be received from all providers before processing. The protocol messages must also be processed in the order they are received.
2) All conversations start with a setup message.
3) When Alice sends a receipt, it must acknowledge all setup, broadcast, and group 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 broadcast or group update, Alice must have sent a receipt.
5) When Alice sends a receipt, she acknowledges messages with every participant and the providers 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 broadcast or group update message, she must acknowledge the most recent prior setup, broadcast or group 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 providers 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 it from all 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, broadcast, and group update messages but not receipts. This was a design choice as requiring receipts to acknowledge receipts would cause significant overhead and excess network traffic. We describe why we made this decision in more detail in Section VI.

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 backward secrecy in Section IV

C. 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:

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

Where H and KDF_1 are used for two-party NAXOS [11] key agreements, KDF_2 is used to produce a random symmetric key from an input string, and KDF_3 is used for anonymous Diffie-Hellman. S is the set of possible session identifiers and U is the set of possible participant identifiers. 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.

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 ($IND\$ - CPA$) [17] and integrity of ciphertext security ($INT - CTXT$) [2]. We choose AES-GCM with random IVs for our AEAD scheme.

D. Registration

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

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

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

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

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 for a single session. The participants must enforce this rule.

E. 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 broadcast messages, in which case it is random symmetric key material. When sending setup, receipt, and group 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 providers where id_b is the id of epk_b. Alice generates a new ephemeral key:

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

Then Alice computes a symmetric key:

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

Alice generates her next ephemeral key pair:

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

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:

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

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. 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.

F. Provider Authentication Block

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

These blocks are simple AEAD ciphertexts that authenticate the message d as associated data. When Alice comes online she simply sends to every provider $j \in S$ a symmetric key $k_{aj} \leftarrow_R \{0,1\}^l$. Then when Alice sends a protocol message d she first generates her next key $k'_{aj} \leftarrow_R \{0,1\}^l$, then computes the authentication block:

$$auth_{aj} \leftarrow Enc_{k_{aj}}(k'_{aj}, d)$$

Provider j verifies d and decrypts the next symmetric key k'_{aj}.

G. Setup Message

All conversation messages are similar in format. For Alice to setup a conversation she generates a random Sid and computes the setup message:

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

Next, Alice computes the two party ciphertext block c_{ai} for every participant $i \in P \setminus \{Alice\}$ as described in Section III-E, 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 provider authentication block $auth_{aj}$ for every provider $j \in S$ as described in Section III-F where $data_1$ is the associated data to authenticate. Alice then sends to the routing provider:

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

where $s = |S|$.

The routing provider sends to each mirror provider j the message $data_1, auth_{aj}$. Each provider can verify the message $data_1$ is from Alice. Then every provider sends to participant $i \in P \setminus \{Alice\}$ the message $Alice, data_0, c_{ai}$. Every user verifies that $data_0, c_{ai}$ is from Alice as detailed in Section III-E. The providers send $data_0$ to Alice and she verifies it is the $data_0$ she sent.

Once a participant has received the setup message from every 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 forwarding the message.

H. Receipt Message

Participants send receipts after they have accepted any setup, broadcast, or group update message. If multiple messages are sent while Alice is offline she sends a single receipt that acknowledges all messages m_i with participant $i \in P \setminus \{Alice\}$. The messages to acknowledge depend on the participant they are being acknowledged to. m_i is composed of all protocol message excluding receipts 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. Let m be a list of all of the unacknowledged setup, broadcast, and group update messages.

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

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

Then she computes the two party ciphertext block c_{ai} for every participant $i \in P \{Alice\}$ as detailed in Section III-E 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 provider authentication blocks as detailed in Section III-F with the associated data as $data_1, m$. Finally, she sends $data_1$ and the authentication blocks to the routing provider.

The providers can verify that all unacknowledged messages are acknowledged in their respective provider authentication block. If the receipt does not acknowledge all unacknowledged messages it is dropped. The providers then send $data_0, c_i$ to participant i. Every participant can verify all of Alice's unacknowledged messages with their respective ciphertext block. Alice verifies $data_0$ is the message she uploaded. If the message does not verify the conversation is terminated.

