Multi-party Off-the-Record Messaging

Ian Goldberg
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

Berkant Ustaoğlu
NTT Information Sharing Platform Laboratories
Tokyo, Japan
bustaoglu@cryptolounge.net

Matthew D. Van Gundy
Department of Computer Science
University of California, Davis, CA, USA
mdvangundy@ucdavis.edu

Hao Chen
Department of Computer Science
University of California, Davis, CA, USA
hchen@cs.ucdavis.edu

ABSTRACT

Most cryptographic algorithms provide a means for secret and authentic communication. However, under many circumstances, the ability to repudiate messages or deny a conversation is no less important than secrecy and authenticity. For whistleblowers, informants, political dissidents and journalists — to name a few — it is most important to have means for deniable conversation, where electronic communication must mimic face-to-face private meetings. *Off-the-Record Messaging*, proposed in 2004 by Borisov, Goldberg and Brewer, and its subsequent improvements, simulate private two-party meetings. Despite some attempts, the multi-party scenario remains unresolved.

In this paper, we first identify the properties of multi-party private meetings. We illustrate the differences not only between the physical and electronic medium but also between two- and multi-party scenarios, which have important implications for the design of private chatrooms. We then propose a solution to multi-party off-the-record instant messaging that satisfies the above properties. Our solution is also composable with extensions that provide other properties, such as anonymity.

Categories and Subject Descriptors

K.4.1 [**Management of Computing and Information Systems**]: Public Policy Issues—*Privacy*; E.3 [**Data**]: Data Encryption; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*; H.4.3 [**Information Systems Applications**]: Communication Applications—*Computer conferencing, teleconferencing, and videoconferencing*; C.2.2 [**Computer-Communication Protocols**]: Network Protocols—*Applications*

General Terms

Security, Algorithms

CCS'09, November 9–13, 2009, Chicago, Illinois, USA.

Keywords

Privacy, deniability, multi-party, instant messaging

1. MOTIVATION

The Internet presents a novel means of communication — instant messaging (IM), where users can engage in active conversations across great distances. However, IM lacks certain fundamental security properties that a physical private conversation can provide. Impersonation, eavesdropping and information copying are all trivial attacks in IM.

Online communication systems commonly provide three properties: confidentiality, authentication and non-repudiation. Confidentiality and authentication are expected traits of face-to-face conversations, but non-repudiation clashes with the expectations for private communication. Non-repudiation denotes a receiver's ability to *prove* to a third party, possibly a judge, that the sender has authored a message. Although desirable under many circumstances, non-repudiation is the very property journalists, dissidents or informants wish to avoid.

Borisov, Goldberg and Brewer [6] argued that instant messaging should mimic casual conversations. Participants of a casual talk can deny their statements in front of outsiders, and can sometimes deny having taken part in the talk at all. The authors presented a mechanism called *Off-the-Record Messaging* (OTR) that allows two-party private conversations using typical IM protocols. OTR aims to provide confidentiality, authentication, repudiation and forward secrecy, while being relatively simple to employ.

Despite its good design, OTR has limitations, the most important of which is that it can serve only two users. Hence it is not suitable for online multi-party conversations, commonly enjoyed by casual users via Internet Relay Chat (IRC), by open-source software developers, and by businesses that cannot afford confidential meetings across vast distances [4, §2.3]. It is non-trivial to extend OTR to allow for multi-party conversations, as OTR uses cryptographic primitives designed for two parties. For example, OTR uses message authentication codes (MACs) to provide authenticity. While for two parties MACs can provide a deniable authentication mechanism, MACs do not provide origin authentication when used by more than two parties.

Bian, Seker and Topaloglu [4] proposed a method for extending OTR for group conversation. The crux of their solution is to designate one user as the "virtual server". While

this may be feasible under certain circumstances, it deviates from the original OTR goal, which is to mimic private conversations. In private group conversations there is no virtual server responsible for smooth meetings. Moreover, the server becomes an enticing target for malicious parties. Finally, the server has to be *assumed* honest, as a dishonest server could compromise both the confidentiality and the integrity of all messages sent during a chat session.

In this work, we present a multi-party off-the-record protocol (mpOTR), which provides confidentiality, authenticity and deniability for conversations among an arbitrary number of participants. Using our protocol, an ad hoc group of individuals can communicate interactively without the need for a central authority. We identify the important traits of multi-party authentication for users, for messages *and* for chatrooms that share users; that is, we take into account that two or more users may concurrently share more than one chatroom with different peers. When considering privacy properties, we allow malicious insiders and identify their goals. These multi-party chatroom properties present new challenges that were not addressed in previous work.

An OTR transcript reveals that a user at some point communicated with someone. Our mpOTR protocol carries deniability further by allowing the user to deny everything except, by virtue of being part of the system, that they were willing at some point to engage in a conversation. In fact, it is unclear how users can deny the latter at all because by using the Internet, they already indicate their intent and ability to engage with others. In this sense, mpOTR is closer than OTR to simulating deniability in private conversations in the physical world: anyone could take or have taken part in a private conversation, but that person can plausibly deny ever having done so.

1.1 Related work

While not the first to address security in instant messaging, Borisov, Goldberg and Brewer [6] popularized its privacy aspects, partly due to their now-popular open-source plugin. Subsequently, more research was devoted to IM; in fact, the original proposal was found to contain errors [10], which were repaired in a subsequent version of OTR.

On a high level there are two approaches to secure IM. Clients can establish their connections via a centralized server and rely on the server for security and authentication [16]. Alternatively, participants can use shared knowledge to authenticate each other [1]. OTR, which aims to simulate casual conversations, is closer to the second solution.

While there is a wide literature on IM (see [15, §2.1] for an extensive list), little research has focused on the *multi-party* privacy aspects of instant messaging. To our knowledge the only published work with the explicit goal of achieving group off-the-record conversations is the aforementioned result by Bian, Seker and Topaloglu [4]. It has traits of Mannan and Van Oorschot's work on two-party IM [16] in the sense that a designated user acts as a server. In some cases, e.g. the Navy [9], it may be easy to establish a superuser whom everyone trusts, but if the goal is a casual off-the-record chat or users are unwilling to trust each other, agreeing on a server user becomes problematic. We adopt the scenario where all users are equal.

1.2 Outline

In §2 we identify the relevant properties of private meetings and how they apply to IM. §3 describes the different players of our model for private communication; we focus on the different adversaries and their goals. §4 outlines our solution at a high level and shows that we have achieved the goals of private conversations. Due to space limitations we only touch upon the many cryptographic primitives and the formal definitions we use in this paper. We conclude in §5.

2. PRIVATE CHATROOMS

2.1 Confidentiality

In meetings a user $\hat{A}$ is willing to reveal information to chatroom members but not outsiders. Hence chatroom messages need to remain *hidden* from the wider community. In private physical communication, should a new party approach, the participants can "detect" the newcomer and take appropriate actions.

On the Internet eavesdropping cannot be detected as easily; however, there are ways to guard against casual eavesdroppers. Cryptographic algorithms can assure parties that observers looking at the transmitted conversation packets are left in dark about the communicated content. That is, the transcript gives an eavesdropper no additional knowledge above information about lengths of messages and traffic patterns, beyond what the eavesdropper could have deduced without having seen the encrypted messages.

2.2 Entity authentication

In a face-to-face meeting we identify peers via their appearances and physical attributes. By contrast, on the Internet a user proves to another user knowledge of some secret identifying information, a process known as *entity authentication*.

The basic goal of entity authentication is to provide evidence that a peer who presents public key $S_{\hat{B}}$ also holds the corresponding private key $s_{\hat{B}}$. For example, suppose Alice provides a challenge to Bob. If Bob can compute a response that can only be computed by an entity possessing $s_{\hat{B}}$, then Bob successfully authenticates himself to Alice. This type of authentication is limited in the sense that Bob only shows knowledge of $s_{\hat{B}}$. If Bob wants to claim any further credentials like "classmate Bob", then Alice would need additional proofs. Two-party entity authentication has been studied in the setting of OTR by Alexander and Goldberg [1, §4 and §5]; their solution is suitable for pairwise authentication.

