Modeling Stateful Communication

Chen-Da Liu-Zhang[1], Christopher Portmann[2], and Guilherme Rito[3] *

[1] Lucerne University of Applied Sciences and Arts & Web3 Foundation
chendaliu@gmail.com
[2] Concordium Zurich, Switzerland
cp@concordium.com
[3] Ruhr-Universität Bochum, Germany
guilherme.teixeirarito@rub.de

Abstract. The most basic property one expects (and, often, assumes) from a group chat is, perhaps arguably, *consistency*. Suppose Alice, Bob and Charlie are having a chat, and Alice reads a "Hi" from Charlie. Alice may naturally expect Bob to see the same "Hi" from Charlie when he looks at his phone. Indeed, it is natural that group members expect having the same view of a chat (i.e. messages, set of participants, and other chat-related information) as any other up-to-date member.

This paper puts forth an abstraction for stateful group communication of this basic guarantee. Our abstraction, *Chat Sessions*, is defined in the Constructive Cryptography (CC) framework (Maurer and Renner, ICS '11) and captures the consistency guarantees achievable in asynchronous settings when one makes no party-honesty assumptions: anyone, *including group members*, may be malicious. We *construct*, *extend* and *use* Chat Sessions:

- Our *construction* is fully decentralized, does not incur extra interaction between chat participants (other than what is inherent from sending chat messages) and liveness depends *solely* on chat messages being delivered.
- We *extend* Chat Sessions to provide *authenticity*, *confidentiality*, *anonymity* and *off-the-record* guarantees, and show our construction trivially preserves each of these properties from the underlying communication channels.
- We *use* Chat Sessions to construct *UatChat*: a simple but well-featured messaging application. UatChat also inherits each of Chat Sessions' additional properties mentioned above. This means when it is instantiated with the application semantics given in (Liu-Zhang et al., ePrint 2025/204) we obtain the first fully Off-The-Record (group) messaging application.

* Part of work done while author was at ETH Zurich.

Table of Contents

1 Introduction 5
  1.1 A note on the messaging literature 6
2 Overview 8
  2.1 Chat Sessions Abstraction 8
  2.2 Building on Chat Sessions: UatChat 11
3 Preliminaries 12
  3.1 (Simplified) Constructive Cryptography 12
  3.2 Modeling Access Control via Repositories 14
    3.2.1 Repository Label Notation 14
    3.2.2 Modeling an Asynchronous Network 15
4 Chat Sessions 15
  4.1 Overview 15
  4.2 Helper Functions 16
  4.3 Real World 18
  4.4 Ideal Chat Sessions 18
    4.4.1 Policy Requirements. 19
  4.5 Security Analysis 20
5 UatChat: A Decentralized Messenger 20
  5.1 The Unanimous Policy $\mathfrak{U}$ 21
  5.2 Defining UatChat 21
    5.2.1 Constructing UatChat. 23
6 The Modularity of **ChatSessions**[$\mathfrak{P}$] 23
  6.1 Application Semantics of Multi-Designated Receiver Signed Public Key Encryption [34] 26
    6.1.1 Application Semantics for MDRS-PKE [34] 26
  6.2 Extending **ChatSessions**[$\mathfrak{P}$] to Provide Extra Guarantees 28
    6.2.1 Authenticity. 28
    6.2.2 Off-The-Record. 29
    6.2.3 Confidentiality and Anonymity. 31
  6.3 Uatchat 32
A Definition of UatChatProt (Algorithm 16) 37
B Proof of Theorem 1 37
  B.1 Proof Structure 37
  B.2 Helper Propositions 38
  B.3 Hybrid Sequence 40
  B.4 Proofs of Helper Propositions 59
    B.4.1 Proof of Proposition 1. 59
    B.4.2 Proof of Proposition 2. 60
    B.4.3 Proof of Proposition 3. 60
    B.4.4 Proof of Proposition 4. 60
    B.4.5 Proof of Proposition 5. 61

B.4.6 Proof of Proposition 6. 62
B.4.7 Proof of Proposition 7. 62
B.4.8 Proof of Proposition 8. 62
B.4.9 Proof of Proposition 9. 62
B.4.10 Proof of Proposition 10. 63
B.4.11 Proof of Proposition 11. 65
B.4.12 Proof of Proposition 12. 66
B.4.13 Proof of Proposition 13. 66
B.4.14 Proof of Proposition 14. 66
C "Zoomed-in" version of Figure 1 66

Note

Some contributions in earlier versions of this paper are in a new paper: [34].

1 Introduction

Motivation. Cryptography's most common use is secure communication—e.g. Alice can use encryption to hide the contents of emails she sends to Bob (confidentiality) and sign them to assure Bob she is the sender (authenticity). While one typically considers *stateless* security guarantees—for example a channel that Alice can use to send messages securely to Bob—one should also consider *stateful* ones—for example to capture the security properties of an interactive conversation between Alice, Bob and their friends where participation is dynamic: new parties can join the conversation and existing ones can leave. A natural application of such stateful guarantees are messengers.

(Stateless) Consistency. For applications that allow sending messages to multiple recipients—e.g. emails and group chats—a natural property one desires is *consistency*: every recipient should get the same messages. For example, when Alice receives an email and sees Bob is also a recipient, it is natural for her to expect Bob also gets the same email. For the case of an honest sender, consistency follows from a scheme's correctness. If the sender is dishonest, however, it does not. Surprisingly:

– neither broadcast encryption nor multi-recipient public key encryption primitives provide this guarantee [25,32,15,20]; and
– neither the MLS standard [14] nor email encryption systems (e.g. PGP or S/MIME) provide this guarantee. (See Section 1.1 for a discussion on MLS.)

A recent line of work initiated by Damgård et al. in [23] has focused on defining and constructing cryptographic schemes with this (stateless) consistency property—that all recipients should obtain the same messages [23,37,38,20,34].

Stateful Consistency. Analogously to emails, in the context of group chats it is also natural for the members of a chat to expect seeing the same messages as other members. More generally, it is natural for them to expect having the same view of the conversation as other participants—which includes not only chat messages, but also any information related to the state of the conversation.[4] Of course, achieving such stateful consistency property is trivial if one assumes a trusted delivery service[5] and builds on communication channels providing stateless consistency. However, trusting a delivery service to be available and correctly follow its protocol means that (at least) a chat's liveness depends on this server.[6] Needless to say, it is desirable that a messenger's liveness does

[4] For messaging applications like Signal [1] and Whatsapp [2] this also includes, for example, who are the current group administrators.

[5] By trusted delivery service, we mean one that is always available and which correctly follows its protocol.

[6] More critically, MLS relies on the delivery service (DS) to enforce chat policy permissions. Quoting [14, Section 16.11, "Additional Policy Enforcement"]: "For example, MLS enables any participant to add or remove members of a group; a DS could enforce a policy that only certain members are allowed to perform these operations.".

not rely on such assumption.[7] This paper focuses on defining a meaningful and achievable stateful consistency notion in such a fully decentralized setting.

Our contribution. We introduce *Chat Sessions*: an idealized abstraction of stateful consistency. Very roughly, it guarantees that for each chat there is an efficiently computable function *mapping* any set $\mathcal{S}$ of chat messages/operations and any party P, to the view that P has of the given chat when the set of messages it has sent or received is (exactly) $\mathcal{S}$. Crucially, $\mathcal{S}$ is a set: a party's view of a chat is independent of the order with which it received the chat messages/operations. This prevents group-splitting attacks because every member of a group chat can compute the view of any other member who has sent or received the messages in $\mathcal{S}$.