I. Broadcast Message

Broadcast messages are similar to receipts except they contain a ciphertext. Let pm be a list of all protocol messages up-to and including the last setup, broadcast, or group update message in a conversation. When Alice wants to send the broadcast 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

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

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

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

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

Each provider verifies the message and sends $data_0, c_{ai}$ to participant i. After receiving the message from every provider each participant verifies and then displays the message. If the message does not verify the conversation is terminated.

J. Group Update Message

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

Group update messages are similar to broadcast messages except that the broadcast message ciphertext is replaced with a list of participants. Again let pm be a list of all protocol messages up-to and including the last setup, broadcast, or group update message in the conversation. When Alice wishes to change the participants of a conversation to P' she creates a message:

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

She then creates the ciphertext block c_{ai} for participant $i \in (P \cup P') \setminus \{Alice\}$ as described in Section III-E where $data_0, pm$ 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 III-F with $data_1$ as the associated data. She uploads $data_1$ along with the provider authentication blocks to the routing provider and the routing provider distributes the message to the other mirror providers.

Each provider checks that Alice is allowed to make the desired group modification and sends $data_0, c_{ai}$ to all $i \in (P \cup P') \setminus \{Alice\}$ and sends $data_0$ to Alice. 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.

K. Two Party Channels

All communication between the clients and providers and between providers is performed over a two-party channel that supplies all of the security properties discussed in Section II. The mirror providers act as clients when communicating with the routing provider. This is a synchronous channel that is setup by first performing an anonymous Diffie-Hellman key agreement followed by 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. Algorithm 1 details the algorithm for setting up the channel from the initiator. Line 1 generates the DH

Algorithm 1 Client To Provider Channel Setup

1: **function** C2SCHANNELSETUP(C, S, lpk_s, lsk_c)
2: $esk_c[0] \leftarrow_R Z_p^*$, $esk_c[1] \leftarrow \{0,1\}^l$
3: $epk_c[0] \leftarrow g^{esk_c[0]}$, $epk_c[1] \leftarrow g^{H(esk_c[1], lsk_c)}$
4: $k_0 \leftarrow KDF_3(lpk_s^{esk_c[0]})$
5: $c_0 \leftarrow Enc_{k_0}(C, epk_c[1])$
6: SEND($S, epk_c[0], c_0$)
7: $epk_s[0], c_1 \leftarrow$ RECV(S)
8: $km_1 \leftarrow lpk_s^{H(esk_c[1], lsk_c)}$
9: $km_2 \leftarrow epk_s[0]^{lsk_c}$
10: $km_3 \leftarrow epk_s[0]^{H(esk_c[1], lsk_c)}$
11: $k_1 \leftarrow KDF_1(km_1, km_2, km_3, S, C)$
12: $t, epk_s[1] \leftarrow Dec_{k_1}(c_1)$
13: $esk_c[2] \leftarrow_R Z_p^*$, $epk_c[2] \leftarrow g^{esk_c[2]}$
14: $km_4 \leftarrow epk_s[1]^{lsk_c}$
15: $km_5 \leftarrow lpk_s^{H(esk_c[1], lsk_c)}$
16: $km_6 \leftarrow epk_s[1]^{H(esk_c[1], lsk_c)}$
17: $k_1 \leftarrow KDF_1(km_4, km_5, km_6, C, S)$
18: $c_2 = Enc_{k_2}(t, epk_c[2])$
19: SEND(S, c_2)
20: **return** esk_c, epk_s

keys for the anonymous DH key agreement and line 2 generates the ephemeral NAXOS keys. Lines 5-6 encrypts the clients identity and NAXOS ephemeral key and send it to the provider. Line 7 receives the provider's response and line 8-9 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 11-13 ratchets the channel keys and sends the challenge back.

Algorithm 2 details setting up the channel from the provider. Lines 2-4 compute the anonymous DH shared secret using the provider's long-term key and decrypt the clients identity and NAXOS ephemeral public key. Line 5 looks up the long-term public key of the client and line 6-8 compute the NAXOS shared key. Lines 9-11 encrypt the challenge and the providers next ephemeral DH key and send it to the client. Lines 12-18 decrypt the clients response and check that the client's response matches the challenge.