The entity authentication goal for mpOTR is to provide a consistent view of chatroom participants: each chat participant should have the same view of the chatroom membership. We achieve this goal by first requiring users to authenticate pairwise to each other. Then users exchange a short message about who they think will take part in the chat. Alternatively, a suitable n-party authentication primitive could be used to authenticate all users to each other simultaneously.

Authentication is challenging. In a centralized approach, if a malicious party successfully authenticates to the server, the security of the whole chatroom is compromised. The problem is more evident when the server itself is malicious. In our approach, parties do not rely on others to perform faithful authentication. All parties check to ensure that no party has been fooled. While we do not provide means to

prevent malicious parties from joining a chat, users can leave a chat if they wish. In other words a malicious party may join a chat with a given set of honest participants only if all the honest participants approve his entrance.

2.3 Origin authentication

Each message has a well-defined source. The goal of origin authentication is to correctly identify the source. First of all a user must be assured that the message is sent from someone who legitimately can author messages in the chatroom. In OTR if Alice is assured that a valid OTR peer sent a message and that peer is not Alice herself, then she knows Bob sent the message and that only she and Bob know the message[1]. In mpOTR if both Bob and Charlie are chat participants, Alice should be able to distinguish messages authored by Bob from messages authored by Charlie. She should also be able to identify origins with respect to chatrooms: if Alice and Charlie are both members of chatrooms $\mathcal{C}_1$ and $\mathcal{C}_2$, then when Alice receives a message from Charlie in $\mathcal{C}_1$, no one should be able to fool her that the message was sent in $\mathcal{C}_2$. In this way Alice is aware of who else sees the message.

Message authentication should be non-repudiable among chat participants in order to allow honest users to relay messages between one another or to expose dishonest users who try to send different messages to different parties. Alice should have the ability to convince Bob, or any other chat member, that a message she accepted from Charlie indeed was sent by Charlie. A word of caution: transferability introduces a subtlety when combined with our deniability requirement. Alice's ability to convince Bob that Charlie authored a message must not allow her to convince Dave, who is not a chat participant, that Charlie authored the message.

2.4 Forward secrecy

The Internet is a public medium: when a typical user sends a data packet, the user has little (if any) idea how the packet will reach its destination. To be on the safe side we assume that the adversary has seen and recorded every transmitted packet for future use. The adversary's ability to see messages motivates encryption; his ability to record messages motivates forward secrecy. Forward secrecy implies that the leakage of static private keys do not reveal the content of past communication. Users achieve forward secrecy by using ephemeral encryption and decryption keys that are securely erased after use and that cannot be recomputed even with the knowledge of static keys.

We separate encryption keys from static keys. Static keys are used to authenticate ephemeral data, which is used to derive short-lived encryption keys. This is a common approach to achieve forward secrecy. Note that this goal is unrelated to deniability: in forward secrecy the user does not aim to refute any message; in fact, the user may not even be aware of the malicious behavior. The goal of the adversary is reading the content of a message as opposed to associating a message with a user.

2.5 Deniability

A casual private meeting leaves no trace[2] after it is dissolved. By contrast, the electronic world typically retains partial information, such as logs for debugging, for future reference, and so on. This contradicts the "no trace" feature of private meetings. As mentioned in the forward secrecy discussion, entities involved in relaying messages may keep a communication record: participants do not and cannot control all copies of messages they send and hence cannot be assured that all copies were securely destroyed. Users can claim the traces are bogus, effectively denying authoring messages. But what is the meaning of "deny" in this context?

Not all deniability definitions are suitable for off-the-record communication. Consider for example the plaintext deniability notion proposed in [8], where the encryption scheme allows a ciphertext author to open the ciphertext into more than one plaintext: Alice wishes to communicate (possibly incriminating) message M_1; she chooses $M_2, \ldots, M_n$ non-incriminating messages and then forms the ciphertext $C = DeniableEncrypt_K(M_1, M_2, \ldots, M_n)$. When challenged to decrypt, Alice can validly decrypt C to any of the alternate messages M_i that she chose when forming C. Even though Alice denies authoring the incriminating plaintext, she implicitly admits to authoring the ciphertext. However, who you speak to may be as incriminating as what you say. Alice might get into deep trouble with her mafia boss by merely admitting that she has spoken with law enforcement, regardless of what she said. She would be in a much better situation if she could claim that she never *authored* the ciphertext, instead of decrypting the ciphertext to an innocuous plaintext and thereby implicitly admitting authorship.

Contrary to the above example, suppose Alice has means of denying all her messages in front of everyone, by arguing that an entity different from herself faked messages coming from her[3]. That is, any message purportedly from Alice could have been authored by Mallory. In that case how could Bob and Charlie have a meaningful conversation with Alice? They have no assurances that messages appearing to come from Alice are indeed hers: Alice's messages can be denied even in front of Bob and Charlie. What we need is a "selective" deniability. We next discuss the selectiveness of deniability in the requirements for multi-party Off-the-Record messaging.

2.5.1 Repudiation

The *fundamental problem of deniability* (FPD) describes the inherent difficulty for a user Alice to repudiate a statement. Let Justin be a judge. Suppose Charlie and Dave come to Justin and accuse Alice of making a statement m. In the best-case scenario for Alice, Charlie and Dave will not be able to provide any evidence that Alice said m, apart from their word. If Alice denies saying m, whom should Justin trust: Charlie and Dave, or Alice? The voices are two to one against Alice, but it is possible that Charlie and Dave are trying to falsely implicate Alice. Justin must decide who to believe based on his evaluation of the trustworthiness of their testimony. Justin's evaluation may be influenced by many hard-to-quantify factors, such as perceived likelihood of the testimony, the number of witnesses in agreement, potential benefits and detriments to the witnesses, etc. Justin may even explicitly favor the testimony of certain witnesses, such as law enforcement officers. In the end, Justin must base his decision on weighing the testimonies rather than on

[1]We assume no "over-the-shoulder" attacks.

[2] If there were no logs, wiretapping, etc.

[3]For example Alice can pick a symmetric encryption key κ encrypt her message with κ, encrypt κ with Bob's public key and send everything to Bob.

physical evidence. In the limit, when n parties accuse Alice of saying m, Alice will have to convince Justin that the other n witnesses are colluding to frame her. We call this the fundamental problem of deniability.

We cannot solve the FPD. The best we can offer is to ensure that Charlie and Dave cannot present any evidence (consisting of an algorithmic proof) that Alice has said m, thereby reducing the question to the FPD. In the online world Charlie and Dave make their claim by presenting a communication transcript. Therefore, we provide Alice with means to argue that Charlie and Dave could have created the transcript without her involvement. As long as Charlie and Dave cannot present an algorithmic proof of Alice's authorship, she can plausibly deny m, so Justin has to rule based on the same factors (e.g., weighing the testimonies rather than on physical evidence) as in the physical world. By this means, we provide comparable levels of repudiation between on-line and face-to-face scenarios.

In §2.3 we alluded to the conflicting goals of message origin authentication and privacy, where the complete deniability example prevents origin authentication. To provide origin authentication, we need a special type of repudiation. Let us look closer at a private communication among Alice, Charlie and Dave. In a face-to-face meeting Charlie and Dave hear what Alice says. This is origin authentication. After the meeting, however, Alice can deny her statements, because, barring recording devices, neither Charlie nor Dave has evidence of Alice's statements. This is the type of repudiation that we aim for.