Chat Sessions is parameterized by (and enforces) a permissions policy $\mathfrak{P}$ that defines what operations parties have the right to perform in a given chat state.

In addition to defining Chat Sessions in the Constructive Cryptography (CC) framework [39,36], we also show how to construct it, extend it and use it:

- Our construction fully decentralized—a group chat's liveness does not depend on neither a functioning delivery service nor the honesty of any of the chat's members; it only depends on chat messages being delivered—and enforces a (parameter) access control policy $\mathfrak{P}$—guaranteeing chat members only perform the operations for which they have permissions.[8]
- Following a modeling technique introduced in [34], we extend Chat Sessions to provide authenticity, confidentiality, anonymity and off-the-record guarantees. We prove our construction inherits each of these properties from the underlying communication channels.
- We use Chat Sessions to construct Uatchat: a simple but realistic messaging application that (analogously) inherits additional security properties provided by our abstraction.

Finally, we note that in recent work [34], Liu-Zhang et al. give the first composable semantics for Multi-Designated Receiver Signed Public Key Encryption (MDRS-PKE) schemes [38,20], and show that Maurer et al.'s MDRS-PKE construction provides them. The application semantics they define for these schemes match the repositories upon which Chat Sessions (and Uatchat) are built. Put together with their results, Uatchat is the first fully off-the-record (group) messaging application.

1.1 A note on the messaging literature

Current works on secure messaging focus on Forward Secrecy (FS) and Post-Compromise Security (PCS) notions [21,16,5,29,6,8,7,31,10,3,9,17,19,4] which aim at providing rather strong confidentiality guarantees in settings where users'

[7] Indeed recent works have focused on eliminating this assumption [44,45,11,22].
[8] Policy enforcement also does not depend on any such assumptions.

devices may get compromised. These notions capture the confidentiality of messages exchanged prior to a compromise (FS), and after group members' devices are no longer compromised (PCS). Being confidentiality guarantees, however, both FS and PCS are only achievable when receivers are honest [35, Theorem 1]. As we now explain, this is the setting considered in the messaging literature—which, despite significant progress on tolerating stronger and stronger attacks, still assumes all group members correctly follow the prescribed protocol.

For example, in [8], Alwen et al. study the security of Continuous Group Key Agreement (CGKA) schemes in the presence of active adversaries—which are allowed to use information obtained from state exposure of users' devices to inject messages on honest users' behalf (thus impersonating them)—and in follow-up work [10], Alwen et al. weaken some of the assumptions made in [8] (in particular it assumes a standard public key infrastructure as opposed to assuming a stronger Key-Registration with Knowledge) to capture so-called insider security. But yet, as explained in [10, Section 3.1], their notions (as the ones from [8]) do not prevent the so-called *group-splitting attacks*, which consist of partitioning a group into subgroups in such a way that members of a partition cannot communicate members of different partitions; this is an attack because group members are unaware of the split [8,10,11].[9]

Another common assumption in the messaging literature is that of an additional external party that is trusted with providing a total ordering on the messages sent by group-members [21,16,6,8,7,3,9]—the delivery service. While this additional party is generally untrusted—i.e. confidentiality is guaranteed even if this party is corrupted—the availability (or liveness) of a chat still depends on this party's honesty [21,16,6,8,7,31,10,3,9]. Even worse, a malicious delivery service can also perform group-split attacks—even in works that consider malicious insiders such as [8,10,11].[10] This has naturally motivated the study of fully decentralized protocols (e.g. [44,45,11,22]) that do not rely on a delivery service, thus avoiding such group-splitting/fork attacks. However, these protocols still do not prevent malicious parties from performing group-split attacks [45,11,22].

It should be noted that early work ([16]) has explicitly identified consistency as a desirable property for MLS [16, Section 5, "Provably Consistent Group Operations"]; in follow up work, Devigne et al. introduce efficient zero-knowledge protocols aimed at providing consistency [24]. However their work does not provide a definition of consistency, and proving their protocols prevent such attacks is an open problem. Unfortunately, since these early works, consistency has received very limited attention from the messaging community.

[9] MLS leaves the responsibility of handling such attacks to the upper application, as explicitly mentioned in [14, Section 16.12, "Group Fragmentation by Malicious Insiders"].

[10] These works explicitly allow an adversary to mount such attacks.

2 Overview

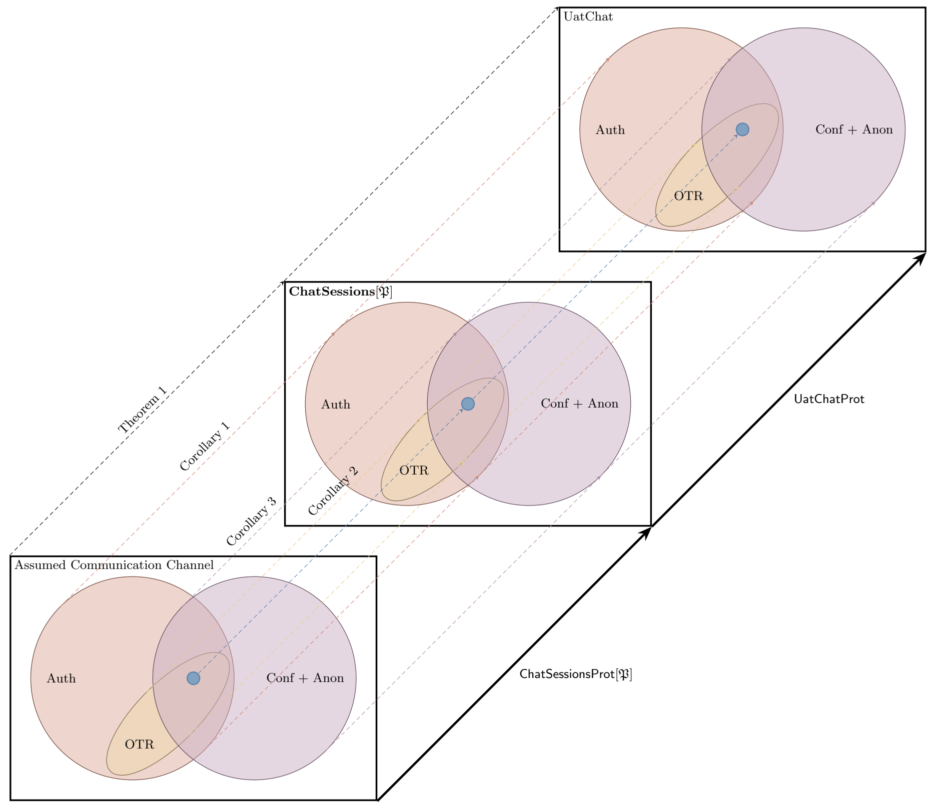


Fig. 1. Visualization of modularity-related results. In the figure, "Auth" denotes Authenticity, "OTR" Off-The-Record and "Conf + Anon" Confidentiality and Anonymity. Each box's Venn diagram illustrates these additional security guarantees. The $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ and $\mathsf{UatChatProt}$ constructions preserve each of these guarantees: if the underlying assumed resources provides any of these guarantees, then the constructed resource (**ChatSessions**[$\mathfrak{P}$] and UatChat, respectively) provides them too. The blue circle in the intersection of all the additional properties denotes the guarantees provided by the application semantics for Multi-Designated Receiver Signed Public Key Encryption schemes, as recently defined and proven in [34]. A "zoomed-in" version of this illustration is in Appendix, Section C.