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

Algorithm 2 Provider To Client Channel Setup

1: **function** S2CCHANNELSETUP(S, C, lsk_s)
2: $epk_c[0], c_0 \leftarrow$ RECV(C)
3: $k_0 \leftarrow KDF_3(epk_c[0]^{lsk_s})$
4: $C, epk_c[1] \leftarrow Dec_{k_0}(c_0)$
5: $lpk_c \leftarrow$LOOKUPUSER(C)
6: $esk_s[0] \leftarrow 0, 1^l, epk_s[0] \leftarrow g^{H(esk_s[0], lsk_s)}$
7: $esk_s[1] \leftarrow Z_p^*, epk_s[1] \leftarrow g^{esk_s[1]}$
8: $km_1 \leftarrow epk_c[1]^{lsk_s}$
9: $km_2 \leftarrow lpk_c^{H(esk_c[0], lsk_s)}$
10: $km_3 \leftarrow epk_c[1]^{H(esk_s[0], lsk_s)}$
11: $k_1 \leftarrow KDF_1(km_1, km_2, km_3, S, C)$
12: $t_s \leftarrow_R \{0,1\}^l$
13: $c_1 \leftarrow Enc_{k_1}(t_s, epk_{s[1]})$
14: SEND($C, epk_s[0], c_1$)
15: $c_2 \leftarrow$ RECV(C)
16: $km_4 \leftarrow lpk_c^{H(esk_s[1], lsk_s)}$
17: $km_5 \leftarrow epk_c[1]^{lsk_s}$
18: $km_6 \leftarrow epk_c[1]^{H(esk_s[1], lsk_s)}$
19: $k_3 \leftarrow KDF_1(km_4, km_5, km_6, C, S)$
20: $t_c, epk_c[2] \leftarrow Dec_{k_2}(c_1)$
21: **if** $t_s = t_c$ **then**
22: **return** (C, esk_s, epk_c)
23: **else**
24: **return** $\perp$

Algorithm 3 Channel Send

1: **function** SECURESEND($S, R, m, lsk_s, esk_s, lpk_r, epk_r$)
2: $n_s \leftarrow |esk_r|$
3: $n_r \leftarrow |epk_r|$
4: $km_1 \leftarrow epk_r[n_r - 1]^{lsk_s}$
5: $km_2 \leftarrow lpk_r[n_r - 1]^{H(esk_s[n_s-1], lsk_s)}$
6: $km_3 \leftarrow epk_r[n_r - 1]^{H(esk_s[n_s-1], lsk_s)}$
7: $k \leftarrow KDF_1(km_1, km_2, km_3, S, R)$
8: $esk_s[n_s] \leftarrow Z_p^*, epk_s[n_s] \leftarrow g^{esk_s[n_s]}$
9: $c \leftarrow Enc_k(m, epk_s[n_s])$
10: SEND($R, n_r - 1, c$)
11: **return** (esk_s, epk_r)

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 Channel Receive

1: **function** SECURERECV(R, S, esk_r, epk_s)
2: $n_s \leftarrow |epk_s|$
3: $n_r, c \leftarrow$RECV(C)
4: $km_1 \leftarrow lpk_s^{H(esk_r[n_r-1], lsk_r)}$
5: $km_2 \leftarrow epk_s[n_s - 1]^{lsk_r}$
6: $km_3 \leftarrow epk_s[n_s - 1]^{H(esk_r[n_r-1], lsk_r)}$
7: $k \leftarrow KDF_1(km_1, km_2, km_3, S, R)$
8: $m, epk_s[n_s] \leftarrow Dec_k(c)$
9: **return** (esk_r, epk_s)

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.

L. 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 [1].

IV. 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 II. We provide full proofs in Appendix A. We model our hash function (H) and key derivation functions (KDF_1, KDF_2, KDF_3) 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.

A. 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:

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

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.

B. Message Authentication and Integrity

Message authentication implies message 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 or sessions. 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:

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

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.

C. 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

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

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:

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

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 broadcast 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.

