Michael Schliep*, Eugene Vasserman, and Nicholas Hopper

# Consistent Synchronous Group Off-The-Record Messaging with SYM-GOTR

**Abstract:** We describe SYM-GOTR, a protocol for secure Group Off-The-Record (GOTR) messaging. In contrast to previous work, SYM-GOTR is the first protocol to offer confidential, authenticated, and repudiable conversations among a dynamic group with the additional properties of message unlinkability and the guarantee that all users see the same conversation, while providing efficient use of network and CPU resources. SYM-GOTR achieves these properties through the use of a novel optimistic consistency check protocol that either determines that all users agree on a transcript with constant-size messages or identifies at least one user that has not followed the protocol. We provide an implementation of SYM-GOTR as a Java library along with a plugin for the Jitsi instant messaging client. We analyze the performance of SYM-GOTR in a real world deployment scenario and discuss the challenges of providing a usable implementation without compromising the security of the conversation.

DOI 10.1515/popets-2018-0027
Received 2017-11-30; revised 2018-03-15; accepted 2018-03-16.

## 1 Introduction

Recent disclosure of state level surveillance on citizens around the globe [1] has caused an increased interest in secure means for electronic communication. What it means to have *secure* communication can be difficult to define with the myriad of devices and services in use today, and many systems have been introduced to fill this need each with varying adversarial models and security properties. An early example of these systems is PGP, which provides confidentiality to emails, while a more modern approach is WhatsApp, offering private real time messaging to 1 billion users. Our goal is to provide an online equivalent to Off-The-Record (OTR) face-to-face group conversations without the need of a proprietary service provider.

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

Earlier works have attempted to address these problems. Borisov, Goldberg, and Brewer [2] introduced the challenges of providing similar security properties for online conversations and offline conversations, and proposed the original OTR protocol, which provides these properties but is limited to a two user setting. Multi-party OTR (mpOTR) [3] was introduced for secure group conversations. While offering confidentiality, authentication, participant consistency and repudiation, mpOTR does not provide message unlinkability, since all messages from the same sender in a transcript can be linked together; transcript consistency, since an adversary controlling the network can cause participants to terminate with differing transcripts; or a mechanism for dynamic groups.

Two more recent systems have the functionality to handle dynamic groups, BD-GOTR [4] and Signal [5](formerly known as TextSecure). BD-GOTR is similar to our work, but lacks message unlinkability and does not provide participant consistency or strong transcript consistency. Signal's goals are different from the previously mentioned works, in that Signal focuses on asynchronous messaging. Schliep, Kariniemi, and Hopper [6] have shown Signal does not provided speaker integrity, a global transcript, or participant consistency.

We describe a new protocol for group OTR instant messaging, Symmetric Group-Off-the-Record (SYM-GOTR). SYM-GOTR is the first protocol to simultaneously provide message unlinkability, participant consistency, and strong transcript consistency for dynamic groups, allowing an online group chat with similar properties to private offline conversations. SYM-GOTR pro-

***Corresponding Author: Michael Schliep:** University of Minnesota, E-mail: schli116@umn.edu
**Eugene Vasserman:** Kansas State University, E-mail: eyv@ksu.edu
**Nicholas Hopper:** University of Minnesota, E-mail: hoppernj@umn.edu

vides high-security for small synchronous groups where as previous protocols have made a trade-off in terms of security properties provided to support larger or asynchronous groups. Compared to previous work, SYM-GOTR is efficient in terms of the number and size of protocol messages; under benign conditions all operations require a constant number of messages sent between all pairs of users and all messages are of constant size.

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

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

In Section 2 we formally describe the goals of SYM-GOTR and discuss the current state of private group messaging. Section 8 describes the protocol with Section 4 overviewing the security of SYM-GOTR. We analyze our implementation's performance in Section 5. We conclude by discussing challenges of implementing a secure messaging application in Section 6.

1 https://github.com/mschliep/gotr4j

2 https://github.com/mschliep/jitsi

# 2 Private Group Messaging

## 2.1 Goals

Unger et al. [7] perform an analysis of multiple secure message designs and implementations. They present a list of goals a secure messaging protocol may provide. Below we describe the goals of our system in terms consistent with their work.

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

- **Confidentiality** A message may only be read by conversation participants.
- **Integrity** No honest party will accept a modified message.
- **Authentication** Participants receive proof of possession of a long term secret from every other participant. Additionally, all participants can verify the author of a broadcast message.
- **Participant Consistency** All honest parties agree on the set of participants.
- **Destination Validation** Honest parties can verify they are an intended recipient of a message.
- **Forward and Backward Secrecy** Previous and future messages are secure when the key material is compromised.
- **Anonymity Preserving** The protocol does not undermine the anonymity features of the underlying messaging service or transport.
- **Speaker Consistency** All honest parties agree on the sequence of messages sent by any one participant.
- **Causality Preserving** Messages may only be displayed after messages that causally proceed them have been displayed. If two messages ($m_2$, $m_3$) are both sent in response to $m_1$, a causality-preserving protocol can display the messages in the order ($m_1$, $m_2$, $m_3$) or ($m_1$, $m_3$, $m_2$), even if $m_2$ arrives before $m_3$.
- **Global Transcript** All participants agree on the order of all messages before processing the next message. A global transcript implies speaker consistency and causality preservation.

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

- **Message Unlinkability** Proving authorship of one message does not prove authorship of any other message.

- **Message Repudiation** There is no way to prove a user authored any message.
- **Participation Repudiation** There is no way to prove a user participated in a chat.