2.1 Chat Sessions Abstraction

As already explained, Chat Sessions is a type of stateful consistency notion. We now overview some aspects of its definition.

Message ordering. Achieving a total order on messages is rather expensive, either in terms of the resources needed to get it (e.g. extra interaction between parties to reach consensus) or in terms of additionally trusting a third party to provide this ordering [3,4,22,44,45,11]. Instead, we only rely on the causal consistency explicitly given by messages: each message m acknowledges a set of prior ones, and a party can only see m if it already sees all the ancestors m is acknowledging. Each chat session consists of a directed graph (digraph) $\mathcal{G} = (V, E)$, where each node $u \in V$ corresponds to a command (e.g. chat message) issued by a group member, and each edge $(u, v) \in E$ corresponds to an ordering between commands—in this case meaning that $v \in V$ should only become visible after u is visible.[11]

Intuition for stateful consistency. For each chat session there is a unique digraph $\mathcal{G}_{\text{Global}} \coloneqq (V, E)$[12]—where each node $v \in V$ defines a sender S, a vector of receivers $\vec{V}$, a command `cmd` and a set of acknowledgments Acks (i.e. prior nodes on which v depends). A bit more formally, the set of nodes V actually defines $\mathcal{G}_{\text{Global}}$, as E is simply the union of the edges incoming to each node $u \in V$ and the edges incoming to a node are defined its set of acknowledgments. Consider two parties P_i and P_j and let $\mathcal{G}_i \coloneqq (V_i, E_i)$ and $\mathcal{G}_j \coloneqq (V_j, E_j)$ be the subgraphs of $\mathcal{G}_{\text{Global}}$ induced by P_i's and P_j's views, respectively. Consistency means, on one hand, that for each node in $V_i \cap V_j$, both parties (i.e. P_i and P_j) see the same sender S, vector of receivers $\vec{V}$, command `cmd` and set of acknowledgments Acks, and on the other hand, that P_i knows which nodes—among the currently visible ones V_i—will become visible to P_j when they are delivered (and vice-versa for P_j).

Arbitrary management policies. Chat sessions does not fix any particular group management policy, and instead is parameterized by one which it enforces. A chat management policy $\mathfrak{P}$ defines two predicates—IsRoot and IsValid—defining the commands each party can issue; chat sessions then guarantees that parties only issue commands they are allowed to (according to $\mathfrak{P}$). This is possible due to the consistency guarantees of chat sessions: every honest party can check the validity of a command, so disallowed commands can be simply ignored.

Related work: In existing literature it is standard to consider a fixed policy supporting operations for party addition, removal and key updates[13] for which all parties have permissions [7,3,10,13,43]. While if all of a group's members are honest such policy is general enough to implement other arbitrary policies [8], trusting parties to behave honestly goes against the very nature of a permissions policy [13,43]. In recent work, Bálbas et al. pave way to the study of group chat administration in the presence of malicious (but non-administrator) group members [13], where they consider a policy that closely matches the ones implemented

[11] A seemingly related concept is that of *history graphs* [7]. However, history graphs were introduced as a means of simplifying security definitions, while in our case honest parties actually get to see each chat sessions' graphs.

[12] These are not formally graphs, as we will see.

[13] Key update operations are key to PCFS guarantees.

in applications like Signal [1] and WhatsApp [2]. While [13] takes a significant step forward in that group members are not trusted to follow a policy (in particular by disallowing non-administrators from performing administrator-reserved operations), it still relies on administrators being honest (e.g. no guarantees are given when a dishonest administrator has its administration rights revoked).[14]

Fine-grained modularity. Neither authenticity, confidentiality, anonymity nor off-the-record are captured by the base chat sessions abstraction. Yet, following a modeling technique introduced in recent work [34], we show how to extend chat sessions to provide these extra properties. This has two important advantages:

- it makes our abstraction cleaner and easier to reason about because it can be understood independently of these extra guarantees; and
- it allows for a cleaner understanding of these additional guarantees because they can also be understood independently of our abstraction.

Stronger security statements: Another advantage of this modularity is that it allows for stronger statements with surprisingly trivial proofs. (See, e.g. the proof of Corollary 3). Our results "lift" the security properties from (stateless) communication channels to chat sessions (Figure 1 illustrates this); the only assumptions that seem inherently necessary from the underlying channels are consistency and replay-protection. However, even an insecure channel provides these properties. (In other words, our Chat Sessions construction can be instantiated from an insecure channel.)

Post-Compromise Forward Secrecy (PCFS) can be modeled following the same approach that [34] uses to capture confidentiality and authenticity. Of course the resulting model will be inherently more involved—due to the added complexity of the PCFS guarantee—but still our results provide strong evidence that such guarantee is also lifted by our construction.[15]

Efficiency Advantages: An important property one expects from a messenger is efficiency: not only in terms of encryption and decryption times, but also in terms of ciphertext sizes. (Efficiency and scalability have indeed been an important focus of the messaging literature [21,8,31,10,9,3,4].) On this regard we make two points:

1. Our chat sessions construction is *very* efficient.
2. While the only construction of the communication channels used by our construction is from [34], and is based on Multi-Designated Receiver Signed Public Key Encryption schemes [38]—for which ciphertext sizes (and hence encryption and decryption times) are inherently linear in the number of

[14] The only focus of [13] is group administration; their notions do not disallow (nor capture) group-splits, and their setting still relies on a delivery service for liveness.

[15] We only write "strong evidence" because we are unaware of a (Constructive Cryptography) model for PCFS that is compatible with our notions, and therefore cannot write a formal statement. Nevertheless, in [30], Jost et al. introduce a PCFS model for the two party case, and we believe a model for the group case could be defined based on their work. Doing so is an interesting direction for future work.

recipients ([23, Theorem 1])—if one is not willing to pay the extra price required for such strong security guarantees one can alternatively consider more efficient schemes (providing fewer guarantees). For example, if one only requires authenticity, then the underlying channels could be constructed using standard sEUF-CMA secure signatures (which can be made compact via hash-then-sign).

2.2 Building on Chat Sessions: UatChat

We show how to use the chat sessions abstraction by constructing a messaging application on top of it. The main principle behind using chat sessions is ensuring parties see subgraphs of the graphs output by chat sessions' Read operations (which are already guaranteed to be consistent). In UatChat parties can 1. create chats with a given set of participants; 2. propose adding/removing parties to/from existing chats; 3. vote on proposals;[16] and 4. write messages—which may include a set of prior commands the message is "replying to". UatChat defines permissions policy $\mathfrak{U}$, which enforces group modifications must be unanimous: a proposal only takes effect if all group members agree. For example, to add a party P' to a chat with set of members $\mathcal{S}$, a party $P \in \mathcal{S}$ needs to propose adding P' and then all parties in $\mathcal{S}$ need to agree with this proposal (by voting). (For removing a party P, it is not necessary for P to agree to the change, only the other members in $\mathcal{S}$.)