In contrast to the physical world, on the Internet Charlie can differentiate between Alice and Dave when the three of them are talking and can send them different messages. While it is impossible to guard against such behavior (either due to malicious intent or connection problems), we would like the proof of authorship that Charlie provides to Alice to also convince other chat participants — and no one else — of his authorship. That way, all parties are assured that (1) they can reach a transcript consensus even in the presence of malicious behavior, and (2) all statements within the chat can be denied in front of outside parties. This condition should hold even if Alice and Charlie share more than one chat concurrently or sequentially: all chats must be independent in the sense that if Alice and Charlie share chats $\mathcal{C}_1$ and $\mathcal{C}_2$ any authorship proof Charlie has in $\mathcal{C}_1$ is unacceptable in $\mathcal{C}_2$. In relation to the previous paragraph we note that such authorship proofs should also become invalid at the end of the meeting.

2.5.2 Forgeability

In some cases[4] it is valuable to deny not only having made a statement but also having participated in a meeting. In the physical world Alice can prove she was absent from a meeting by supplying an alibi. On the Internet, however, such an alibi is impossible as Alice can participate in multiple chatrooms simultaneously. Short of an alibi, the next best denial is a design where transcripts allegedly involving Alice can be created without her participation. Although this mechanism would not allow Alice to prove that she was absent from the meeting, it prevents her accusers from proving that she was present at the meeting. A refinement is to design transcripts that can be extended to include users that did not participate, to exclude users who did participate, or both. Effectively, such transcripts will offer little, if any[5], evidence about who participated in it.

For example, suppose Mallory tries to convince Alice that Bob spoke with Dave and Eve by presenting a transcript with participants Bob, Dave, and Eve. Ideally, forgeability would allow Bob to argue that Mallory fabricated the transcript even though Mallory is an outsider with respect to the transcript.

2.5.3 Malleability

In §2.5.2 we dealt with forging who participated in a communication transcript. With malleability we address the issue of the transcript content. Ideally, the transcript should be malleable in the sense that given a transcript $\mathbb{T}^1$ and a message m_1 that belongs to $\mathbb{T}^1$, it is possible to obtain a transcript $\mathbb{T}^2$, where message m_1 is substituted with message m_2. Along with forgeability this approach provides a strong case for users who wish to deny statements and involvement in chat meetings. For accusers, transcripts with this level of flexible modification provide little convincing evidence, even in the event of confidentiality breaches.

2.6 Anonymity and pseudonymity

While in our current work anonymity is not the main goal, we desire that our solution preserves anonymity. This includes, but is not restricted to, not writing users' identities on the wire. While we do not explicitly address it, users may wish to use our protocol over a transport protocol that provides pseudonymity. If they do so, it would be unacceptable if our protocol deanonymizes users to adversaries on the network. We do, however, use anonymity-like techniques to achieve a subset of our deniability goals.

3. THREAT MODEL

3.1 Players

We will first introduce the different players and their relations with each other. The set of users, denoted by $\mathcal{U}$, is a collection of entities that are willing to participate in multiparty meetings. Honest parties, denoted by $\hat{A}$, $\hat{B}$, $\hat{C}$, ... follow the specifications faithfully; these parties are referred to as Alice, Bob, Charlie, ... Dishonest parties deviate from the prescribed protocol. Each party $\hat{A}$ has an associated long-lived static public-private key pair $(S_{\hat{A}}, s_{\hat{A}})$. We assume that the associated public key for each party is known to all other parties. (These associations can be communicated via an out-of-band mechanism or through authentication protocols as in [1].) A subset $\mathcal{P}$ of users can come together and form a chatroom $\mathcal{C}$; each member of $\mathcal{P}$ is called a *participant* of $\mathcal{C}$. While honest users follow the protocol specifications, they may observe behavior that is not protocol compliant due to either network failures, intentional malicious behavior by other parties, or both.

In addition to users that take part in the conversation we have three types of adversaries: (i) a security adversary, denoted by $\mathcal{O}$; (ii) a consensus adversary, $\mathcal{T}$; and (iii) a privacy adversary, $\mathcal{M}$. The last player in the system, the judge $\mathcal{J}$, does not interact with users but only with adversaries, in particular with $\mathcal{M}$. We will see his purpose when discussing the adversaries' goals.

[4]Police informants, for example.

[5]If the plaintext is recovered, the writing style or statements made may reveal the author's identity.

3.2 Goals

Honest users wish to have on-line chats that emulate face-to-face meetings. It is the presence of the adversaries that necessitates cryptographic measures to ensure confidentiality and privacy.

3.2.1 Security adversary

The goal of the security adversary is to read messages that he is not entitled to. Let $T_{\mathcal{C}_1} = \left\{ \mathbb{T}^{\hat{X}}_{\mathcal{C}_1} \mid \hat{X} \in \mathcal{P} \right\}$ be a collection of transcripts resulting from a chat $\mathcal{C}_1$ with set of chat participants $\mathcal{P}$, such that no user in $\mathcal{P}$ revealed private[6] information to, or collaborated with, the security adversary $\mathcal{O}$ prior to the completion of $\mathcal{C}_1$. Suppose also that for each honest participant $\hat{A}$, who owns $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1} \in T_{\mathcal{C}_1}$[7], and for each honest participant $\hat{B}$, who owns $\mathbb{T}^{\hat{B}}_{\mathcal{C}_1} \in T_{\mathcal{C}_1}$, $\hat{A}$ and $\hat{B}$ have consistent view of the messages and participants. We say that $\mathcal{O}$ is successful if $\mathcal{O}$ can read at least one message in some $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$ without obtaining the message from a user $\hat{B}$ who owns $\mathbb{T}^{\hat{B}}_{\mathcal{C}_1}$.

A few remarks on $\mathcal{O}$'s goals are in order. The security adversary can control communication channels and observe the actions of any number of users in $\mathcal{P}$, learn messages that they broadcast in other chatrooms, and start chatroom sessions with them via proxy users. All these actions can take place before, during or after $\mathcal{C}_1$. However, $\mathcal{O}$ is allowed neither to ask for static private information of any user in $\mathcal{P}$ before the completion of $\mathcal{C}_1$ nor to take part in $\mathcal{C}_1$ via a proxy user. The adversary may ask an honest user to send messages to $\mathcal{C}_1$, but should still be unable to decide if or when his request is honored. Essentially, $\mathcal{O}$ aims to impersonate an honest user during key agreement or to read messages in a chatroom that consists only of honest users. $\mathcal{O}$'s capabilities are similar to the standard notion of indistinguishability under chosen-plaintext attack for encryption schemes [2].

3.2.2 Consensus adversary

For details on communication in asynchronous networks and how users can keep transcripts we refer the reader to Reardon et. al. [18]. We first explain the meaning of consensus, which relates to what Alice thinks about her and Bob's view of past messages. We say that $\hat{A}$ reaches consensus on $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$ with $\hat{B}$ if $\hat{A}$ believes that $\hat{B}$ admits having transcript $\mathbb{T}^{\hat{B}}_{\mathcal{C}_2}$[8] such that:

1. $\mathcal{C}_1$ and $\mathcal{C}_2$ have the same set of participants;
2. $\mathcal{C}_1$ and $\mathcal{C}_2$ are the same chatroom instance;
3. $\mathbb{T}^{\hat{B}}_{\mathcal{C}_2}$ has the same set of messages as $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$;
4. $\mathbb{T}^{\hat{B}}_{\mathcal{C}_2}$ and $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$ agree on each message's origin.

At the end of the meeting (or at predefined intermediate stages) honest users attempt to reach consensus with each other about the current transcript. Our consensus definition allows the possibility that Alice reaches a consensus with Bob but Bob does not reach consensus with Alice: for example if either Bob or Alice goes offline due to network failure before protocol completion. We also allow the application to interpret "same set of messages" appropriately for its setting. For instance, the importance of message delivery order may vary by application.