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

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

Unger et al. [7] discussed additional properties that are sometimes mutually exclusive to ours. The properties we do not offer focus on asynchronous protocols without a global transcript. These properties are:

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

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

## 2.2 Related Work

The original OTR protocol [2] provided the security properties we desire for two party communication but there does not exist a mechanism to scale the protocol to a group setting. Goldberg et al. proposed a multi-party OTR protocol (mpOTR) [3] with a subset of our goals. They were the first to discuss deniability in the group setting, both in terms of message deniability and also the requirement of participant repudiation. mpOTR provides secure group messaging for a static set of users. Briefly, the mpOTR protocol operates as follows. Out of band, users agree to the set of participants. Then in-band, users perform a group consistency check, set up deniable peer-to-peer (p2p) channels between each pair of users, and then share an ephemeral signature verification key. After this the group performs a Group Key Agreement (GKA), the details of which are not addressed. When a user wishes to send a message, they encrypt it with the group key, and sign it with their ephemeral signing key. After the conversation is complete, users sign and share the lexicographical ordered digest of the entire chat, along with any unreceived messages.

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

Liu, Vasserman, and Hopper [4] proposed an improved group OTR (GOTR) protocol which we refer to as BD-GOTR due to its use of the Burmester-Desmedt group key agreement protocol. The design and goals of their protocol are most similar to our work. As we will discuss, however, BD-GOTR provides only partial mes-

sage unlinkability and does not offer participant consistency, or strong guarantees about a global transcript. The protocol provides optional forward and backward secrecy at the cost of re-executing the setup phase. Finally, the size of some messages in BD-GOTR scales linearly with the size of the group, which makes the protocol inefficient for larger group conversations.

BD-GOTR works as follows. Out of band, users agree to the list of chat participants. Users set up secure p2p channels between every pair of users $U_i$ and $U_j$. Then $U_i$ and $U_j$ perform a three step process to compute a Burmester-Desmedt GKA [8] between the pair. Each user then uses a "Hotplugging" property to combine their $n-1$ (pairwise) group keys into a single "circle" key. Each user broadcasts their circle public key, and all other users are able to compute the circle key from having participated in a single BD-GKA with that user. To send a secure message a participant uses a Key Derivation Function (KDF) to generate a symmetric encryption and MAC key from their circle key, then encrypts the message and chat transcript digest with the keys. The user then broadcasts the encrypted message along with their circle public key to the group. Upon receiving the broadcast all users perform a digest consistency check over the p2p channels. This allows the participants to agree upon the author of the message.

BD-GOTR provides *partial* message unlinkability in that if it can be shown that user $U_i$ contributed some message to a conversation, the user's confirmation of the conversation digest implicitly confirms any previous messages attributed to the user in the transcript, while *not* linking the user to messages appearing later in the transcript (in contrast to mpOTR, in which confirming one message sent by $U_i$ confirms all messages sent by $U_i$ in the transcript).

Signal [5, 9, 10] is an asynchronous secure group messaging protocol. The goal of Signal is to provide a secure alternative to text messages. Its security goals are similar to ours but Signal focuses more on asynchronous communication and abandons the global transcript property. Signal requires a service provider to support authentication and message delivery. The Signal authors do not provide a formal list of goals, threat model, or security proofs. Recent academic work has started to provide these missing pieces. Frosch et al. [11] analyzed an earlier version of TextSecure and show that the protocol is an Authenticated Key Exchange. They also identify an Unknown Key Share attack against the protocol. Cohn-Gordon et al. [12] analyze the Signal protocol as a multi-stage key execute protocol. They also provide the most detailed description of the Signal protocol as deployed. Kobeissi et al. [13] use a novel automated verification technique to identify a key compromise impersonation attack and a replay attack in Signal. They show a modified signal protocol protects against these attacks. Finally, Schliep, Kariniemi, and Hopper, [6] have shown Signal does not provide speaker consistency, causality preservation, a global transcript, or participant consistency.

(n+1)sec [14] is a recently published draft of a secure group communication protocol. The goals of (n+1)sec are similar to SYM-GOTR with eventual transcript consistency. (n+1)sec utilizes an mBD+P GKA [15] to generate a group sending key and pairwise secrets between all participants. Each group member generates a signing key used to authenticate messages. Periodically participants send a consistency check message that contains their view of the conversation. The delayed consistency checks allow messages to be displayed with low latency, but violations to the global transcript property may not be detected until some out-of-order messages have already been displayed. (n+1)sec does not provide a formal security analysis or proofs that the desired security properties are provided. In comparison to (n+1)sec, SYM-GOTR prefers simplicity and security at the expense of performance.

## 2.3 System Model

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

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

## 2.4 Threat Model

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

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

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

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

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

# 3 Design

## 3.1 Strawman Design

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

To provide unlinkability, new signing and verification keys need to be generated and distributed for every message. Distributing these new verification keys in an authenticated and unlinkable manner is not cheap. The first idea would be to include them in the plaintext of the previous message. However, this method does not provide unlinkability. The signature on the previous ciphertext commits to the next verification key. These verification keys must be sent over pairwise deniable, unlinkable, authenticated, and confidential channels. This increases the communication cost of the protocol but is needed to achieve all of the properties we want in secure group communication.

We now present the SYM-GOTR protocol.

## 3.2 Primitives

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

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

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

SYM-GOTR uses a signature scheme for accusation of malicious group members. The signature scheme consists of three functions: $KeyGen()$ to generate a ran-

dom signing and verification key pair $(sk_i, vk_i)$ for user $i$, $Sign_{sk}(m)$ to sign the message $m$ with key $sk$, and $Verify_{vk}(m, sig)$ to verify the signature $sig$ of $m$ with verification key $vk$.

In our implementation the authenticated encryption scheme uses AES in Counter Mode along with an HMAC of the ciphertext in an Encrypt-then-MAC fashion as described by Bellare and Namprempre [16]. The IV is always 0 because symmetric keys are never reused. The hash functions and KDFs are built upon SHA-256. The group $G$ is NIST-p256 and ECDSA is used for the signature scheme.