Note: In messengers it is often necessary for party addition proposals to include the current state of the group and for each group member to have to acknowledge this state: these acknowledgments guarantee to the added party that it is indeed being added to the group. This is needed, for example, in policies where only certain group members have permissions to add new parties to the group. To see why, consider a group management policy distinguishing administrators (admins) from non-administrator members (non-admins), being that only admins can promote other members to become admins, and make changes to the set of members of the group (i.e. add/remove members to/from the group); and consider a group of two parties, Alice and Bob, where Alice is the sole administrator.[17] A dishonest Bob could try deceiving an honest outsider, say Charlie, into believing that he was added to the group; however, by requiring an acknowledgment from other group members, Charlie only considers himself part of the group once Alice would acknowledge it. But since that would not occur, Bob would not deceive Charlie.

Group versions: unconciliable command orderings. The inexistence of a total order on the commands issued by group members makes it unavoidable that a chat may have unconciliable versions even when all parties are honest. To illustrate,

[16] Voting on a proposal means agreeing to it: if a party does not agree with a proposal then it simply does not vote.

[17] This policy is similar to those implemented in messengers such as WhatsApp [2] and Signal [1].

suppose that a party P_1 just created a chat with a set of (all honest) parties $\mathcal{S} = \{P_1, P_2, P_3, P_4\}$. Then, suppose that, concurrently, P_2 and P_3 propose to remove, respectively, P_3 and P_2 from the chat, and let $prop_2$ and $prop_3$ be P_2's and P_3's proposals, respectively. Finally, suppose that P_1 receives $prop_2$ first, and immediately votes in its favor, whereas P_4 receives $prop_3$ first, and immediately votes in its favor too. One can then ask, when P_1 and P_4 later receive $prop_3$ and $prop_2$, respectively, what should happen? This is a typical problem that shows up in the theory of parallel computing [42,27,26,12], a topic with a rather vast literature. There are various ways to handle this (type of) problem; for simplicity, in our messenger there can be multiple versions of the same group that may evolve concurrently; applied to this particular case, there would be two new versions of the group chat: one where the proposal $prop_2$ may take effect, and one where $prop_3$ may take effect. Whether any of these changes actually takes effect then depends on parties agreeing with them, but it is possible for the two proposals to come into effect. We emphasize that our goal here is showing how one can use the chat sessions abstraction to construct a messaging application, not to come up with an "intuitive and easy to use" messenger. Nevertheless, it is an interesting direction for future work to consider other possible constructions of messaging applications, perhaps by leveraging what is known from communities working on concurrent/parallel computing.

3 Preliminaries

We use the same notation and adopt the same conventions from [34], which we now introduce. (Much of this section is taken verbatim from [34] with only minor modifications.) For a set/alphabet S, we denote the set of non-empty vectors/strings over S by S^+. We denote the arity of a vector $\vec{x}$ by $|\vec{x}|$ and its i-th element by x_i. We write $\mathrm{Set}(\vec{x})$ to denote the set induced by $\vec{x}$: $\mathrm{Set}(\vec{x}) \coloneqq \{x_i \mid x_i \in \vec{x}\}$. We will denote the set of all parties by $\mathcal{P}$. For any subset of parties $\mathcal{S} \subseteq \mathcal{P}$, we denote by $\mathcal{S}^H$ and $\overline{\mathcal{S}^H}$ the partitions of $\mathcal{S}$ corresponding to honest and dishonest parties, respectively (with $\mathcal{S} = \mathcal{S}^H \uplus \overline{\mathcal{S}^H}$).

3.1 (Simplified) Constructive Cryptography

Our paper's statements are phrased in a (variant of the) simplified version of the Constructive Cryptography (CC) framework [39,36] introduced in [34], which allows for fine-grained information-theoretic security notions and requires no familiarity with CC. (As for [34], all construction statements trivially carry to CC.) We now present the framework our paper uses; much of this (sub-)section is taken verbatim from [37] and [34], with only minor changes.

CC views cryptography as a resource theory: protocols construct new resources from existing (assumed) ones. The notion of resource construction is inherently composable: if a protocol π_1 constructs $\mathbf{S}$ from $\mathbf{R}$ and π_2 constructs $\mathbf{T}$ from $\mathbf{S}$, then running both protocols $(\pi_2 \cdot \pi_1)$ constructs $\mathbf{T}$ from $\mathbf{R}$.

Resources are interactive systems akin to functionalities in UC [18]. Similarly to a function $f : X \to Y$, a resource also has input and output domains; if a resource $\mathbf{R}$ has input domain $\mathcal{X}$ and output (co-)domain $\mathcal{Y}$, we say $\mathbf{R}$ is an $(\mathcal{X}, \mathcal{Y})$ resource. One interacts with a $(\mathcal{X}, \mathcal{Y})$ resource by providing an input $x \in \mathcal{X}$ and receiving an output $y \in \mathcal{Y}$. Formally, resources are random systems [40,41]; in turn, a random system is defined as a sequence of conditional probability distributions [41, Definition 2]. If two $(\mathcal{X}, \mathcal{Y})$-resources $\mathbf{R}$ and $\mathbf{S}$ are the same sequence of conditional probability distributions, we say they are equivalent and write $\mathbf{R} \equiv \mathbf{S}$ [41, Definition 3]. We will describe resources by pseudo-code.

We often attach resources together; for (compatible) resources $\mathbf{R}$ and $\mathbf{S}$, we denote by $\mathbf{R} \cdot \mathbf{S}$ the resource resulting from attaching $\mathbf{R}$ and $\mathbf{S}$. (Resources $\mathbf{R}$ and $\mathbf{S}$ can only be attached together if their composition results in a well-defined sequence of conditional probability distributions—see, e.g. [33, Definition 7]; this is not the case for all pairs of resources.) For n resources $\{\mathbf{R}_i\}_{i=1}^n$, where each $\mathbf{R}_i$ is an $(\mathcal{X}_i, \mathcal{Y}_i)$-resource, if for all distinct $i, j \in [n]$, both $\mathcal{X}_i$ and $\mathcal{Y}_i$ are disjoint from $\mathcal{Y}_j$, then we denote the combined resource—i.e. $\mathbf{R}_1, \dots, \mathbf{R}_n$ attached together—by $\mathbf{R} \coloneqq [\mathbf{R}_1, \dots, \mathbf{R}_n]$, and call $\mathbf{R}$ the parallel composition of $\{\mathbf{R}_i\}_{i=1}^n$.

For an $(\mathcal{X}, \mathcal{Y})$-resource $\mathbf{R}$, an interface $I = (I_\mathcal{X}, I_\mathcal{Y})$ is a pair of subsets of $\mathbf{R}$'s input and output domains, i.e. $I_\mathcal{X} \subseteq \mathcal{X}$ and $I_\mathcal{Y} \subseteq \mathcal{Y}$; we call $I_\mathcal{X}$ and $I_\mathcal{Y}$ input and output interfaces of $\mathbf{R}$, respectively. For two interfaces $I_1 = (I_{1,\mathcal{X}}, I_{1,\mathcal{Y}})$ and $I_2 = (I_{2,\mathcal{X}}, I_{2,\mathcal{Y}})$, we say that I_1 is a subset of I_2—or write $I_1 \subseteq I_2$—to mean $I_{1,\mathcal{X}} \subseteq I_{2,\mathcal{X}}$ and $I_{1,\mathcal{Y}} \subseteq I_{2,\mathcal{Y}}$. Similarly, we say I_1 and I_2 are disjoint—or write $I_1 \cap I_2 = \emptyset$—to mean $I_{1,\mathcal{X}} \cap I_{2,\mathcal{X}} = \emptyset$ and $I_{1,\mathcal{Y}} \cap I_{2,\mathcal{Y}} = \emptyset$. We define the union of interfaces I_1 and I_2 as $I_1 \cup I_2 \coloneqq (I_{1,\mathcal{X}} \cup I_{2,\mathcal{X}}, I_{1,\mathcal{Y}} \cup I_{2,\mathcal{Y}})$.