D. Backward Secrecy

Backward 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 broadcast messages following Alice's next receipt provide backward secrecy. Similar to forward secrecy we need to show that Alice's compromised ephemeral private keys are not used in the next broadcast 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 broadcast message does not use esk_{ai}. Let Alice's receipt be:

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

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 broadcast 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.

E. 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 provider or other 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 group update message and not terminating. Since setup and group update messages in Mobile CoWPI are part of the protocol transcript and Mobile CoWPI provides conversation integrity, Mobile CoWPI also provides participant consistency.

F. 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, broadcast, and group change messages. Conversation integrity considers an adversary that controls the network, can compromise all but one provider, and can compromise participants in the conversation. The adversary is not allowed to compromise all the providers, otherwise breaking conversation integrity is trivial, irregardless 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:

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

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 receiving it from all providers and an honest provider will forward all messages in a consistent order to all participants. The adversary must be able to send a message to an honest participant A as if it came from honest provider S. If the adversary can inject such a message, it must be able to do one of the following:

1) Compute a two-party NAXOS key between A and S in their two-party channel, allowing the adversary to send any message to A.
2) Break the INT-CTXT property of the AEAD scheme of the two-party channel between A and S.

G. Deniability

Recall deniability as discussed in Section II. Deniability is provided if a single user can run a simulator and produce a simulated transcript that is indistinguishable from transcript of a real protocol execution. The simulator must only take as input information that is known to a single user. 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.

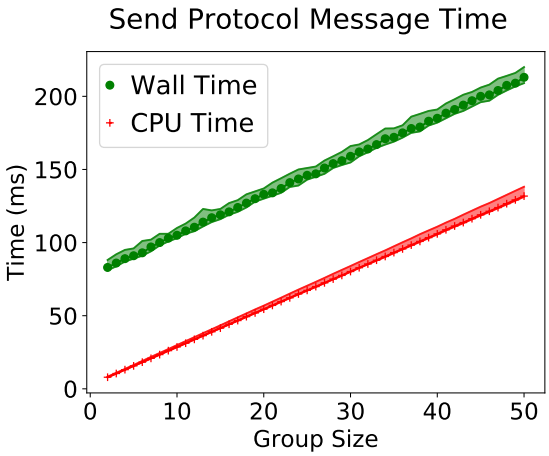

Fig. 1: The wallclock and cpu time (25th, 50th, and 90th percentile) to send a protocol message.

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. Alice will generated any ephemeral keys she does not have as random group elements as before. She then generates the NAXOS key as a random symmetric key.

As before, 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.

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 [8], 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.

H. Anonymity Preserving

A protocol preserves anonymity if the protocol does not undermine any anonymity features of the underlying transport protocol. The adversarial model for this property is that of a passive network adversary, e.g. the user's internet service provider. Mobile CoWPI preserves anonymity as it does not reveal the identity or long-term key of a user in plaintext. This is achieved by using anonymous Diffie-Hellman to initiate the connections between the clients and providers.

If an adversary can break the anonymity preserving property they must be able to do one of the following:

1) Compute the anonymous DH shared secret between the client and provider.
2) Break the IND$-CPA property of the AEAD scheme.

V. Evaluation

We implemented Mobile CoWPI as a Java server and client library. Since all protocol messages can be processed without interaction between clients the overhead of Mobile CoWPI is low. To measure the run time overhead we deployed Mobile CoWPI with two servers, a routing server and a single mirror. Only two servers are necessary since conversation integrity only requires a single honest provider. The routing server is run on a $10 dollar/month linode [12] virtual private server hosted in New Jersey. The mirror is hosted at University of Tennessee, Knoxville

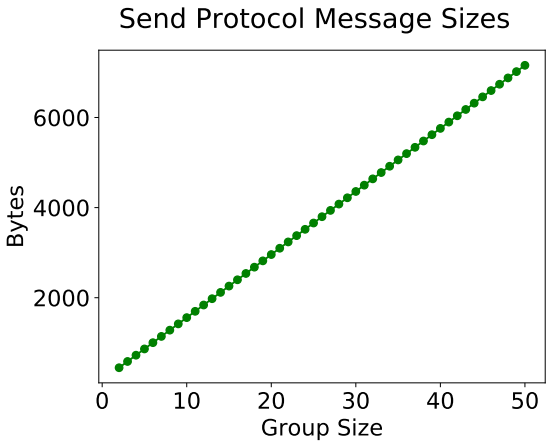

Fig. 2: The message size in bytes to send a message.