We also assume a messaging server. The server provides group coordination such as group set up and participant change notification as well as peer-to-peer (p2p) communication channels between all participants and a broadcast channel for the group. We also assume the server attempts to send broadcast messages in a consistent order. This order is verified by all clients before message are displayed and the conversation can progress.

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

## 3.3 Overview

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

**Algorithm 1** SYM-GOTR protocol

```
1: for all U_j ∈ U \ {U_i} do
2:     ChannelSetup(Sid, U_j)
3: state ← GroupSetup(Sid, U)
4: while state ≠ ⊥ do
5:     (U_j, c, m) ← SecRecvBcast(Sid, state)
6:     sigs ← BcastConCheck(Sid, U_j, c, m, state)
7:     state ← SigConCheck(Sid, U_j, c, sigs, state)
```

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

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

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

**Algorithm 2** Group Setup

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

**Algorithm 3** Receive Secure Broadcast

```
1:  func SecRecvBcast(Sid, (S, VK, sk_i))
2:      U_j, c ← BcastRecv(Sid)
3:      K_j ← KDF_3(Sid, (U_j, S[j]),
            (U_1, S[1]), ..., (U_n, S[n]))
4:      m ← ADec_{K_j}(c)
5:      if m ≠ ⊥ then
6:          return (U_j, c, m)
7:      else
8:          SecRecvBcast(Sid, (S, VK, sk_i))
```

**Algorithm 4** Broadcast Consistency Check

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

## 3.4 SYM-GOTR Protocol

We first describe the SYM-GOTR protocol assuming we have a protocol for a confidential, deniable, unlinkable, authenticated p2p channel. With a secure p2p channel the group setup phase is simple. User $U_i$ generates a group-wide secret $s_i \in \{0,1\}^l$. These group-wide secrets are used to generate the ephemeral encryption keys used for secure group messages. $U_i$ also generates a signing key $sk_i$ and a corresponding verification key $vk_i$ for message consistency checks. Finally, $U_i$ generates a group view $gv_i$ as a hash of all group participants. This group view is used to check that honest participants agree on the group. The group-wide secrets, verification keys, and group views are shared over the secure p2p channels.

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

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

When $U_i$ receives a broadcast message $c$ from $U_j$, $U_i$ computes $K_j = KDF_3(Sid, (U_j, s_j), (U_1, s_1), \ldots, (U_n, s_n))$ and checks that $c$ is an authenticated encryption with $K_j$. If $c$ does not authenticate it is dropped and the next broadcast is received. Algorithm 3 describes the process. When an authentic $c$ is received a broadcast consistency check is performed (Alg. 1, line 6).

The broadcast consistency check guarantees all participants have received the same $c$ from $U_j$ and that $U_j$ authored the message, then the plaintext $m = ADec_{K_j}(c)$ is displayed. Algorithm 4 describes the broadcast consistency check. To check the consistency of a ciphertext $c$, $U_i$ computes a digest of the ciphertext along with the string "accept" or "reject". This digest

**Algorithm 5** Signature Consistency Check

1: **func** SigConCheck($Sid, U_j, c, sigs, (S, VK, sk_i)$)
2: $S' \leftarrow [], VK' \leftarrow []$
3: $S'[i] \leftarrow \{0,1\}^l$
4: $(sk'_i, VK'[i]) \leftarrow KeyGen()$
5: $share_i \leftarrow$ OptimizeShare$(c, sigs)$
6: $sucess \leftarrow true$
7: **for all** $U_k \in U \setminus \{U_i\}$ **do**
8: SecSend$(Sid, U_k, share_i, S'[i], VK'[i])$
9: $share_k, S'[k], VK'[k] \leftarrow$SecRecv$(Sid, U_k)$
10: **if** $share_k \neq share_i$ **then**
11: $sucess \leftarrow false$
12: Warn the user.
13: Share inconsistent signatures.
14: **if** $success = true$ **then**
15: ConfirmBcast$(U_j, m)$
16: **return** $(S', VK', sk'_i)$

**Algorithm 6** Optimize Share

1: **func** OptimizeShare$(c, (valid, invalid))$
2: $input =$ ""
3: **for all** $(U_j, d_j, sig_j) \in valid$ **do**
4: $input \leftarrow input||(U_j, sig_j)$
5: $h = H_4(input)$
6: $d_{valid} \leftarrow H_3(\text{“}accept\text{”}||c)$
7: **return** $(d_{valid}, h, invalid)$

is then signed with the ephemeral signing key $sk_i$ and shared to all participants over the secure p2p channels. The message is only displayed for $U_i$ if all participants notified $U_i$ that they have "accept"ed the broadcast as valid from $U_j$. If $U_j$ did not author $c$, $U_j$ should "reject" the ciphertext. If there is a malicious participant and they try to convince $U_i$ to display the message and another user to not display the message, the malicious participant will be identified in the next phase.

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

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

#### 3.4.1 Churn

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

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

#### 3.4.2 P2P Secure Channel

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

#### 3.4.3 P2P Channel Setup

Set up of the secure p2p channel uses the NAXOS [17] deniable Authenticated Key Exchange. Since we do not assume a Public Key Infrastructure we simply send the long-term key with the ephemeral key in the first mes-

**Algorithm 7** P2P Secure Channel Setup

1: **func** ChannelSetup($Sid, U_j$)
2: $esk_i \leftarrow \{0,1\}^l$
3: $h_i \leftarrow H_1(esk_i, lsk_i)$
4: Send($Sid, U_j, g^{h_i}, g^{lski}$)
5: $epk_j, lpk_j \leftarrow$ Recv($Sid, U_j$)
6: $K_{ij} \leftarrow KDF_1(epk_j^{lsk_i}, lpk_j^{h_i}, epk_j^{h_i}, U_i, U_j)$
7: $K_{ji} \leftarrow KDF_1(epk_j^{lsk_i}, lpk_j^{h_i}, epk_j^{h_i}, U_j, U_i)$

sage. User identity and key verification are addressed later.