The goal of the consensus adversary $\mathcal{T}$ is to get an honest user Alice to reach consensus with another honest user Bob on a transcript $\mathbb{T}^{\hat{A}}_{\mathcal{C}}$, while at least one consensus condition is violated; that is, $\mathcal{T}$ wins if (honest) Alice believes that (honest) Bob has a transcript matching hers (in the above sense), but in fact Bob does not have such a transcript. Note that while Alice and Bob are honest users there is no restriction on the remaining chat members — they may even be $\mathcal{T}$-controlled, which is an improvement over KleeQ [18] where all parties are assumed honest. Resilience against $\mathcal{T}$ implies that users cannot be forced to have different views of exchanged messages and no messages can be injected on behalf of honest users without being detected.

Our consensus definition captures both the standard notions of entity and origin authentication and the adversary's abilities to make conflicting statements to different participants in the same chat session (as described in §2.5.1) as well as drop, duplicate, reorder, and replay messages from other chat sessions.

3.2.3 Privacy adversary

The goal of the privacy adversary $\mathcal{M}$ is to create a transcript $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$ to convince the judge $\mathcal{J}$ that $\hat{A}$ took part in $\mathcal{C}_1$ and/or read and/or authored messages in $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$. The only restriction is that $\mathcal{J}$ is not directly involved in $\mathcal{C}_1$. This is perhaps the hardest adversary to guard against as $\mathcal{M}$ has few restrictions: $\mathcal{M}$ can interact in advance with $\mathcal{J}$ before $\mathcal{C}_1$ is established and, by taking part in $\mathcal{C}_1$, can obtain consensus with respect to $\hat{A}$. Furthermore, the judge can force $\hat{A}$ as well as all other participants to reveal their long-term secrets. If under such a powerful adversary and judge combination, Alice can still plausibly deny $\mathbb{T}^{\hat{A}}_{\mathcal{C}_1}$, then many of her privacy concerns can be assuaged. Our privacy requirement is stronger than the settings presented in [11, 12] because $\mathcal{J}$ must not be able to distinguish between Alice's transcripts and forgeries even if $\mathcal{J}$ gets Alice's long-term secrets.

3.3 Local views

We complete the section by saying that from an honest user's perspective it is unclear a priori whether an honestly behaving user has no malicious intent. Conversely, if a user observes deviation from the protocol the user cannot always distinguish a true malicious player from network instability. (Certain deviations, such as a participant making conflicting statements, can be identified, however.)

4. SOLUTION DESIGN

The mpOTR protocol follows a straightforward construction. To ensure confidentiality among the participants $\mathcal{P}_1$ of a chatroom $\mathcal{C}_1$ the participants derive a shared encryption key gk_1. Messages sent to the chatroom are encrypted under gk_1 to ensure that only members of $\mathcal{P}_1$ can read them. To provide message authentication, each participant $\hat{A} \in \mathcal{P}_1$ generates an ephemeral signature keypair $(E_{\hat{A},1}, e_{\hat{A},1})$ to be used only in the current session. Each message sent by $\hat{A}$ will be signed under $\hat{A}$'s ephemeral signing key for the current session $e_{\hat{A},1}$. Participants exchange ephemeral public keys for the current session $E_{\hat{X},1}$ $(\hat{X} \in \mathcal{P}_1)$ amongst themselves

[6] Either static private keys or $\mathcal{C}_1$-related information.
[7] That is, user $\hat{A}$ did take part in $\mathcal{C}_1$, and in particular $\hat{A} \in \mathcal{P}$.
[8] By admitting this transcript $\hat{B}$ admits taking part in $\mathcal{C}_2$.

Algorithm 1: $Initiate(\mathcal{P}_i)$ — initiate a chatroom $\mathcal{C}_i$ among the participants $\mathcal{P}_i$ in the context of party $\hat{X}$. On successful completion, all participants hold a shared encryption key, ephemeral public signature keys for all other participants, and have authenticated all other participants and protocol parameters.

Input: chat participants $\mathcal{P}_i$
Output: an encryption key gk_i, session id sid_i, ephemeral public signature keys of all other participants $\{E_{\hat{Y},i} \mid \hat{Y} \in \mathcal{P}_i\}$

`// Initialize variables`
$sid_i \leftarrow \perp$, $Sent \leftarrow \emptyset$, $Received \leftarrow \emptyset$;
$consensus_{\hat{Y}} \leftarrow$ false for all $\hat{Y} \in \mathcal{P}_i$;
$sid_i \leftarrow SessionID(\mathcal{P}_i)$;

`// Exchange ephemeral signature keys`
$(result, R) \xleftarrow{\$} DSKE(sid_i, \mathcal{P}_i)$;
if $result =$ accept **then**
  foreach $(E, \hat{Y}) \in R$ **do** $E_{\hat{Y},i} \leftarrow E$;
else
  abort session initiation;

`// Agree on shared encryption key`
$gk_i \xleftarrow{\$} GKA(\mathcal{P}_i, R)$;
if $gk_i = \perp$ **then** abort session initiation;
$Attest()$;

in a deniable fashion. At the end of the session, each participant publishes their ephemeral private key $e_{\hat{X},1}$ ($\hat{X} \in \mathcal{P}_1$) for the current session to allow third parties to modify and extend the chatroom transcript.

The mpOTR protocol lifecycle consists of three phases: setup, communication, and shutdown. In the setup phase all chatroom participants negotiate any protocol parameters, derive a shared key, generate and exchange ephemeral signing keys, and explicitly authenticate all protocol parameters including the set of chatroom members and the binding between participants and their ephemeral signature keys. During the communication phase, participants can send confidential, authenticated, deniable messages to the chatroom. To end a chatroom session, the protocol enters the shutdown phase. In the shutdown phase, each participant determines if he has reached consensus with each other participant, after which participants publish their ephemeral private keys.

4.1 Network communication

Our constructions assume the existence of the following network primitives, typically provided by application layer protocols, such as IM or IRC. To free our constructions from undue dependence on the underlying network layer, we limit ourselves to the following primitives:

- $Broadcast(M)$ — sends message M over the broadcast channel where it can be $Receive()$'ed by all other participants. In the absence of a broadcast medium, like an IRC channel, $Broadcast()$ can be simulated by sending M directly to each other participant in $\mathcal{P}$.
- $Send(\hat{A}, M)$ — sends message M addressed explicitly to $\hat{A}$. The network may send M to $\hat{A}$ directly (point-to-point) or via broadcast (during broadcast, all the honest participants other than $\hat{A}$ ignore M).

Algorithm 2: $SessionID(\mathcal{P}_i)$ — invoked in the context of party $\hat{X}$, the algorithm returns a unique (with high probability) chatroom identifier for the set $\mathcal{P}_i$ upon successful completion.

Input: chat participants $\mathcal{P}_i$
Output: session id sid_i
$x_{\hat{X}} \xleftarrow{\$} \{0,1\}^k$;
$Broadcast(x_{\hat{X}})$;
$Outstanding \leftarrow \mathcal{P}_i \setminus \{\hat{X}\}$;
while $Outstanding \neq \emptyset$ **do**
  $(\hat{Y}, x) \leftarrow Receive()$;
  if $\hat{Y} \in Outstanding$ **then**
    $x_{\hat{Y}} \leftarrow x$;
    $Outstanding \leftarrow Outstanding \setminus \{\hat{Y}\}$;
return $H(\mathcal{P}_i, x_{\hat{Y}_1}, x_{\hat{Y}_2}, \ldots)$ for all $\hat{Y}_j \in \mathcal{P}_i$ ordered lexically;

- $Receive() \rightarrow (\hat{A}, M)$ — returns any waiting message M received by the party that invokes $Receive()$ along with M's alleged author $\hat{A}$.
- $Receive(\hat{A}) \rightarrow M$ — waits until a message is received from $\hat{A}$ and returns that message (M).