on a desktop computer with an Intel Core i7-6700K processor. The client recording measurements was running on a desktop computer at the University of Minnesota, with an Intel Xeon processor. The network round-trip-time between all locations is $31ms$. We ran the measurements with 2 to 50 participants in a conversation and sent 100 messages for each size of conversation. These measurements show Mobile CoWPI is practical for real world deployments.

Figure 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 servers. This is due to the authentication being pairwise with all receivers. We discuss why this overhead is necessary in Section VI. Pairwise ciphertext blocks does allow for very little overhead to receive a message. Broadcast and receipt messages are constant size while setup and group update message must be $O(n)$ in size to distribute the list of participants. The overhead of Mobile CoWPI for incoming messages is less than 300 bytes.

VI. Limitations

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

A. Pairwise Ciphertext Blocks

The pairwise ciphertext blocks attached to every protocol message cause the most overhead in Mobile CoWPI. We use these pairwise ciphertext blocks to provide deniable message author authentication. We argue that the pairwise nature of our solution seems to be an inherent result of the system properties of deniability and authorship authentication. In particular, we now sketch a proof that a scheme in which each user uses a single public and private key pair to deniably authenticate messages to the full group is impossible.

Consider the system model property that a participant should be able to authenticate the author of a message even when all other participants are offline. This means message authentication must be non-interactive. Also recall that deniability requires a user to be able to simulate a conversation even if the distinguisher has access to the long-term private keys of all participants. It must be the case that a user can simulate an authorship authentication that the distinguisher will accept but a real user will reject. However, the distinguisher has the real users' secret keys and can use the same verification method to distinguish the simulated authorship authentication from a real one. A more detailed discussion is provided in Appendix A

Laguillaumie and Vergnaud provide a multi-designated verifier signatures [10] scheme providing signatures that can be validated by anyone and that provide a *source hiding* property that the signature was either generated by the author or the group of receivers. This scheme could be used for deniable group messaging, however it is a weaker form of deniability than Mobile CoWPI provides since simulation requires all users. Additionally, Mobile CoWPI is built on well understood efficiently implementable primitives.

B. 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 (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. (And at least one providers is required for offline participants to receive messages)

C. 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 the server or participants.

D. Receipt Ordering

As described in Section III, receipts do not acknowledge other receipts. To explain why we made this decision, consider the case where receipts are required to acknowledge all previous receipts. Upon receiving a broadcast message, all users will attempt to send a receipt immediately but only one receipt will be accepted. The remaining $n-1$ will be rejected as they do not acknowledge the accepted receipt. Then the remaining $n-1$ users will attempt to send the another receipt with only one being accepted. This issue continues until all users have a receipt accepted and creates a significant amount of wasted traffic which delays conversation progress.

VII. Related Work

Off-The-Record (OTR) [3] 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) [8] tries to provide the properties of OTR for group conversations. At a high level it works as follows. First, All participants setup pairwise secure channels using a deniable authenticated key agreement (DAKE). Then over the secure channels the participants execute a Group Key Agreement (GKA) to compute an ephemeral encryption key. The users also distribute ephemeral verification keys used to sign conversation messages. The participants also compare a hash of the group information to enforce participant consistency. When Alice wants to send a message to the group she encrypts the message with the ephemeral group key then signs the ciphertext with her ephemeral verification key. Then broadcasts the ciphertext and signature to all participants of the conversation. All recipients can verify the signature is from Alice and decrypt the message. To enforce conversation integrity, at the end of a conversation the participants execute a byzantine agreement on a lexographically ordered list of the messages. Even though mpOTR provides participant repudiation via the DAKE during setup it does not provide message unlinkability due to the use of the verification keys. With knowledge of a verification key a distinguished can verify all messages authored by a particular user. mpOTR also lacks strong conversation integrity since the transcript consistency is not checked until the conversation has ended and is only checked on a lexographically order transcript. This requires mpOTR to operate in the non-mobile, synchronous, model with static participants.