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

After the deniable AKE completes, each user $U_i$ can generate a pair of ephemeral, deniable, and authenticated sending and receiving keys $K_{ij}, K_{ji}$ with every other participant $\forall U_j \in U \setminus \{U_i\}$

### 3.4.4 P2P Key Ratcheting

The keys for the p2p channels are ratcheted forward every message to provide forward and backward secrecy. SYM-GOTR ratchets these keys similar to OTR [2]. When Alice sends a message to Bob she includes a new DH public key. To send the next message she uses her last sent DH share and the DH public key she most recently received from Bob to compute a shared secret and encrypt the message. All p2p protocol messages sent in SYM-GOTR expect a response. This allows SYM-GOTR to use the keys generated during the secure p2p channel setup phase as the first sending and receiving keys to bootstrap the DH key ratcheting. Algorithm 8 describes the functions of the secure p2p channel and key ratcheting. SecSend describes $U_i$ sending $m$ to $U_j$ and SecRecv describes $U_i$ receiving $m$ from $U_j$, $Sid$ is the session id for the SYM-GOTR session. Participants must only maintain a single p2p channel per pair of user per session.

## 3.5 User Authentication

The deniable AKE between all participants does not provide full user authentication. The AKE only guarantees that the remote party knows the long-term secret that corresponds to the long-term public key. To authenticate a remote user and associate that user with their long-term public key we use the Socialist Millionaire Protocol (SMP) [18]. SMP must be executed between every pair of participants with inputs being their long-term public keys and a pre-shared secret. If SMP is successful the pair of users can trust the remote user is the expected party. This trust is linked to the long-term public key of a user and may be applied in future SYM-GOTR sessions. That is the SMP need only be executed once between a pair of users to build trust in a user seen in multiple sessions.

**Algorithm 8** P2P Key Ratchet

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

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

# 4 Security

In this section we discuss the security properties of SYM-GOTR under the threat models described in Section 2.

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

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

The *anonymity preserving* property is provided because SYM-GOTR does not incorporate any information about the transport channels and a participant may generate fresh long-term keys and pseudonyms for every session. To avoid generating fresh long-term keys and pseudonyms the two party secure channel setup may be modified to provide anonymity preservation. As described in Section 8 the first step is to send an ephemeral DH public key and the long-term DH public key in plaintext. To preserve anonymity an unauthenticated DH key exchange may be executed first, then the long-term public key can be hidden within the resulting anonymous encrypted channel.

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

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

SYM-GOTR can be simulated by anyone who possess the long-term public keys of the participants, all encryption and signing keys used for communication are ephemeral. Because of this the sessions have *participant repudiation* and *message repudiation*.

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

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

## 4.1 Confidentiality

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

## 4.2 Message Integrity and Authentication

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

## 4.3 Participant Consistency

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

## 4.4 Forward and Backward Secrecy

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

## 4.5 Participant Repudiation

We describe participant and message repudiation in a similar manner to that of Raimondo, Gennaro, and Krawczyk [19] deniable authentication and key ex-

change. We show that SYM-GOTR is deniable because a session can be simulated to produce a transcript that is identically distributed to a transcript of a real SYM-GOTR session. Since any SYM-GOTR protocol transcript could have been forged by running a simulator, a user appearing in an alleged transcript can plausibly deny their involvement by claiming that the transcript was forged. All simulators are simple extensions to the simulator for the p2p deniable AKE.

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

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

## 4.6 Message Repudiation

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

The simulator is almost identical to the simulator for participant repudiation. The message repudiation simulator executes the participant repudiation simulator then follows the protocol as described in Section 8 to produce a transcript for $\tau$ that is identically distributed to $T$. It is a simple extension to the participant repudiation simulator because all inputs of the message and signature consistency checks are ephemeral values generated during the SYM-GOTR session. Message repudiation is provided for the group conversation since the underlying p2p protocol provides message repudiation.

## 4.7 Message Unlinkability

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

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

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

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

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

### 4.8 Global Transcript

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

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

### 4.9 GOTR Improvements

We quickly discuss the differences between SYM-GOTR, BD-GOTR, and Signal in terms of security properties. Table 1 compares the security properties provided by each protocol.

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

**Participant Repudiation** is not provided by BD-GOTR as described in [4]. The authors claim the p2p channels do not need to be deniable. Only that they are confidential and authenticated channels. If BD-GOTR is implemented with deniable p2p channels then the protocol provides participant repudiation.

**Message Unlinkability** only partially exists in BD-GOTR. The broadcast messages of BD-GOTR contain the digest of the chat transcript thus far. This builds a chain of all the broadcast messages. Once authorship of a single message is proven acceptance of all previous messages is proven as well as authorship of all messages using the same keys.

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

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

## 5 Performance Evaluation

We implemented the SYM-GOTR protocol as a Java library and a plugin for the Jitsi IM client. We measured the performance using the library and a $10/month virtual private server hosted on linode [20]. Our clients ran across a cluster of ten machines each with an Intel i7 3.4GHz Quad-Core processor and 32 GB of RAM. Our server was an ejabberd XMPP server. The server had a 1 Gb network connection and the clients shared a 1 Gb connection. The round trip time between the clients and server was approximately 100 ms. Care needs to be taken when choosing a communication service due

**Table 1.** Conversation and Deniability properties provided by SYM-GOTR, BD-GOTR, and Signal

| | Confidentiality | Integrity | Participant Consistency | Destination Validation | Forward and Backward Secrecy | Anonymity Preserving | Speaker Consistency | Causality Preserving | Global Transcript | Message Unlinkability | Message Repudiation | Participant Repudiation | Asynchronous |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **SYM-GOTR** | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ○ |
| **BD-GOTR** | ● | ● | ○ | ● | ◖ | ● | ● | ● | ◖ | ◖ | ● | ◖ | ○ |
| **(n+1)sec** | ● | ● | ● | ● | ● | ● | ◖ | ◖ | ◖ | ○ | ● | ● | ○ |
| **Signal** | ● | ● | ○ | ● | ◖ | ○ | ○ | ○ | ○ | ● | ● | ● | ● |