To simplify our protocols, we make the following assumptions. $Broadcast()$ and $Send()$ are non-blocking. If message M from party $\hat{A}$ arrives at $\hat{B}$ before $\hat{B}$ executes a $Receive()$ call, M is buffered at $\hat{B}$ and will be returned upon some subsequent invocation of $Receive()$ by $\hat{B}$. $Receive()$ calls block until a message is available. If the current instance of some party $\hat{A}$ has assigned a value to its session id (sid_i) variable, $Receive()$ will only return messages M that were sent from an instance of some party $\hat{B}$ that has set its session id to the same value (i.e. $Broadcast()$, $Send()$, and $Receive()$ multiplex on sid_i).

Recall that, with all network access, the adversary has control over message delivery and may modify or deliver messages at will. Thus, when $Receive()$ invoked by $\hat{B}$ returns $(\hat{A}, M)$, $\hat{A}$ may have invoked either $Broadcast(M)$ or $Send(\hat{B}, M)$, or the adversary may have sent M under the identity of $\hat{A}$.

In the following discussion, we abuse notation in that a single value M may be replaced by a tuple $(x_1, x_2, \ldots)$. This indicates that the values $x_1, x_2, \ldots$ have been encoded into a single message using an unambiguous encoding scheme. Upon receiving such a message, if parsing fails, the protocol assigns the distinguished value $\perp$ to each of $x_1, x_2, \ldots$.

4.2 Setup phase

The setup phase is responsible for deriving the shared encryption key gk_i for the chatroom $\mathcal{C}_i$, performing entity authentication, facilitating exchange of ephemeral signing keys $E_{\hat{X},i}$ ($\hat{X} \in \mathcal{P}_i$), and ensuring forward secrecy and deniability. In the following, we assume that the participants have negotiated the participant set $\mathcal{P}_i$ for the chatroom instance $\mathcal{C}_i$ via an unspecified, unauthenticated means. Each participant in the protocol executes the $Initiate(\mathcal{P}_i)$ algorithm with their view of $\mathcal{P}_i$. The $Initiate()$ procedure will only succeed if every other party in $\mathcal{P}_i$ completes its portion of the protocol correctly and has the same view of $\mathcal{P}_i$.

First, the participants calculate a globally unique session id sid_i for the current session. Each participant $\hat{X}$ chooses a random value $x_{\hat{X}}$ of suitable length k and broadcasts it to the other participants. Each participant calculates sid_i by hashing the participant set $\mathcal{P}_i$ with the random contributions of all other participants. Under the assumption that $H(\cdot)$ is a collision-resistant hash function, sid_i is globally unique with high probability as long as at least one participant behaves honestly. If the adversary has manipulated the random contributions (x), it will be detected during the *Attest*() algorithm executed at the end of *Initiate*() when sid_i and any other unauthenticated parameters $params_i$ are authenticated.

$\hat{X}$ then enters into a deniable signature key exchange protocol with the other participants of $\mathcal{P}_i$ ($DSKE(sid_i, \mathcal{P}_i)$) to generate an ephemeral signature key pair $(E_{\hat{X},i}, e_{\hat{X},i})$ and to exchange ephemeral public keys with the other parties in $\mathcal{P}_i$. $\hat{X}$ will use $e_{\hat{X},i}$ to sign messages sent to the chatroom $\mathcal{C}_i$. $\hat{X}$ generates a new signing key pair in each session so that there is no transferable proof that he has signed any messages in the chat transcript. However, the other participants must know that $E_{\hat{X},i}$ will be $\hat{X}$'s public signature key for this session.

Next, *Initiate*() invokes a group key agreement protocol that uses the set of participants $\mathcal{P}_i$ and their ephemeral signature keys to derive a fresh encryption key gk_i shared by all members of $\mathcal{P}_i$. If any stage of the group key agreement fails, $GKA()$ returns $\perp$ and *Initiate*() aborts.

Finally, all participants execute the *Attest*() algorithm to ensure that they agree on all lower-level protocol parameters that they may have negotiated before invoking *Initiate*(). Each participant takes a hash over all of these values and the session identifier, and uses the *AuthSend*() and *AuthReceive*() procedures (see §4.3) to transmit the hash value to all the other participants in a confidential, authenticated manner. Each participant then ensures that the value sent by all other participants matches their own. Upon successfully completing *Attest*(), the participants have fully initialized the chat session and can enter the communication phase.

When users wish to join or leave a chatroom, the protocol shuts down the current session and then calls *Initiate*() with the new set of participants to initialize a new chat session. We handle joins and leaves in this manner because we currently determine transcript consensus during the shutdown phase and must derive a new encryption key before a membership change can take place. Client software can shut down and initialize a new session behind the scenes so that users need only decide whether or not they accept the proposed membership change.

4.2.1 Deniable Signature Key Exchange (DSKE)

In our construction, we use a sub-protocol that we call Deniable Signature Key Exchange. Deniable Signature Key Exchange allows the participants in a session to exchange ephemeral signature keys with each other in a deniable fashion. A participant will use his ephemeral signature key to sign messages during one session. Because it is ephemeral (used only in one session), the private key can be published at the end of the session to permit transcript modification. Because the key exchange protocol is deniable, there is no transferable proof that any party has committed to use any given key.

Algorithm 3: *Attest*() — authenticate (previously) unauthenticated protocol parameters for the current session in the context of party $\hat{X}$.

Input: session id sid_i, chat participant set $\mathcal{P}_i$, negotiated protocol parameters $params_i$
Output: aborts protocol initiation on failure
$M \leftarrow H(sid_i, params_i)$;
$AuthSend(M)$;
$Outstanding \leftarrow \mathcal{P}_i \setminus \{\hat{X}\}$;
while $Outstanding \neq \emptyset$ **do**
  $(\hat{Y}, M_Y) \leftarrow AuthReceive()$;
  if $M_Y \neq M$ **then**
    abort the session;
  else
    $Outstanding \leftarrow Outstanding \setminus \{\hat{Y}\}$;

Deniable Signature Key Exchange is an n-party interactive protocol operating over common inputs: sid — a fresh session identifier, and $\mathcal{P}$ — the set of participants for the session identified by sid. When the protocol concludes, each participant outputs a termination condition (either accept or reject) and a set R relating the members of $\mathcal{P}$ to public signature keys (e.g. $R = \{(E_{\hat{A}}, \hat{A}), (E_{\hat{B}}, \hat{B}), \ldots\}$).

Two-party signature key exchange.

The goal of two party signature exchange (Algorithm 4) is to allow Alice and Bob to exchange signing key pairs $(E_{\hat{A}}, e_{\hat{A}})$ and $(E_{\hat{B}}, e_{\hat{B}})$, respectively, such that: (i) Alice is assured that Bob knows $e_{\hat{B}}$ corresponding to $E_{\hat{B}}$; (ii) Alice is assured that Bob, if honest, will not associate $E \neq E_{\hat{A}}$ with Alice; and (iii) Alice is assured that after completing the exchange Bob cannot prove to a third party Charlie (without Alice's consent) that Alice has associated herself with $E_{\hat{A}}$ and knows $e_{\hat{A}}$. The same conditions must hold for Bob with respect to Alice.

Algorithm 4: $AuthUser(sid, \hat{B}, E_{\hat{A}}, e_{\hat{A}})$ — obtain and associate $\hat{B}$ with a signing key pair, and send $\hat{B}$ one's own signing key $E_{\hat{A}}$.