Group Off-The-Record (GOTR)[13] 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 [20] 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 in the first phase. 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 [21] (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], [4], [9]. With [19], [18] 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 [16] 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) [5] 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.

VIII. 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 servers must be used to provide strong conversation integrity and (2) to provide message unlinkability, messages must be $O(n)$ in size and must provide pairwise unlinkability. We then showed that given an any-trust multiple-server model, a relatively simple protocol, Mobile CoWPI, can achieve these strong security properties while being practically efficiency. 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.

References

[1] C. Alexander and I. Goldberg, "Improved user authentication in off-the-record messaging," in *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM, 2007, pp. 41–47.

[2] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, no. 4, pp. 469–491, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1007/s00145-008-9026-x

[3] N. Borisov, I. Goldberg, and E. Brewer, "Off-the-record communication, or, why not to use pgp," in *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*. ACM, 2004, pp. 77–84.

[4] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," in *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*. IEEE, 2017, pp. 451–466.

[5] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 1802–1819.

[6] M. Di Raimondo, R. Gennaro, and H. Krawczyk, "Deniable authentication and key exchange," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 400–409.

[7] T. Frosch, C. Mainka, C. Bader, F. Bergsma, J. Schwenk, and T. Holz, "How secure is textsecure?" in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 457–472.

[8] I. Goldberg, B. Ustaoğlu, M. D. Van Gundy, and H. Chen, "Multi-party off-the-record messaging," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 358–368.

[9] N. Kobeissi, K. Bhargavan, and B. Blanchet, "Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach," in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[10] F. Laguillaumie and D. Vergnaud, "Multi-designated verifiers signatures," in *International Conference on Information and Communications Security*. Springer, 2004, pp. 495–507.

[11] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Provable Security*. Springer, 2007, pp. 1–16.

[12] linode, *linode*, https://linode.com/.

[13] H. Liu, E. Y. Vasserman, and N. Hopper, "Improved group off-the-record messaging," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 2013, pp. 249–254.

[14] M. Marlinspike, "Facebook messenger deploys signal protocol for end to end encryption," 2016. [Online]. Available: https://whispersystems.org/blog/facebook-messenger/

[15] ——, "Open whisper systems partners with google on end-to-end encryption for allo," 2016. [Online]. Available: https://whispersystems.org/blog/allo/

[16] M. Marlinspike and T. Perrin, "The x3dh key agreement protocol," 2016. [Online]. Available: https://whispersystems.org/docs/specifications/x3dh/

[17] P. Rogaway, "Nonce-based symmetric encryption," in *International Workshop on Fast Software Encryption*. Springer, 2004, pp. 348–358.

[18] P. Rösler, C. Mainka, and J. Schwenk, "More is less: On the end-to-end security of group chats in signal, whatsapp, and threema," 2018.

[19] M. Schliep, I. Kariniemi, and N. Hopper, "Is bob sending mixed signals?" in *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. ACM, 2017, pp. 31–40.

[20] M. Schliep, E. Vasserman, and N. Hopper, "Consistent synchronous group off-the-record messaging with sym-gotr," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 181–202, 2018.

[21] O. W. Systems, *Open Whisper Systems*, https://whispersystems.org/.

[22] P. Syverson, R. Dingledine, and N. Mathewson, "Tor: The secondgeneration onion router," in *Usenix Security*, 2004.

[23] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, "Sok: Secure messaging," in *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 2015, pp. 232–249.

[24] WhatsApp, 2017. [Online]. Available: https://www.whatsapp.com/security

Fig. 3: IND$-CPA Game

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

Fig. 4: INT-CTXT Game

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

APPENDIX

A. Security Assumptions

We assume our symmetric AEAD scheme ciphertext are indistinguishable from random bit strings (IND$-CPA) detailed in Figure 3 and provides integrity of ciphertext detailed in Figure 4. The advantage of an adversary M winning each of the games is defined as $Adv^{IND-CPA}(M) = Pr[M \text{ wins}] - \frac{1}{2}$, $Adv^{INT-CTXT}(M) = Pr[M \text{wins}]$ respectively.