●= provides; ◖= partially provides; ○= does not provide

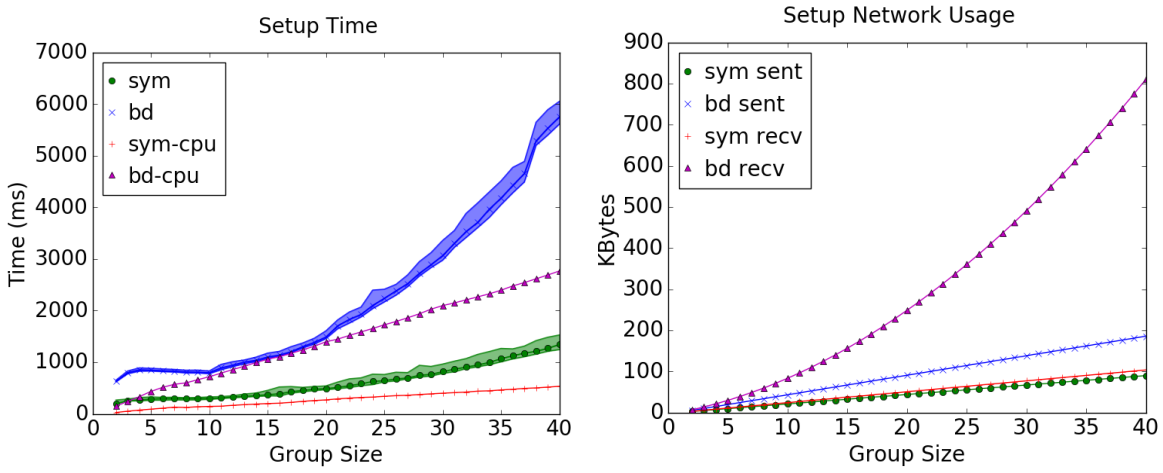


**Fig. 1.** The time (25th, 50th, and 90th percentile) and network traffic to set up a secure chat room with SYM-GOTR and BD-GOTR.

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

For performance analysis we consider groups of a practical size to enforce authentication between all participants. We base our groups sizes on analysis of social graphs in the Facebook social network [21]. We would like to know the average clique size for a set of friends. If the participants did not form a clique they would not be able to pairwise authenticate, negating the security provided by SYM-GOTR. The results of the study did not reveal the clique size but did discuss the median degree of a user and a local clustering coefficient. The median degree of a user is 100 with a local clustering coefficient of 14%. This implies that the maximum clique size for an average user is ∼37.

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

## 5.1 Setup

For each round of secure chat setup we created a Multi-User-Chat (MUC) with $n$ participants then initiated the

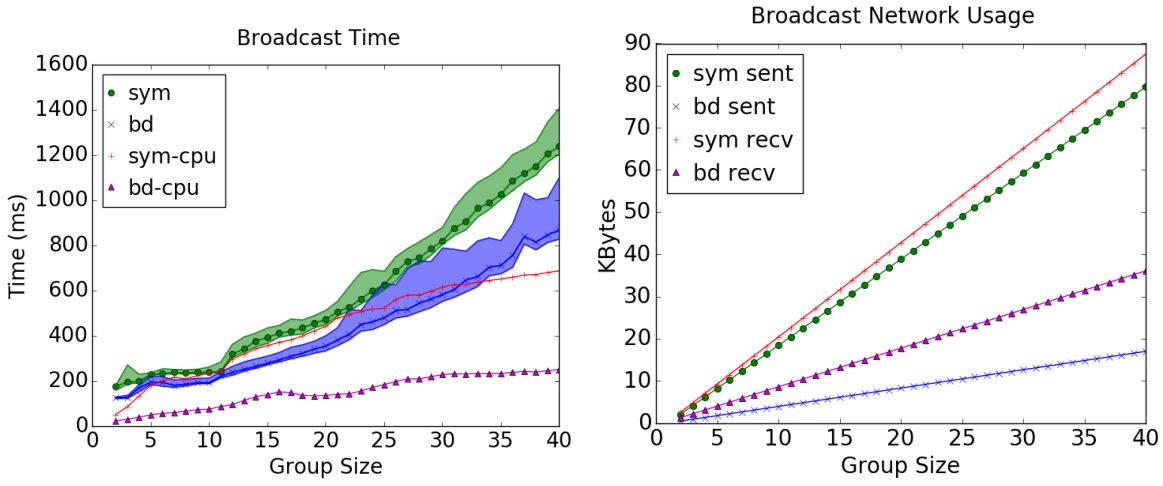


**Fig. 2.** The time (25th, 50th, and 90th percentile) and network traffic to broadcast a message to a secure group chat.

GOTR protocol. After the protocol reached the secure state we shutdown the protocol. We measured the cost as the time it takes from initiating the protocol to reaching the secure state.

Figure 1 compares the cost of setting up a secure chat session with $n$ users. The number of messages a user sends during the setup phase is $O(n)$ for both SYM-GOTR and BD-GOTR. SYM-GOTR sees a significant improvement in time and network traffic as a result of constant sized messages compared to message sizes of $O(n)$ with BD-GOTR.