Input: session id sid, peer identity $\hat{B}$, signature pair $(E_{\hat{A}}, e_{\hat{A}})$
Output: associate $\hat{B}$ with $E_{\hat{B}}$ or $\perp$
$k, k_m \leftarrow denAKE(\hat{A}, \hat{B})$;
$Send(\hat{B}, SymMacEnc_k^{k_m}(E_{\hat{A}}, sid, \hat{A}, \hat{B}))$;
$(E_{\hat{B}}, sid', \hat{B}', \hat{A}') \leftarrow SymDec_k^{k_m}(Receive(\hat{B}))$;
$Send(\hat{B}, SymEnc_k^{k_m}(Sign_{e_{\hat{A}}}(E_{\hat{B}}, sid, \hat{A}, \hat{B}))$;
$m \leftarrow SymDec_k^{k_m}(Receive(\hat{B}))$;
if $(sid' = sid) \wedge (\hat{A}' = \hat{A}) \wedge (\hat{B}' = \hat{B})$
  $\wedge Verify(m, E_{\hat{B}}, (E_{\hat{A}}, sid', \hat{B}, \hat{A})) == 1$ **then**
  return $\hat{B}, E_{\hat{B}}$;
else
  return $\perp$;

The signature exchange proceeds as follows: first Alice and Bob run a deniable two-party key agreement protocol $denAKE(\hat{A}, \hat{B})$ to derive a shared secret. Using symmetric key techniques they exchange signature keys that they in-

tend to use in the subsequent chatroom. Finally, both users sign the ephemeral public key of their peer along with both Alice's and Bob's identities under their ephemeral keys for the current session.

Assume that $denAKE()$ is a secure, deniable authenticated key agreement protocol. Let $SymMacEnc_k^{k_m}()$ be an algorithm that encrypts and authenticates messages with the symmetric keys k and k_m, and let $Sign()$ be an existentially unforgeable signature scheme. The protocol $denAKE()$ provides keying material only to Bob and Alice. Hence, they are assured about each other's identity. Since Bob signs Alice's ephemeral public signature key she is assured that the signature that Bob generated is not a replay from other sessions and that Bob knows the corresponding ephemeral private key. Bob is assured that $E_{\hat{A}}$ is connected with Alice because he did not generate $E_{\hat{A}}$ and his peer has to know k and k_m to complete the protocol. Since $denAKE()$ is secure, the only party other than Bob that could have computed k and k_m is Alice. Likewise, Alice is assured that an honest Bob will not associate $E \neq E_{\hat{A}}$ with her because Bob will only associate an ephemeral key with Alice if Bob received it through a secure channel that only Bob and Alice share. The only proof that Bob has about communicating with Alice is the $denAKE()$ transcript. Since $denAKE()$ is deniable Alice can argue that any transcript between herself and Bob was created without her contribution; in other words, Bob's view cannot associate Alice to $E_{\hat{A}}$ unless Alice admits the association. Thus Algorithm 4 achieves the three conditions that we described.

We conclude by saying that that $E_{\hat{A}}$ and $E_{\hat{B}}$ are "pseudonyms" that Alice and Bob exchange. As long as the corresponding private keys are not leaked each one of them is assured about the identity behind the pseudonym and messages signed with the keys, but cannot prove to a third party the relation between the pseudonym and a real entity. Furthermore, any party Mallory can create a fake pseudonym for Alice or Bob.

Multi-party signature key exchange.

We extend the two-party algorithm to the multi-party setting. In particular, given a set of participants $\mathcal{P}$, every pair of users in $\mathcal{P}$ runs Algorithm 4. For a given identifier sid, Alice uses the same key pair $(E_{\hat{A}}, e_{\hat{A}})$.

The next stage is for participants to assure each other of the consistency of the association table that they build. Let $(E_{\hat{A}}, \hat{A}), \ldots, (E_{\hat{X}}, \hat{X})$, be the association table built by Alice, lexicographically ordered on the signing keys. Each user computes a hash of that table, signs the hash with her ephemeral signing key and sends it to the rest of the participants[9]. As a result each participant is assured that the remaining members have the same view about the association table. Note that the exchange does not reveal anything about the table. The set of participants can collaborate to introduce "non-existent" users into the chatroom. In other words, if agreed, a set of users can create a transcript that allegedly involves an absent user Alice. Such a transcript can be indistinguishable from a transcript where Alice did take part.

[9]This can be incorporated into $Attest()$

Deniable AKE.

By a "secure" key agreement protocol we mean the standard indistinguishable from random key notion introduced by Bellare and Rogaway [3]. However, we are concerned with malicious insiders so protocols that meet models as introduced in [17] are more suitable for our needs, since they allow the adversary to adaptively introduce malicious parties to the system.

In contrast to secure key exchange, "deniable" key exchange has not been as widely studied. On one hand there is a formal definition, presented in [11, Definition 1], that relies on the fact that a receiver's view can be simulated. The authors prove the deniability of SKEME [13] according to their definition. However, there are some pitfalls related to leaking static secrets and the deniability of SKEME. If the judge $\mathcal{J}$ has access to the static secrets of the alleged participants, $\mathcal{J}$ can distinguish between authentic and simulated transcripts. Therefore, SKEME does not meet our privacy requirement (§3.2.3).

On the other hand, Diffie-Hellman variants like MQV [14] provide plausible deniability as outlined in [7]. The shared key is derived only from public values, so a peer can plausibly argue that he did not take part in the key agreement. Additionally, implicitly authenticated protocols that meet the security definition of [17] appear to meet our privacy notion. This allows any such protocol to be used in settings where the participants may expose their long-lived secrets without sacrificing deniability.

As suggested in [7], one can achieve improved deniability via self-signed certificates that users authenticate. At the extreme it is possible for users *not* to have any static secrets but to authenticate each other via out-of-band means for every session. While such a solution is possible, its usability is questionable. We accept that users cannot convincingly deny their static secrets in order to achieve a less complicated protocol. The users can still deny taking part in any fixed chatroom and the content of messages that they sent.

4.2.2 Group Key Agreement

Assuming that users successfully run the signature exchange protocol, they can proceed to establish group keys. Given sid and an association table from sid users run a typical key group key agreement protocol to derive a shared secret key gk to ensure that they have a means for confidential communication. Note that when the group key agreement is based on the session-specific signature keys, Alice can deny knowing gk by arguing that she took no part in the protocol — recall there is no proof of her relation with $E_{\hat{A}}$.

4.2.3 Properties

Alice can plausibly argue that she did not take part in a chat because it is possible to create a protocol transcript that includes users who did not actually take part in the chat. This can happen if all participants collaborate to introduce such non-existent users. In the limit, this allows a single party to create a transcript involving any number of other non-cooperating parties. With an appropriate deniable signature key exchange, the forging party need not even be a member of $\mathcal{P}$. The issue of modifying existing messages in a transcript will be addressed in the shutdown phase.

4.3 Communication phase

During the communication phase, chat participants may exchange confidential messages with the assurance of origin authentication — that they have received messages unchanged from their purported authors. Given a chatroom instance $\mathcal{C}_1$ with participant set $\mathcal{P}_1$, we use the group key gk_1, ephemeral public keys of the participants $E_{\hat{X},1}$ $(\hat{X} \in \mathcal{P}_1)$ and session id sid_1 for $\mathcal{C}_1$ in a standard Encrypt-then-Sign construction to provide authenticated encryption [2] for messages sent to the chatroom. Algorithms *AuthSend*() and *AuthReceive*() give our construction.

Algorithm 5: *AuthSend*(M) — broadcast message M authenticated under party $\hat{X}$'s ephemeral signing key to chatroom $\mathcal{C}_i$.

Input: message M, session id sid_i, shared chat encryption key gk_i, ephemeral private signing key $e_{\hat{X},i}$
Output: authenticated encryption of M is broadcast to chat channel

$Sent \leftarrow Sent \cup \{(\hat{X}, M)\}$;
$C \leftarrow Encrypt_{gk_i}(M)$, $\sigma \leftarrow Sign_{e_{\hat{X},i}}(\ (sid_i, C)\ )$;
$Broadcast(\ (sid_i, C, \sigma)\ )$;

Algorithm 6: *AuthReceive*() — attempt to receive an authenticated message from $\mathcal{C}_i$, return the sender and plaintext on success, sender and $\perp$ on failure.