We assume the NAXOS protocol is a secure authenticated key agreement. Figure 5 describes the game used by the original authors modified to include an addition bit string ($\{0,1\}^l$) into the EPHEMERAL KEY REVEAL and TEST queries that is included in the input of KDF_2 of NAXOS. This modification is to allow the Mobile CoWPI session id Sid to be incorporated into the KDF and does not affect the security of NAXOS. The NAXOS session id is

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

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

An adversary wins if if it issues 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

Fig. 5: NAXOS Game

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

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

B. 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 6.

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

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

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

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

Theorem A.1. *Mobile CoWPI provides message confidentiality if all hash and key derivation functions are modeled as random oracles.*

Fig. 6: Message Confidentiality Game G_0

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

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

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

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

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

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

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

function TEST(Sid, P, m)
if then$b = 0$
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 $(d = b)$

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$-CPA game.
- The challenger replaces the ciphertext of the test broadcast message an IND$-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($m, \cdot$).

Fig. 7: Message Authentication Game

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

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

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

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

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

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

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

function FINALIZE
return $True$ if protocol broadcast message pm was accepted by R in session Sid from P where $pm \notin Out[P]$, and R and P are clean, else $False$.

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. ■

C. 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 7 captures the message authentication and integrity property in a game similar to message confidentiality. The adversary first INITIALIZES the PKI and can register adversary controlled users and long-term keys. The adversary controls the network and uses the SEND function to send messages between users and servers. The adversary may also instruct honest users to SETUPGROUP, SENDGROUPMESSAGE, and UPDATEPARTICIPANTS as with message confidentiality. The adversary is allowed to REVEALLONGTERMKEYS and REVEALEPHEMERALKEYS of users. Finally, the adversary FINALIZES the game and wins if a participant R accepted protocol broadcast message pm from P in session Sid where the P did not send c and R and P have not had their long-term and ephemeral keys of ciphertext block of pm revealed.

That is R must have received a message:

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

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

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

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

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

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

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

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

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

G_0	The challenger behaves correctly.
G_1	The challenger replaces the NAXOS key exchange used to decrypt a random ciphertext block between the sender and a random receiver of a random forged message with a random key.
G_2	The challenger replaces the ciphertext block of a forged message between the sender and a random receiver with an instance of the INT-CTXT game.

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

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

The challenger M_0 deviates from G_0 in the following way:

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

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

1) $epk_{ab} \leftarrow$ SEND(A, B)
2) $epk_{ba} \leftarrow$ SEND(A, B, epk_{ab}), epk_{ba} may be a pre-key of B.
3) When computing a NAXOS key of a ciphertext block not part of the test message, $k_{ab} \leftarrow$ 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.

Fig. 8: Conversation Integrity Game G_0

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

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

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

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

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

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

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

function FINALIZE()
return $True$ if honest user A accepts protocol message pm_a as the i^{th} message of Sid and honest user B accepts protocol message pm_b as the i^{th} message of session Sid and $pm_a \neq pm_b$. Otherwise return $False$.

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

■

D. Conversation Integrity

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

Figure 8 details the conversation integrity game. First the adversary INITIALIZES the PKI and registers corrupt users and servers. The adversary may then issue commands instructing participants and servers 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 servers 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 IV 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 KDF_2 oracle queries and sends at most y messages between honest servers 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 encrypt a random message 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 two-party channels and is uses 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 two-party channel 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 ciphertext block not part of the test message, $k_{ab} \leftarrow$ REVEAL($A, B, epk_{ab}, epkba$) is used as the key.
5) 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 two party message 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.

Fig. 9: Deniability Game G_0

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

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

- The challenger replaces a random two-party channel ciphertext message between an honest server and participant with an INT-CTXT game detailed next.

We now detail how the challenger M_1 generates the random two-party ciphertext message between an honest server 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 ciphertext $c'_{ab} \neq c_{ab}$, the challenger submits c'_{ab} to VF of the INT-CTXT game.