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

## 5.2 Broadcast

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

Figure 2 shows a group of 30 users takes 820ms and 65 KB of network traffic down to perform a secure broadcast with SYM-GOTR. BD-GOTR is faster for broadcast operations due to a weaker single round consistency check that does not offer message unlinkability or digest accountability. It requires only sending $n-1$ p2p messages where as SYM-GOTR requires $2(n-1)$. The difference is small enough to be worth the additional properties. This difference is also not noticeable to the user due to broadcasts being displayed after the message consistency check. The signature consistency check only warns the user on failures which should not normally occur. The message consistency check of SYM-GOTR and BD-GOTR have the same size and complete in the same time. The network overhead of BD-GOTR comes from broadcast messages being $O(n)$ in size.

## 5.3 CPU Usage

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

### 5.4 Complexity

Due to lack of control of the Signal network we cannot produce a meaningful comparison of SYM-GOTR and Signal but we do provide an asymptotic comparison of each operation. In comparison to Signal, SYM-GOTR requires sending more p2p messages to offer the additional consistency properties. Signal does not attempt to offer group consistency which allows for efficient setup and participant changes. Signal also does not guarantee message consistency or ordering allowing for cheaper message broadcasts. Table 2 shows the asymptotic complexity for each operation under SYM-GOTR, BD-GOTR, and Signal.

### 5.5 Practical Example

To show SYM-GOTR performs well in practice, we simulated one year's worth of IRC meetings from the OpenStack High Availability group [22]. We simulated all meetings from 2016. There are 38 meetings between January 04, 2016 and December 21, 2016. Each meeting had on average 5 participants with 127 messages and lasted 35 minutes. We first replayed the meeting log without SYM-GOTR to record statistics under our network conditions. Then we replayed the meeting logs using SYM-GOTR. SYM-GOTR introduced an average delay of 127 milliseconds to display a message which extended the conversation by 16 seconds on average. This experiment demonstrates a realistic deployment of SYM-GOTR and shows it is practical for day-to-day use.

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

## 6 Discussion

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

### 6.1 Usability

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

### 6.2 Key Verification

Key verification is another challenge without a clear solution. In terms of SYM-GOTR we choose to utilize the Socialist Millionaire Protocol allowing users to authenticate each other using a pre-shared secret. We do not claim this is the best or only mechanism for key verification. Any other technique could be deployed with SYM-GOTR. Another existing system attempting to address this problem is keybase.io which links a users online identity to a long-term public key by connecting it to a users Twitter [23], reddit [24], facebook [25], or a handful of other online accounts. Other solutions include a web-of-trust, Public Key Infrastructure, or a Password Authenticated Key Exchanges (PAKE).

## 7 Conclusion

We propose a solution for secure group Off-The-Record instant messaging. Our protocol offers all of the security properties of existing works, e.g. confidentiality, authentication, repudiation, etc.; along with the additional properties of participant consistency, message unlinka-

**Table 2.** Asymptotic complexity for each operation. The top line is the size of the broadcast message and the bottom is the maximum number of p2p messages sent by an individual. All p2p messages are of constant size. The last two columns represent the computational complexity of the operation.

| | Setup | Broadcast | Add | Remove | Sending | Receiving |
|---|---|---|---|---|---|---|
| **SYM-GOTR** | $\mathbf{0}$<br>$\mathbf{2(n-1)}$ | $\mathbf{1}$<br>$\mathbf{2(n-1)}$ | $\mathbf{0}$<br>$\mathbf{2(n-1)}$ | $\mathbf{0}$<br>$\mathbf{(n-1)}$ | $O(n)$ | $O(n)$ |
| **BD-GOTR** | $\mathbf{8n}$<br>$\mathbf{5(n-1)}$ | $\mathbf{8n}$<br>$\mathbf{(n-1)}$ | $\mathbf{8n}$<br>$\mathbf{5(n-1)}$ | $\mathbf{8n}$<br>$\mathbf{0}$ | $O(n)$ | $O(n)$ |
| **Signal** | $\mathbf{0}$<br>$\mathbf{(n-1)}$ | $\mathbf{0}$<br>$\mathbf{(n-1)}$ | $\mathbf{0}$<br>$\mathbf{(n-1)}$ | $\mathbf{0}$<br>$\mathbf{(n-1)}$ | $O(n)$ | $O(1)$ |

bility and a global transcript. SYM-GOTR achieves this with little overhead in terms for computation and network bandwidth, requiring each user to send $O(n)$ messages of constant size for any operation of the group.

Our implementation of SYM-GOTR is efficient and practical for day-to-day communication. We also provide a Jitsi plugin making secure group communication available now.

# 8 Acknowledgments

We thank Roger Dingledine and Aaron Johnson for key discussion on the system model and properties of secure messaging. We also thank Nik Unger for helping us clarify the presentation of this work and the anonymous reviewers for their helpful feedback. This work was partially supported by the NSF under grant 1314637.

# References

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

[2] 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*, WPES '04, (New York, NY, USA), pp. 77–84, ACM, 2004.

[3] 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*, CCS '09, (New York, NY, USA), pp. 358–368, ACM, 2009.

[4] 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*, WPES '13, (New York, NY, USA), pp. 249–254, ACM, 2013.

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

[6] 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*, WPES '17, (New York, NY, USA), pp. 31–40, ACM, 2017.

[7] 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*, pp. 232–249, IEEE, 2015.

[8] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Advances in cryptology EUROCRYPT'94*, pp. 275–286, Springer, 1994.

[9] M. Marlinspike and T. Perrin, "The x3dh key agreement protocol," 2016.

[10] M. Marlinspike and T. Perrin, "The double ratchet algorithm," 2016.

[11] 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*, pp. 457–472, IEEE, 2016.

[12] 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*, pp. 451–466, IEEE, 2017.

[13] 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.

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

[15] M. Abdalla, C. Chevalier, M. Manulis, and D. Pointcheval, "Flexible group key exchange with on-demand computation of subgroup keys.," *Africacrypt*, vol. 10, pp. 351–368, 2010.

[16] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, pp. 469–491, Sept. 2008.