Input: session id sid_i, shared chat encryption key gk_i, ephemeral public signature keys of other participants $\{E_{\hat{Y},i} \mid \hat{Y} \in \mathcal{P}_i\}$
Output: sender identity $\hat{Y}$ and plaintext message M, or $\perp$ on failure

$(\hat{Y}, (sid, C, \sigma)) \leftarrow Receive()$;
if $sid \neq sid_i \vee Verify(\ (sid, C), \sigma, E_{\hat{Y},i}) \neq 1$ **then**
  return $(\hat{Y}, \perp)$; `// Bad signature or session id`
$M \leftarrow Decrypt_{gk_i}(C)$; `// returns` $\perp$ `on failure`
if $M \neq \perp$ **then**
  $Received \leftarrow Received \cup \{(\hat{Y}, M)\}$;
return $(\hat{Y}, M)$;

When $\hat{A}$ sends a message to the chatroom, she first encrypts the message under the shared key of the chatroom gk_1 to ensure that only legitimate chat participants $(\mathcal{P}_1)$ will be able to read it. Then, $\hat{A}$ signs the session id sid_1 and ciphertext using his ephemeral signing key $e_{\hat{A},1}$ and broadcasts the session id, ciphertext, and signature to the network allowing all recipients to verify that $\hat{A}$ has sent the ciphertext to $\mathcal{C}_1$ and that it has been received unmodified.

We assume that *Encrypt*() and *Decrypt*() constitute a secure encryption scheme indistinguishable under chosen plaintext attack (IND-CPA) [2], $GKA()$ is a secure group key agreement scheme [5], $DSKE()$ is secure as described in §4.2.1, *Sign*() and $Verify()$ constitute an existentially unforgeable signature scheme, and session identifiers are globally unique. Under these assumptions, we can transform any confidentiality adversary $\mathcal{O}$ (§3.2.1) into a successful adversary against the encryption scheme, the group key agreement that derives the encryption key gk_i, or the deniable signature key exchange scheme that distributes the ephemeral signature keys that are used to authenticate messages sent during the group key agreement. Therefore, under the assumption that the above protocols are secure, our full scheme is secure against any confidentiality adversary $\mathcal{O}$.

Likewise, the security of $DSKE()$ and the signature scheme imply that the adversary cannot forge messages that are acceptable by *AuthReceive*(). Including the globally unique session id in the message to be signed prevents a message from one session from being replayed in another session. We can also achieve this by deriving a chatroom-specific MAC key from gk_i, which verifies that messages are designated for sid_i. While a consensus adversary $\mathcal{T}$ is unable to successfully forge messages, she can attempt to break consensus by dropping or duplicating messages or by sending different correctly authenticated messages from a corrupted participant to disjoint subsets of honest participants. E.g. $\mathcal{T}$ uses corrupted participant $\hat{C}$ to send M_1 to $\hat{X}$ and M_2 to $\hat{Y}$ where $M_1 \neq M_2$. We address these last three threats during the shutdown phase.

4.4 Shutdown phase

When the application determines that there are no outstanding in-flight messages between participants and that the chat session should be ended, it invokes the *Shutdown*() algorithm. *Shutdown*() is responsible for determining whether all participants have reached a consensus and for publishing the ephemeral signature key generated for the current session. All in-flight messages must have been delivered before invoking shutdown for two reasons: (i) in-flight messages will cause unnecessary failure to reach consensus; and (ii) publication of the ephemeral signature key would allow the adversary to modify any in-flight messages.

To establish consensus, the local party $(\hat{X})$ takes a digest over all the messages authored by $\hat{X}$ during the chat session and sends it along with the distinguished message "shutdown" to all the other parties. This message allows each other participant to verify that his transcript of received messages from $\hat{X}$ is identical to $\hat{X}$'s view. To ensure that out-of-order message delivery does not affect this digest, the messages are taken in lexical order. Note, however, that should messages include a suitable order fingerprint, then lexical order can coincide with delivery or creation order, hence our ordering is unrestrictive. For example, if each message starts with an author identifier and a sequence number, lexical order will group messages by author in the order that they were created.

Shutdown() collects the digests published by all the other participants. It then calculates the digests of $\hat{X}$'s transcripts of the messages received from each other party, combines these digests into a single digest, publishes the combined digest, and collects the combined digests from all the other parties. At this point, $\hat{X}$ determines if it has reached consensus with each of the other parties on the session transcript.

Since at the setup phase parties confirmed their views of chat participants and *sid* of the chat, all transcripts already agree on the set of participants and the chat instance. As argued in §4.3, the only remaining way for an adversary to break consensus is to force different messages in the transcript. The consensus adversary does not (yet) have the signature keys hence he is still not able to inject new messages or impersonate honest users; his only freedom is the hash function that we assume collision and preimage resistant.

Algorithm 7: *Shutdown*() — called in the context of party $\hat{X}$ when the application determines that the session should be shut down. Determines if consensus has been reached with other participants and publishes ephemeral signing key.

Input: all sent messages *Sent*, all received messages *Received*, participant set $\mathcal{P}_i$, session id sid_i, ephemeral signing key $e_{\hat{X},i}$

Output: $consensus_{\hat{Y}}$ values indicating if consensus has been reached for each party $\hat{Y}$, publishes private ephemeral signing key for current session $e_{\hat{X},i}$

```
// Publish digest of sent messages
```
Let $((\hat{X}, M_1^{\hat{X}}), (\hat{X}, M_2^{\hat{X}}), \ldots) = Sent$ in lexical order;
$h_{\hat{X}} \leftarrow H(M_1^{\hat{X}}, M_2^{\hat{X}}, \ldots)$;
$AuthSend($ ("shutdown", $h_{\hat{X}}$));
```
// Collect digests of others' transcripts
// and calculate digest of our view
```
$Outstanding \leftarrow \mathcal{P}_i \setminus \{\hat{X}\}$;
while $Outstanding \neq \emptyset$ **do**
  $(\hat{Y}, (\text{"shutdown"}, h'_{\hat{Y}})) \leftarrow AuthReceive()$;
  Let $(M_1^{\hat{Y}}, M_2^{\hat{Y}}, \ldots) = \{M \mid (\hat{Y}, M) \in Received\}$ in lexical order;
  $h_{\hat{Y}} \leftarrow H(M_1^{\hat{Y}}, M_2^{\hat{Y}}, \ldots)$;
  $Outstanding \leftarrow Outstanding \setminus \{\hat{Y}\}$;

```
// Publish digest of full chat
```
Let $(\hat{Y}_1, \hat{Y}_2, \ldots) = \mathcal{P}_i$ in lexical order;
$h \leftarrow H(h_{\hat{Y}_1}, h_{\hat{Y}_2}, \ldots)$;
$AuthSend($ ("digest", h));
```
// Determine consensus
```
$Outstanding \leftarrow \mathcal{P}_i \setminus \{\hat{X}\}$;
while $Outstanding \neq \emptyset$ **do**
  $(\hat{Y}, (M, h')) \leftarrow AuthReceive()$;
  if $M =$ *"digest"* $\wedge\, \hat{Y} \in Outstanding$ **then**
    $consensus_{\hat{Y}} \leftarrow h = h'$;
    $Outstanding \leftarrow Outstanding \setminus \{\hat{Y}\}$;

```
// Verify that nobody's listening
```
$AuthSend($*"end"*$)$;
$Outstanding \leftarrow \mathcal{P}_i \setminus \{\hat{X}\}$;
while $Outstanding \neq \emptyset$ **do**
  $(\hat{Y}, M) \leftarrow AuthReceive()$;
  if $M \neq$ *"end"* **then**
    return;
  else
    $Outstanding \leftarrow Outstanding \setminus \{\hat{Y}\}$;

```
// Publish ephemeral signing key
```
$Broadcast($ $(sid_i, \hat{X}, e_{\hat{X},i})$);

Thus chat participants obtain assurances about consistency — they reach pairwise consensus in the sense of §3.2.2.