A set of interfaces $\mathcal{I}$ of an $(\mathcal{X}, \mathcal{Y})$-resource $\mathbf{R}$ is one such that any distinct interfaces $I_1, I_2 \in \mathcal{I}$ are disjoint, and the union of all interfaces in $\mathcal{I}$ is $\mathbf{R}$'s input and output domains, i.e. $(\mathcal{X}, \mathcal{Y}) = \bigcup_{I \in \mathcal{I}} I$.

When considering (simulator-based) security notions it is often helpful to have the notion of a party. For a set of n parties $\mathcal{P} \coloneqq (P_1, \dots, P_n)$, one considers a set of interfaces $\mathcal{I}$ where for each party $P \in \mathcal{P}$ there is an interface $I_P = (I_{P,\mathcal{X}} \coloneqq (\{P\} \times \mathcal{X}_P), I_{P,\mathcal{Y}} \coloneqq (\{P\} \times \mathcal{Y}_P))$. We say that $I_{P,\mathcal{X}}$ and $I_{P,\mathcal{Y}}$ are P's input and output interfaces for $\mathbf{R}$, respectively.

A *converter* is an $(\mathcal{X}, \mathcal{Y})$-resource that is executed either locally by a single party or cooperatively by multiple parties. The inside interface connects to (a subset of those parties' interfaces of) the available resources, resulting in a new resource. For instance, connecting a converter α to Alice's interface A of a resource $\mathbf{R}$ results in a new resource denoted $\alpha^A\mathbf{R}$; we denote the inside interface of α by $\alpha.in$. The outside interface of α, denoted $\alpha.out$, is the new A-interface of $\alpha^A\mathbf{R}$. This means resource $\mathbf{R}$'s A interface is no longer present in the new resource $\alpha^A\mathbf{R}$: it is covered by converter α. Converters applied at different interfaces commute [28, Proposition 1]: $\beta^B\alpha^A\mathbf{R} \equiv \alpha^A\beta^B\mathbf{R}$.

A protocol is given by a tuple of converters $\pi = (\pi_{P_i})_{P_i \in \mathcal{P}}$, one for each party $P_i \in \mathcal{P}$. Simulators are also given by converters. For a party set $\mathcal{S}$, $\pi^\mathcal{S}\mathbf{R}$ denotes $(\pi_{P_i})_{P_i \in \mathcal{S}}\mathbf{R}$. When clear from context, we omit the interfaces π connects to, writing simply $\pi\mathbf{R}$.

Definition 1 (Construction). *Let* $\mathbf{R}$ *and* $\mathbf{S}$ *be two resources with a free interface* I_F*, and* π *a protocol for* $\mathbf{R}$*. We say* π *constructs* $\mathbf{S}$ *from* $\mathbf{R}$ *if there is a simulator* sim *such that* $\pi\mathbf{R} \equiv \mathsf{sim}\mathbf{S}$*, i.e. are perfectly indistinguishable and the interfaces of* sim*, of* π *and* I_F *are all pairwise disjoint. We call* $\mathbf{R}$ *the assumed resource and* $\mathbf{S}$ *the ideal resource.*

3.2 Modeling Access Control via Repositories

We use the repository model from [37,34] to capture access control. A repository contains a set of registers and a corresponding set of register identifiers IdSet; a register is a pair $\mathtt{reg} = (\mathtt{id}, m)$, where m is a message and $\mathtt{id}$ is the register's identifier, which uniquely identifies it among all repositories. We consider two types of repository access rights: *read access* and *write access*. We denote by $\mathcal{W}$ and $\mathcal{R}$ the sets of parties with write and read access to a repository $\mathbf{rep}$, respectively; to make the access permissions explicit we write $\mathbf{rep}^{\mathcal{W}}_{\mathcal{R}}$, but otherwise simply write $\mathbf{rep}$. For example, consider a three party setting with a sender Alice, a receiver Bob and a dishonest third-party Eve—so $\mathcal{P} = \{A, B, E\}$. An insecure repository—which allows everyone to read and write—is given by $\mathbf{INS}^{\mathcal{P}}_{\mathcal{P}}$; a (replay-protected) authentic repository from Alice to Bob is given by $\mathbf{AUT}^{\{A\}}_{\{B,E\}}$. The semantics of atomic repositories is defined in Algorithm 1.

Algorithm 1 Atomic repository $\mathbf{rep}^{\mathcal{W}}_{\mathcal{R}}$.

```
◇ Initialization: IdSet ← ∅

▷ (P ∈ W)-Write(m)
  id ← NewRegister(m)
  IdSet ← IdSet ∪ {id}
  Output(id)

▷ (P ∈ R)-Read
  list ← ∅
  for id ∈ IdSet :
     list ← list ∪ {(id, GetMessage(id))}
  Output(list)
```

Algorithm 2 Repository $\mathbf{REP} = [\mathbf{rep_1}^{\mathcal{W}_1}_{\mathcal{R}_1}, \ldots, \mathbf{rep_n}^{\mathcal{W}_n}_{\mathcal{R}_n}]$.

```
  ▷ (P ∈ P)-Write(rep_i^{W_i}_{R_i}, m)
Require: (P ∈ W_i)
  Output(rep_i-Write(m))

  ▷ (P ∈ P)-Read
  list ← ∅
  for rep_i^{W_i}_{R_i} ∈ REP with P ∈ R_i :
     for (id, m) ∈ rep_i-Read :
        list ← list ∪ (id, (rep_i, m))
  Output(list)
```

Following [37], to model that parties may have access to multiple repositories—say $\mathbf{rep_1}^{\mathcal{W}_1}_{\mathcal{R}_1}, \ldots, \mathbf{rep_n}^{\mathcal{W}_n}_{\mathcal{R}_n}$—we define a new type of repository denoted $\mathbf{REP} = [\mathbf{rep_1}^{\mathcal{W}_1}_{\mathcal{R}_1}, \ldots, \mathbf{rep_n}^{\mathcal{W}_n}_{\mathcal{R}_n}]$, which consists of a parallel composition of atomic repositories equipped with a single read operation that allows parties to (efficiently) read all their incoming messages at once (instead of having to read from each atomic repository $\mathbf{rep_i}$ they have access to). The exact semantics of $\mathbf{REP}$ is defined in Algorithm 2.

3.2.1 Repository Label Notation

This paper also adopts the repository label notation from [37,34]: label $\langle S \to \vec{V} \rangle$ denotes an atomic repository with a (supposed) sender S and (supposed) receiver-vector $\vec{V}$. To be more concrete, let $\langle S \to \vec{V} \rangle \coloneqq \langle S \to \vec{V} \rangle_{\mathcal{R}}^{\mathcal{W}}$, i.e. $\mathcal{W}$ and $\mathcal{R}$ are the sets of writers and readers of $\langle S \to \vec{V} \rangle$. Sender S is always a writer, i.e. $S \in \mathcal{W}$, and receiver-vector $\vec{V}$ is always a subset of the readers, i.e. $\vec{V} \subseteq \mathcal{R}$. Above we wrote supposed because $\mathcal{W}$ may include other parties (in which case $\langle S \to \vec{V} \rangle$ is not authenticated), and $\mathcal{R}$ may include readers that are not part of the receiver vector $\vec{V}$.