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

[18] 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*, pp. 41–47,

ACM, 2007.
[19] 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*, pp. 400–409, ACM, 2006.
[20] linode, *linode*. https://linode.com/.
[21] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *arXiv preprint arXiv:1111.4503*, 2011.
[22] *OpenStack IRC meetings*. http://eavesdrop.openstack.org/.
[23] twitter, *twitter*. https://twitter.com/.
[24] reddit, *reddit*. https://reddit.com/.
[25] Facebook, *Facebook*. https://facebook.com/.
[26] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, (London, UK, UK), pp. 453–474, Springer-Verlag, 2001.

# A Proofs

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

## A.1 Assumptions

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

First we assume our authenticated encryption scheme (Encrypt-Then-MAC) is IND-CPA and INT-PTXT secure as shown by Bellare and Chanathip [16].

Figure 3 details the IND-CPA game. An adversary $M$ is said to win the game if $M$ can determine which of

**Fig. 3.** IND-CPA Game

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

**Fig. 4.** INT-PTXT Game

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

**Fig. 5.** NAXOS Game

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

**Fig. 6.** DDH Game

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

**func** FINALIZE($d$)
  **return** $(d = b)$

two chosen plaintext has been encrypted. The advantage of $M$ is defined as $Adv^{IND-CPA}(M) = Pr[Mwins] - \frac{1}{2}$.

Figure 4 details the INT-PTXT game. An adversary $M$ is said to win the game if $M$ can forge a valid ciphertext of a plaintext that has not been queried. The advantage of $M$ is defined as $Adv^{IND-PTXT}(M) = Pr[Mwins]$.

We also assume the NAXOS AKE is secure in the game detailed in [17]. The authors of NAXOS define an extended Canetti-Krawczyk [26] model (eCK) to prove NAXOS secure under a stronger adversary.

Figure 5 describes the NAXOS AKE game. An adversary $M$ is allowed to create multiple sessions and reveal the long-term, ephemeral, and session keys of users. Defining Session IDs is a important part of key agreement models. Our definition is consistent with the eCK model. That is the Session ID of a p2p session is defined as the transcript of messages sent between the parties. This p2p Session ID is different from SYM-GOTR group Session IDs. The adversary wins the game if $M$ can distinguish between a valid NAXOS session key and a random bit string if both parties are not compromised. The advantage of $M$ is defined as $Adv^{NAXOS}(M) = Pr[Mwins] - \frac{1}{2}$.

We assume the Decisional Diffie-Hellman(DDH) problem is hard. Figure 6 describes the DDH problem in terms of a game. An adversary wins the DDH game if given $(g^x, g^y, g^z)$ the adversary can distinguish between $g^z = g^{xy}$ and $g^z \leftarrow^R G$. The advantage of an adversary $M$ is defined as $Adv^{DDH} = Pr[Mwins] - \frac{1}{2}$.

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

**Fig. 7.** SYM-GOTR Game Functions

**func** SETUPSESSION($Sid, U_i, U$)
  Setup session $Sid$ with $U_i$ for group $U$
  **return** $U_i$'s response to session setup

**func** ADDUSER($Sid, U_i, U_j$)
  Add user $U_j$ to session $Sid$ of user $U_i$
  **return** $U_i$'s response to adding $U_j$.

**func** REMOVEUSER($Sid, U_i, U_j$)
  Remove user $U_j$ to session $Sid$ of user $U_i$
  **return** $U_i$'s response to removing $U_j$.

**func** BROADCAST($Sid, U_i, m$)
  $U_i$ sends $m$ securely in session $Sid$.
  **return** $U_i$'s network traffic to send the message

**func** REVEALLONGTERMKEY($U_i$)
  **return** $U_i$'s long-term key

**func** REVEALSESSIONSTATE($Sid, U_i$)
  **return** $U_i$'s session state for session $Sid$.

**func** SEND($U_i, U_j, comm$)
  $U_i$ sends $comm$ to $U_j$
  **return** $U_j$'s response to $comm$

## A.2 Model

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

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

**Fig. 8.** SYM-GOTR Confidentiality Game

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

$U_i$'s current ephemeral secret key and the remote parties ephemeral public key. Figure 7 describes the functions that are consistent between all SYM-GOTR games.

## A.3 Confidentiality

The confidentiality property states a message may only be read by a conversation participant. This is equivalent to the indistinguishable chosen plaintext game. Figure 8 describes the additional functions of the confidentiality game for SYM-GOTR. The TEST function allows the adversary to force $U_i$ to send a test secure broadcast in session $Sid$. The adversary may then guess the bit $b$ by calling FINALIZE. A session is defined as clean if the adversary has not revealed the long-term key or the session state for any participant in the session. The adversary wins if the session is clean and the guess is correct. The advantage of adversary $M$ is defined as $Adv^{conf}(M) = Pr[Mwins] - \frac{1}{2}$.

**Theorem A.1.** *SYM-GOTR is confidential if all hash and key derivation functions are modeled as random oracles. For any confidentiality adversary $M$ that runs in time at most $t$, establishes at most $s$ sessions with at most $w$ users per session and establishes at most $n$ p2p channels. We show that there exists a NAXOS adversary $N$, an IND-CPA adversary $P_0$, and an IND-CPA adversary $B$ such that*

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

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

*Proof.* The adversary $M$ can win the confidentiality game in two ways. They can learn the group sending key of the target user or win the IND-CPA game against our authenticated encryption. Since $KDF_3$ is modeled as a random oracle to learn the group sending key, $M$ must query the random oracle with the same inputs as the sending user. These inputs are shared over the secure p2p channels. For $M$ to learn these inputs $M$ must either brake the NAXOS protocol or win the IND-CPA game against our authenticated encryption. We construct an adversary $N$ that can win the NAXOS game and an adversary $P_0$ that can win the IND-CPA game of the p2p channels and an adversary $B$ that can win the IND-CPA game against the broadcast ciphertext given an adversary $M$ that can win the confidentiality game.