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

E. Deniability

We capture the deniability property with the general-purpose game detailed in Figure 9. The distinguisher INITIALIZES the game with a plaintext transcript τ. Then the challenger executes Mobile CoWPI on τ producing a real protocol transcript T_0 and three outputs $input_d$, $input_s$, $output_0$. The challenger then runs a simulator with inputs τ and $input_s$ producing a forged protocol transcript T_1 and state $output_1$. The challenger returns a random transcript T_b, output $output_b$, and $input_d$ to the distinguisher. The distinguisher wins the game if it guesses b correctly. The advantage of the distinguisher M is defined as $Adv^{DENY-*}(M) = Pr[M \text{ wins}] - \frac{1}{2}$. The DENY-* game depends on how $input_d$, $input_s$, and $output_*$ are defined.

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

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

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

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

F. Message Unlinkability

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

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

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

This definition provides the distinguisher with knowledge of a non-deniable protocol message. The goals is to simulate a transcript that contains pm_i and is identically distributed to the real. The message unlinkability simulator behaves as a participant repudiation simulator discussed earlier. When the simulation party sends its' last ciphertext block prior to pm_i the simulator uses $epk_{ai} \leftarrow g^{H(esk_{ai}, lsk_a)}$ as the next ephemeral public to the sender. Similarly,

Fig. 10: Deniable Group Message Authentication

function INITIALIZE(U)
$b \leftarrow_R \{0,1\}$
Generate secret and public keys sk_i, vk_i for all $i \in U$.
return all secret and public keys.

function TEST($m, a, c, vk_0, \ldots, vk_c, \ldots, vk_n$)
$\sigma_0 \leftarrow$ AUTH($m, P, sk_a, vk_0, \ldots, vk_n$)
$\sigma_1 \leftarrow$ SIMULATE($m, P, sk_a, vk_0, \ldots, vk_n$)
return σ_b

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

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.

G. Deniable Group Message Authentication

We now show that a deniable group message authentication primitive is impossible. We first define the scheme of deniable group message authentication, detail the deniability security game, then provide an adversary that can always win the game.

A deniable group message authentication scheme is defined a follows:

- SETUP(l) takes as input the security parameter l and outputs public parameters P.
- KEYGEN(P) takes as input the parameters P and outputs a secret value sk_a and public verification value vk_a for user a. Each user will generate own secret values and publish their verification value.
- AUTH($m, P, sk_a, vk_0, \ldots, vk_n$) takes as input a message to authenticate m the public parameters P, the secrete value of the author sk_a, and the verification values of all of the users the message is authenticated to $vk_0, \ldots, vk_n$. The output is σ_a.
- VERIFY($m, \sigma_a, P, sk_c, vk_a$) returns true if σ_a was generated by a executing AUTH($m, P, sk_a, vk_0, \ldots, vk_b, \ldots, vk_n$) or $c = a$. Otherwise returns false. VERIFY is allowed to return true if $c = a$ as a can keep track of all messages she authenticated and drop any message she has not authenticated but verifies as from her.
- SIMULATE($m, P, sk_c, vk_0, \ldots, vk_a, \ldots, vk_n$) simulates authentication of message m by a. It is executed by c and returns σ_a that must be indistinguishable from from a executing AUTH on m.

Figure 10 describes a security game that captures the deniability of group message authentication. A distinguisher plays the game and wins if it can distinguish a SIMULATEd σ from an AUTH σ. The distinguisher is given all of the secret and verification keys of all users.

We now describe a distinguisher that can always win the deniability game. On TEST output σ_b, let e be a valid verifier of σ_b, $e \neq a$, and $e \neq c$. If VERIFY($m, \sigma_b, P, sk_e, vk_a$) returns true guess 0, else guess 1. By the definition of VERIFY a SIMULATEd value will never return true. Thus, a deniable group authentication scheme does not exist.

This model does not capture ephemeral keys. However we argue that any ephemeral keys would need to be distributed in a deniably authenticated manner, necessitating a scheme that does not use ephemeral keys. Mobile CoWPI provides deniable authentication as all authentication is pair wise.