3.2.2 Modeling an Asynchronous Network Following [34], we model an asynchronous network via converter Net (Algorithm 3), which has a message delivery interface and ensures honest receivers only read delivered messages.

Algorithm 3 Semantics of Net for a repository $\mathbf{REP} = [\mathbf{rep_1}, \ldots, \mathbf{rep_n}]$.

◇ Initialization
  for $P_i \in \mathcal{P}$ **:**
    $\text{Received}[P_i] \leftarrow \emptyset$

▷ $(P \in \mathcal{P}^H)$-Read
  $\text{list} \leftarrow \emptyset$
  for $(\mathtt{id}, (\mathbf{rep_i}, m)) \in \text{Read}$**,** $\mathtt{id} \in \text{Received}[P]$**:**
    $\text{list} \leftarrow \text{list} \cup (\mathtt{id}, (\mathbf{rep_i}, m))$
  Output(list)

▷ $(P \in \mathcal{P})$-Write$(\mathbf{rep_i}, m)$
  Output(Write$(\mathbf{rep_i}, m)$)

▷ $(P \in \overline{\mathcal{P}^H})$-Read
  Output(Read)

▷ Deliver$(P \in \mathcal{P}^H, \mathtt{id})$
  $\text{Received}[P] \leftarrow \text{Received}[P] \cup \{\mathtt{id}\}$

4 Chat Sessions

The set of messaging parties is denoted $\mathcal{M} = \{P_1, \ldots, P_n\}$; we assume $\mathcal{M}^H$ and $\overline{\mathcal{M}^H}$ are non-empty. ([34] considers two distinct sets: a set $\mathcal{S}$ of senders and a set $\mathcal{R}$ of receivers. Our model is compatible with [34] because we can have each party $P_i \in \mathcal{M}$ play the roles of a sender and receiver.) The set of parties $\mathcal{P}$ from [34] also includes a judge J; for compatibility, we also define the set of parties as $\mathcal{P} = \mathcal{M} \cup \{J\}$; however, for our abstractions J can be ignored.

4.1 Overview

Interfaces. Both the real and the ideal chat sessions resources (defined ahead in Sections 4.3 and 4.4, respectively) allow parties to perform Read and Write operations. When a party $P \in \mathcal{M}^H$ issues a Read operation (which takes no input), these resources output a set of pairs $(\mathtt{sid}, \mathcal{G}^+)$, where $\mathtt{sid}$ is a (chat) *session identifier*—uniquely identifying the chat session—and $\mathcal{G}^+$ (essentially) is a (non-empty) digraph corresponding to P's view of that particular session. Write operations are uniquely identified by an $\mathtt{id}$ and have an associated writer/sender S, vector of receivers $\vec{V}$, and message $m \coloneqq (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$—a triple comprising an $\mathtt{sid}$, a command $\mathtt{cmd}$ and a set Acks of (prior) Write operation identifiers to acknowledge. These operations take as input an $\mathtt{sid}$, a vector of receivers $\vec{V}$, a command $\mathtt{cmd}$ and a set of acknowledgements Acks, and output their own identifier $\mathtt{id}$.

Policies. Both the real and the ideal chat sessions resources are parameterized by a policy $\mathfrak{P}$ which defines two (deterministic) predicates: IsRoot and IsValid. IsRoot takes as input a session identifier $\mathtt{sid}$, a sender S, a vector of receivers $\vec{V}$ and a command $\mathtt{cmd}$; IsValid's input includes all of IsRoot's inputs plus an extended graph $\mathcal{G}^+ = (V^+, E^+)$—corresponding to a party's view of that session's graph—and a set Acks of Write operation $\mathtt{id}$s to acknowledge.

The Abstraction. **ChatSessions**[$\mathfrak{P}$] embodies a type of stateful consistency notion. For each existing chat session $\mathtt{sid}$, it keeps track of a global directed graph (digraph) $\mathcal{G} = (V, E)$.[18] A node $v \in V$ of such global graph is the identifier $\mathtt{id}$ of a Write operation, and for any node $v \in V$, we have that $(u, v) \in E$ if and only if the (message) triple $m := (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})$ corresponding to (Write) v is such that $u \in \mathrm{Acks}$. The elements $\mathcal{G}^+ = (V^+, E^+)$ output by Read operations are of a different type than the global digraphs: on one hand, each $u \in V^+$ is of the form $(\mathtt{id}, (\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})))$—with $\mathtt{id}$ being a Write operation identifier, $\langle S \to \vec{V}\rangle$ being a label identifying a sender S and the vector of receivers $\vec{V}$ (see Algorithm 2), and with $\mathtt{sid}$, $\mathtt{cmd}$ and Acks being, respectively, the session identifier, the command and the set of Write operations acknowledged; on the other hand, the elements of E^+ (i.e. edges) are pairs $(\mathtt{id}, \mathtt{id}')$ of Write operation identifiers. Since there are no two different tuples $u, v \in V^+$ with the same Write operation identifier (i.e. $\forall u, v \in V^+, u \neq v \to u.\mathtt{id} \neq v.\mathtt{id}$), one can alternatively think of $\mathcal{G}^+$ as a triple $\mathcal{G}^+ = (\mathcal{G}' = (V', E'), \mathtt{f})$ where $\mathcal{G}'$ is (informally) a subgraph of the global digraph from before and $\mathtt{f}$ is a function—with domain V'—mapping each Write operation $\mathtt{id} \in V'$ to a tuple $(\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$. We call these the extended (di)graphs. We denote the extended version of a graph $\mathcal{G} = (V, E)$ by $\mathcal{G}^+ = (V^+, E^+)$ and call each $u \in V^+$ an extended node.

4.2 Helper Functions

The three helper functions described below are key to understanding both our abstraction and protocol; they are formally defined in Algorithm 4. We note their descriptions rely on variables that are not defined at this point;[19] we explain what is necessary so one can understand the helper functions. In addition we consider a policy $\mathfrak{P}$ (which defines predicates IsRoot and IsValid).

Extended: On input a (chat session) graph $\mathcal{G} = (V, E)$, this function outputs the corresponding extended graph $\mathcal{G}^+ = (V^+, E^+)$. In the description, variable Contents is a mapping from Write operation identifiers to their corresponding contents, which are pairs $(\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$ that include a label and a message; in our case, messages are triples that consist of an identifier $\mathtt{sid}$, a command $\mathtt{cmd}$, and a set Acks of Write operation identifiers.

[18] Formally, these digraphs are not actual graphs because there may be edges $(u, v) \in E$ for which $u \notin V$; for simplicity we still call these objects digraphs.

[19] They are defined by our **ChatSessions**[$\mathfrak{P}$] abstraction and our ChatSessionsProt[$\mathfrak{P}$] protocol.

$\mathsf{UpdatedGraph}$: On input a graph $\mathcal{G}_0$ and a set ToHandle of potential new nodes, this function outputs a graph $\mathcal{G}$ containing $\mathcal{G}_0$ plus all the nodes from ToHandle that were added (together with their edges), and also outputs a set Handled which is the subset of ToHandle consisting of the added nodes. Variable Contents in the description is the same as above.