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

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

**Fig. 9.** SYM-GOTR Authentication Game

**func** INITIALIZE($U$)
  $b \leftarrow^R \{0,1\}$
  Initialize PKI for all users in $U$.

**func** FINALIZE()
  **return** *true* if a message $m'$ was displayed by $U'_j$ with author $U'_i$ in session $Sid'$ that was not broadcast with BROADCAST($Sid', U'_i, m'$) and $U'_i$ and $U'_j$ are honest.

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

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

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

## A.4 Integrity and Authentication

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

Figure 9 describes the additional functions of the game that captures the integrity and authentication properties. The game is similar to the game for confidentiality but differs in the BROADCAST function. The adversary can ask user $U_i$ to broadcast a message securely to session $Sid$. The adversary wins the game if an honest user $U'_j$ displays a message $m'$ from an honest user $U'_i$ in session $Sid'$ where BROADCAST($Sid', U'_i, m'$) was not invoked by the adversary. Users are said to be honest if they have not had their long-term key or session state revealed for session $Sid'$. The advantage of adversary $M$ at wining the message authentication game is defined as $Adv^{int}(M) = Pr[Mwins]$.

**Theorem A.2.** *SYM-GOTR provides integrity and message authentication if all hash and key derivation functions are modeled as random oracles. For any authentication adversary $M$ that runs in time at most $t$, establishes at most $s$ SYM-GOTR sessions, and at most $n$ p2p channels, we show that there exists a NAXOS adversary $N$ and an INT-PTXT adversary $P_1$ such that*

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

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

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

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

## A.5 Participant Consistency

The participant consistency property ensures that all users agree on the set of participants in a group conversation. Figure 10 describes the additional functions for the participant consistency game. The participant

**Fig. 10.** SYM-GOTR Participant Consistency Game

**func** INITIALIZE($U$)
$b \leftarrow^R \{0,1\}$
Initialize PKI for all users in $U$.

**func** FINALIZE($Sid, U_i, U_j$)
**return** *true* if $U_i$ and $U_j$ are honest and have completed the setup phase with differing views of participants for session $Sid$.

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

**Theorem A.3.** *SYM-GOTR provides participant consistency if all hash and key derivation functions are modeled as random oracles. For any participant consistency adversary $M$ that runs in time at most $t$, establishes at most $s$ SYM-GOTR sessions, and at most $n$ p2p channels, we show that there exists a NAXOS adversary $N$ and an INT-PTXT adversary $P_1$ such that*

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

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

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

□

## A.6 Perfect Forward Secrecy

Forward Secrecy is the property that any message sent prior to an honest user state reveal is secure against a passive adversary. Figure 11 details the additional functions of the forward secrecy game. All methods return the network traffic generated by the group to perform

**Fig. 11.** SYM-GOTR Perfect Forward Secrecy Game

**func** INITIALIZE($U$)
$b \leftarrow^R \{0,1\}$
Initialize PKI for all users in $U$.

**func** TEST($Sid, U_i, m_0, m_1$)
$U_i$ securely sends $m_b$ in session $Sid$
TEST may only be called once and must be called before REVEALSTATE
**return** All network traffic generated for $U_i$ to send $m_b$ to session $Sid$

**func** REVEALSTATE($Sid, U_i$)
May only be called after TEST
**return** All of $U_i$'s current state for session $Sid$ along with $U_i$'s long-term key

**func** FINALIZE($d$)
**return** $(d = b)$

an operation. An adversary $M$ may request an honest party to send a message with BROADCAST. When ready $M$ may request $U_i$ to send a test message with TEST to a clean session. After sending a test message $M$ may query for the internal state of a user. $M$ must then guess if the challenger sent $m_0$ or $m_1$. An adversary's advantage is defined as $Adv^{pfs}(M) = Pr[Mwins] - \frac{1}{2}$.

**Theorem A.4.** *SYM-GOTR provides forward secrecy if all hash and key derivation functions are modeled as random oracles. For any adversary $M$ that wins the forward secrecy game and runs in time at most $t$, establishes at most $s$ sessions with at most $w$ users per session, establishes at most $n$ p2p sessions, and sends at most $r$ p2p messages, we show that there exists a DDH adversary $D$, an IND-CPA adversary $P_0$, and an IND-CPA adversary $B$ such that*

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

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

*Proof.* For $M$ to win the game he must either compute a previous sending key which requires knowledge of previous p2p channel plaintexts. $M$ may either compute previous p2p channel keys or break the confidentiality of our authenticated encryption scheme, $M$ may also break the confidentiality of the authenticated encryption for broadcast messages.

If $M$ computes previous p2p keys we can construct a challenger $D_0$ for the forward secrecy game that can

**Fig. 12.** SYM-GOTR Backward Secrecy Game

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

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

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

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

## A.7 Backward Secrecy

The backward secrecy property guarantees that after a users state is revealed only the next message is compromised by a passive adversary. Figure 12 describes the additional functions of the backward secrecy game. An adversary $M$ may reveal the state of a user with the REVEALSTATE query. After revealing a state $M$ may issue TEST to instruct $U_i$ to send a message $m$ followed by test message $m_b$ to a clean session. The adversary then guesses if $m_0$ or $m_1$ was sent. The advantage of $M$ is defined as $Adv^{bs}(M) = Pr[Mwins] - \frac{1}{2}$.

**Theorem A.5.** *SYM-GOTR provides backward secrecy if all hash and key derivation functions are modeled as random oracles. For any adversary $M$ that that wins the backward secrecy game and runs in time at most $t$, establishes at most $s$ sessions with at most $w$ users per session, and establishes at most $n$ p2p sessions, we show that there exists a DDH adversary $D_1$, an IND-CPA adversary $P_0$, and an IND-CPA adversary $B$ such that*

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

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

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

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

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