The consensus approach adopted above is crude, as it does not attempt to remedy any consensus errors, and it only determines consensus at the very end of the chat session. We adopt this simple approach to allow the network layer to freely choose consensus-ensuring algorithms. The network could provide totally ordered multicast or KleeQ-like algorithms optimized for the broadcast medium. Whatever approach is chosen, our protocol can detect any violations of reliable delivery at the mpOTR level. Furthermore, the signatures used to authenticate messages are transferable within the chatroom since all members have the correct association between the chatroom-specific signature keys and the entities behind the keys. Therefore the protocol can identify malicious users, since an honest party Alice has transferable proofs to convince any other honest party about the origin of the messages that she received. Thus she can prove that she did not modify or inject messages on behalf of other users. Likewise, she can update her transcript with messages that she failed to receive. Ultimately, honest users can agree on a transcript that is the union of all the messages that have reached at least one honest user. Although we have chosen the simple approach above for its clarity, approaches that ensure consensus incrementally throughout the chat session are possible and useful.

After exchanging all the values, *Shutdown*() sends the distinguished message "end" indicating $\hat{X}$ will no longer send any authenticated messages. Once $\hat{X}$ has received the "end" message from each other participant, $\hat{X}$ knows that all participants have determined their consensus values and will no longer accept messages from $\hat{X}$. This allows $\hat{X}$ to publish his ephemeral signing key to permit modifying the chat transcript.

Publishing the ephemeral signing key is a delicate issue. If the key is published too soon, the adversary could use the ephemeral signing key to impersonate the current party to others. Therefore, the protocol only publishes the ephemeral signing key at the end of *Shutdown*() if it can verify that all other parties have agreed that they have determined their consensus values and will only publish their keys or end the session. The adversary can trivially prevent any party $\hat{X}$ from publishing its signing key by preventing the delivery of even one of the "end" messages. However, this is not a problem. The protocol is deniable even without publishing the ephemeral signing keys. Therefore, we gladly trade the deniability benefits gained by allowing malleability for ensuring that the adversary will not be able to impersonate $\hat{X}$. However, if parties do publish their ephemeral signing keys then the existing transcripts can be tweaked. This a posteriori publication of signing keys allows for a user Alice who accepts a relation between her chatroom signing key and herself to argue that the messages in the transcript are bogus. Indeed the adversary could inject and/or delete messages on behalf of Alice's ephemeral signing key, since all secret information has been made public.

5. CONCLUSION

Our proposed framework for multi-party Off-the-Record communication does not depend on a central server; instead we developed a model that mimics a typical private meeting where each user authenticates the other participants for himself. We identified three main goals for mpOTR: confiden-

tiality, consensus and repudiation. We achieve confidentiality via standard cryptographic measures. Consensus is based on unforgeable signatures. Repudiation is based on a user's ability to disassociate from the signing key pair. The crucial step in our solution is the distribution of chatroom-specific signature keys, which become the authentication mechanism during the chat. The deniability is a consequence of the forward secrecy and deniability of the key agreement protocol that is used to establish authentic, confidential, deniable channels between pairs of parties.

We are currently implementing and improving the efficiency of mpOTR. Since the setup phase is crucial for consensus and deniability we opted for a relatively slow solution that requires pairwise interaction. It is natural to look for a more efficient protocol for authentic, deniable, confidential exchange of signing keys. We also believe that a complete formalization and verification of our model will improve our understanding and will help us select suitable primitives and analyze mpOTR's interaction with anonymity-providing protocols and networks.

Acknowledgments

We would like to thank Matt Franklin, Matt Bishop, Zhendong Su, and Phillip Rogaway for their feedback during the early stages of this research. We would also like to thank the anonymous reviewers for their helpful comments. This research is based upon work supported by the National Science Foundation under Grant No 0831547 (Van Gundy and Chen), MITACS (Goldberg) and NSERC (Goldberg).

6. REFERENCES

[1] C. Alexander and I. Goldberg. Improved User Authentication in Off-The-Record Messaging. In P. Ning and T. Yu, editors, *WPES'07: Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 41–47, New York, NY, USA, 2007. ACM.

[2] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *LNCS*, New York, NY, USA, Dec. 2000. Springer-Verlag.

[3] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *LNCS*, pages 232–249, Santa Barbara, CA, USA, 1993. Springer Verlag. Full version available at `http://www.cs.ucdavis.edu/~rogaway/papers/eakd-abstract.html`.

[4] J. Bian, R. Seker, and U. Topaloglu. Off-the-Record Instant Messaging for Group Conversation. In *IRI '07: Proceedings of Information Reuse and Integration*, pages 79–84. IEEE Computer Society, 2007.

[5] J.-M. Bohli, M. I. G. Vasco, and R. Steinwandt. Secure Group Key Establishment Revisited. Cryptology ePrint Archive, Report 2005/395, 2005. `http://eprint.iacr.org/2005/395`.

[6] N. Borisov, I. Goldberg, and E. Brewer. Off-the-record communication, or, why not to use PGP. In V. Atluri, P. Syverson, and S. D. C. di Vimercati, editors, *WPES '04: Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, pages 77–84, New York, NY, USA, 2004. ACM.

[7] C. Boyd, W. Mao, and K. G. Paterson. Key Agreement Using Statically Keyed Authenticators. In B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, editors, *Security Protocols, 11th International Workshop, Revised Selected Papers*, volume 3364 of *LNCS*, pages 255–271, Berlin, Germany, 2005. Springer Verlag.

[8] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable Encryption. In B. S. Kaliski, Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *LNCS*, pages 90–104, Santa Barbara, CA, USA, 1997. Springer Verlag.

[9] S. M. Cherry. IM means business. *IEEE Spectrum*, 38:28–32, November 2002.

[10] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Secure Off-the-Record Messaging. In V. Atluri, S. D. C. di Vimercati, and R. Dingledine, editors, *WPES'05: Proceedings of the 2005 ACM workshop on Privacy in electronic society*, pages 81–89, New York, NY, USA, 2005. ACM.

[11] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Deniable Authentication and Key Exchange. In R. N. Wright, S. De Capitani di Vimercati, and V. Shmatikov, editors, *CCS 2006: Proceedings of the 13th ACM Conference on Computer and Communications security*, pages 400–409, New York, NY, USA, 2006. ACM.

[12] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. *Journal of the ACM*, 51(6):851–898, 2004. `http://www.wisdom.weizmann.ac.il/%7Enaor/PAPERS/time.ps`.

[13] H. Krawczyk. SKEME: A Versatile Secure Key Exchange Mechanism for Internet. In *SNDSS '96: Proceedings of the 1996 Symposium on Network and Distributed System Security*, pages 114–127, 1996.

[14] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An Efficient Protocol for Authenticated Key Agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[15] M. Mannan. Secure Public Instant Messaging. Master's thesis, Carleton University, Ottawa, Canada, August 2005.

[16] M. Mannan and P. C. van Oorschot. A Protocol for Secure Public Instant Messaging. In G. Di Crescenzo and A. Rubin, editors, *Financial Cryptography and Data Security – FC 2006*, volume 4107 of *LNCS*, pages 20–35, Anguilla, British West Indies, 2006. Springer Verlag. Full version available at `http://www.scs.carleton.ca/research/tech_reports/2006/download/TR-06-01.pdf`.

[17] A. Menezes and B. Ustaoglu. Comparing the pre- and post-specified peer models for key agreement. In Y. Mu, W. Susilo, and J. Seberry, editors, *Information Security and Privacy – ACISP 2008*, volume 5107 of *LNCS*, pages 53–68. Springer, 2008.

[18] J. Reardon, A. Kligman, B. Agala, and I. Goldberg. KleeQ: Asynchronous Key Management for Dynamic Ad-Hoc Networks. Technical Report CACR 2007-03, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, 2007.