$\mathsf{InducedPartyGraph}^+$: On input a session identifier $\texttt{sid}$ and a party P, this function outputs an extended graph corresponding to P's view of the chat sessions graph identified by $\texttt{sid}$. This function is only used for describing our abstraction **ChatSessions**[$\mathfrak{P}$] (but not our protocol). Variables:

SessionGraphs: maps chat session identifiers to their corresponding graphs—i.e. the global graphs our abstraction keeps track of for each chat, as explained in Section 4.1. In particular, SessionGraphs[$\texttt{sid}$] is the one corresponding to $\texttt{sid}$.

AREP-Read $\cup$ Sent[P]: the set of messages that were either already delivered to party P—**AREP**-Read—or that were sent by P—Sent[P]; the contents of this variable have the same structure as the ones in variable Contents.

Algorithm 4 Helper functions.

$\diamond$ $\mathsf{Extended}(\mathcal{G} = (V, E))$: **return** $\mathcal{G}^+ := (\{(\texttt{id}, \text{Contents}[\texttt{id}]) \mid \texttt{id} \in V\}, E)$

$\diamond$ $\mathsf{UpdatedGraph}(\mathcal{G}_0, \text{ToHandle})$
- $i \leftarrow 0, \text{Handled} \leftarrow \emptyset$
- **repeat**
  - $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i$
  - **for** $\texttt{id} \in \text{ToHandle}$ **with** $\texttt{id} \notin \text{Handled}$ **:**
    - $(\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks})) \leftarrow \text{Contents}[\texttt{id}]$
    - **if** $\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}_{i+1}), S, \vec{V}, \texttt{cmd}, \text{Acks})$ **:**
      - $\mathcal{G}_{i+1} \leftarrow (\mathcal{G}_{i+1}.V \cup \{\texttt{id}\}, \mathcal{G}_{i+1}.E \cup (\text{Acks} \times \{\texttt{id}\}))$
      - $\text{Handled} \leftarrow \text{Handled} \cup \{\texttt{id}\}$
  - $i \leftarrow i + 1$
- **until** $\mathcal{G}_i = \mathcal{G}_{i-1}$
- **return** $(\mathcal{G}_i, \text{Handled})$

$\diamond$ $\mathsf{InducedPartyGraph}^+(\texttt{sid}, P)$
- $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}[\texttt{sid}]$
- $V_P := V \cap \{\texttt{id} \mid (\texttt{id}, (\cdot, (\texttt{sid}, \cdot, \cdot))) \in \textbf{AREP}\text{-}\textsc{Read} \cup \text{Sent}[P]\}$
- $V_0 := \{\texttt{id} \in V_P \mid \text{Contents}[\texttt{id}] = (\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \cdot)) \wedge \mathfrak{P}[\textsc{IsRoot}](\texttt{sid}, S, \vec{V}, \texttt{cmd})\}$
- $i \leftarrow 0$
- **repeat**
  - $V_{i+1} \leftarrow V_i$
  - **for** $\texttt{id} \in V_P$ **:**
    - $(\cdot, (\cdot, \cdot, \text{Acks})) \leftarrow \text{Contents}[\texttt{id}]$
    - **if** $\text{Acks} \subseteq V_i$ **:** $V_{i+1} \leftarrow V_{i+1} \cup \{\texttt{id}\}$
  - $i \leftarrow i + 1$
- **until** $V_i = V_{i-1}$
- $V_E := \{\texttt{id} \mid (\texttt{id}, \texttt{id}') \in E\}$
- **return** $\mathsf{Extended}(\mathcal{G}_i := (V_i, E \cap (V_E \times V_i)))$

4.3 Real World

We now define the real world resource, i.e. the assumed resource and the protocol parties run. The assumed resource is an asynchronous repository **AREP**, which consists of a repository **REP** (Algorithm 2) with converter Net (Algorithm 3) attached, i.e. **AREP** $:= \mathsf{Net} \cdot$ **REP** with **REP** being defined as

$$\mathbf{REP} := \left[\langle P \to \vec{V} \rangle^{\{P\} \cup \overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+}. \tag{4.1}$$

As for the protocol, honest parties run converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ (Algorithm 5), which is parameterized by a policy $\mathfrak{P}$. The real world system is

$$\mathbf{R}[\mathfrak{P}] := \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AREP}. \tag{4.2}$$

Algorithm 5 Converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$.

$\diamond$ Initialization: SessionGraphs, Contents $\leftarrow \emptyset$

$\triangleright$ $(P \in \mathcal{M}^H)$-Read
  $\mathsf{ProcessReceived}$
  Output$(\{(\mathtt{sid}, \mathsf{Extended}(\mathcal{G})) \mid (\mathtt{sid}, \mathcal{G}) \in \text{SessionGraphs} \land \mathcal{G} \neq (\emptyset, \emptyset)\})$

$\triangleright$ $(P \in \mathcal{M}^H)$-Write$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$
  $\mathsf{ProcessReceived}$
  $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}[\mathtt{sid}]$ // If $\mathtt{sid} \notin$ SessionGraphs then $\mathcal{G} = (\emptyset, \emptyset)$.
  Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}), P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow$ **AREP**-Write$(\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{SessionGraphs}[\mathtt{sid}] \leftarrow (V \cup \{\mathtt{id}\}, E \cup (\text{Acks} \times \{\mathtt{id}\}))$
  Output$(\mathtt{id})$

$\diamond$ $\mathsf{ProcessReceived}$
  ToHandle $\leftarrow \emptyset$
  for $(\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))) \in$ **AREP**-Read **with** $\mathtt{id} \notin \text{SessionGraphs}[\mathtt{sid}].V$ **:**
    $\text{Contents}[\mathtt{id}] \leftarrow (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
    $\text{ToHandle}[\mathtt{sid}] \leftarrow \text{ToHandle}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  for $\mathtt{sid} \in$ ToHandle **:**
    $(\text{SessionGraphs}[\mathtt{sid}], \cdot) \leftarrow \mathsf{UpdatedGraph}(\text{SessionGraphs}[\mathtt{sid}], \text{ToHandle}[\mathtt{sid}])$

4.4 Ideal Chat Sessions

ChatSessions$[\mathfrak{P}]$ is formally defined in Algorithm 6; to simplify its description we rely on the asynchronous repository **AREP** from the real world resource.

Remark 1. We *purposefully* define **ChatSessions**$[\mathfrak{P}]$ so it captures a minimal set of guarantees (e.g. *per se* it does not provide authenticity nor confidentiality). This is not a limitation: in Section 6 we show how to capture authenticity, Off-The-Record, confidentiality and anonymity guarantees. It is an *advantage*: **ChatSessions**$[\mathfrak{P}]$ is more general and more abstract; it is independent of such extra guarantees.

Algorithm 6 The **ChatSessions**[$\mathfrak{P}$] abstraction.

$\diamond$ Initialization
AREP-Initialization
SessionGraphs, Contents, ToHandle $\leftarrow \emptyset$
for $P \in \mathcal{M}^H$ **:** Sent[P] $\leftarrow \emptyset$

$\triangleright$ Deliver(P, $\mathtt{id}$): **AREP**-Deliver(P, $\mathtt{id}$)

$\triangleright$ ($P \in \overline{\mathcal{P}^H}$)-Read: Output(**AREP**-Read)

$\triangleright$ ($P \in \mathcal{M}^H$)-Write($\mathtt{sid}, \mathtt{cmd}, \vec{V}$, Acks)
$\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$
Require: $\mathfrak{P}$[IsValid]($\mathtt{sid}, \mathcal{G}^+, P, \vec{V}, \mathtt{cmd}$, Acks)
$\mathtt{id} \leftarrow$ **AREP**-Write($\langle P \rightarrow \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
Contents[$\mathtt{id}$] $\leftarrow (\langle P \rightarrow \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
Sent[P] $\leftarrow$ Sent[P] $\cup \{\mathtt{id}\}$
$\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
Output($\mathtt{id}$)

$\triangleright$ ($P \in \overline{\mathcal{P}^H}$)-Write($\langle S \rightarrow \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
$\mathtt{id} \leftarrow$ **AREP**-Write($\langle S \rightarrow \vec{V} \rangle, m$)
Contents[$\mathtt{id}$] $\leftarrow (\langle S \rightarrow \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
$\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
Output($\mathtt{id}$)

$\triangleright$ ($P \in \mathcal{M}^H$)-Read: Output($\{(\mathtt{sid}, \mathcal{G}^+) \mid \mathcal{G}^+ = \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P) \wedge \mathcal{G}^+ \neq (\emptyset, \emptyset)\}$)

$\diamond$ $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
ToHandle[$\mathtt{sid}$] $\leftarrow$ ToHandle[$\mathtt{sid}$] $\cup \{\mathtt{id}\}$
(SessionGraphs[$\mathtt{sid}$], Handled) $\leftarrow \mathsf{UpdatedGraph}$(SessionGraphs[$\mathtt{sid}$], ToHandle[$\mathtt{sid}$])
ToHandle[$\mathtt{sid}$] $\leftarrow$ ToHandle[$\mathtt{sid}$] $\setminus$ Handled

4.4.1 Policy Requirements.

We now define three policy requirements we assume in our analysis. Let $\mathfrak{P}$ be a policy defining predicates IsRoot and IsValid.

For some chat session identifier $\mathtt{sid}$, command $\mathtt{cmd}$, sender $S \in \mathcal{M}$ and vector of receivers $\vec{V} \in \mathcal{M}^+$, we call $(\mathtt{sid}, S, \vec{V}, \mathtt{cmd})$ a *root* if IsRoot$(\mathtt{sid}, S, \vec{V}, \mathtt{cmd}) = 1$. We start by defining what it means for a chat session graph to be *proper*. (Ahead, we will always assume that the graphs input to IsValid are proper.)

Definition 2 (Proper (Extended) Chat Session Graph). *The empty graph* $\mathcal{G}^+_\emptyset := (\emptyset, \emptyset)$ *is* proper*. Let* $\mathcal{G}^+ = (V^+, E^+)$ *be a proper graph. For any label* $\langle S \rightarrow \vec{V} \rangle$*, any triple* $(\mathtt{sid}, \mathtt{cmd}, \text{Acks})$*, and any* $\mathtt{id}$ *for a corresponding* Write *operation, if* IsValid$(\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$*, then* $\mathcal{G}^{+\prime} = (V^{+\prime}, E^{+\prime})$ *is* proper*, where* $V^{+\prime} := V^+ \cup \{(\mathtt{id}, (\langle S \rightarrow \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks})))\}$*, and* $E^{+\prime} := E^+ \cup (\text{Acks} \times \{\mathtt{id}\})$*.*

The first requirement is that any root node is a valid node:

Requirement 1 (Root validity). *For any proper graph* $\mathcal{G}^+ = (V^+, E^+)$*, any root* $(\mathtt{sid}, S, \vec{V}, \mathtt{cmd})$ *and any finite set of* Write *operation identifiers* Acks*:* IsValid$(\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$*.*

Requirement 2 guarantees a non-root node is only valid if its set of acknowledged nodes is contained in the input graph:

Requirement 2 (Non-root acknowledgements). *For any proper graph* $\mathcal{G}^+ = (V^+, E^+)$*, any quadruple* $(\mathtt{sid}, S, \vec{V}, \mathtt{cmd})$ *that is not a root, and any finite set* *of* WRITE *operation identifiers* Acks*, if* $\textsc{IsValid}(\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \text{Acks}) = 1$*, then* $\forall \texttt{id} \in \text{Acks}$ *there is a node* $(\texttt{id}, \cdot) \in V^+$*.*

Informally, the third requirement captures that a command's validity is consistent among any proper (extended) subgraphs of a chat session.

Requirement 3 (Subgraph validity). *Let* $\mathcal{G}^+ = (V^+, E^+)$ *be some proper graph,* S *be some party* $S \in \mathcal{M}$*,* $\vec{V}$ *be some (non-empty) vector of parties* $\vec{V} \in \mathcal{M}^+$*, and* $(\texttt{sid}, \texttt{cmd}, \text{Acks})$ *be some triple—where* Acks *is a set of* WRITE *operation identifiers. Then, for every subset* ${V'}^+ \subseteq V^+$*, and letting* ${\mathcal{G}'}^+$ *be the (extended) sub-graph of* $\mathcal{G}^+$ *induced by* ${V'}^+$*, if* ${\mathcal{G}'}^+$ *is proper and* $\forall \texttt{id} \in \text{Acks}$ *there is a node* $(\texttt{id}, \cdot) \in {V'}^+$*, then*

$$\textsc{IsValid}(\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \text{Acks}) = \textsc{IsValid}(\texttt{sid}, {\mathcal{G}'}^+, S, \vec{V}, \texttt{cmd}, \text{Acks}).$$

4.5 Security Analysis

Theorem 1. *For any policy* $\mathfrak{P}$ *satisfying Requirements 1, 2 and 3:*

$$\mathbf{R}[\mathfrak{P}] \equiv \mathbf{ChatSessions}[\mathfrak{P}].$$

(See Appendix Section B for the proof.)

5 UatChat: A Decentralized Messenger

We introduce UatChat to exemplify how one can construct a messenger on top of our Chat Sessions abstraction. (We rely on the Chat Sessions abstraction to describe UatChat.)

In UatChat (Algorithms 9 and 10) there are two main types of operations: READ and command writing. The first output the graphs of each chat a party is in, similarly to chat sessions. There are four command writing interfaces: CREATECHAT, PROPOSECHANGE, VOTE and WRITE. These writing interfaces take as input a chat session identifier `sid` and upon a query issue a chat sessions' WRITE operation and output the resulting `id`; concretely:

$\textsc{CreateChat}(\texttt{sid}, \vec{G})$: for group member vector $\vec{G}$, issues a WRITE with command $(\mathsf{Create}, \vec{G})$ and acknowledgements $\text{Acks} = \emptyset$;

$\textsc{ProposeChange}(\texttt{sid}, \texttt{vid}, \mathsf{change}, P)$: for a `vid` specifying the chat *version* to which the change is to be made—where change and P' specify the actual change: if $\mathsf{change} = \mathsf{Add}$, P' is being added; if $\mathsf{change} = \mathsf{Rm}$, P' is being removed—issues a WRITE with command $(\texttt{vid}, \mathsf{change}, \vec{G}, P')$, where vector $\vec{G}$ is the current group roster for chat version `vid`; (Adding $\vec{G}$ to the command allows the joining party to learn the current group roster and each group member to confirm this roster.)

$\textsc{Vote}(\texttt{sid}, \texttt{vid})$: for proposed chat version `vid`, issues a WRITE with command $(\texttt{vid}, \mathsf{Vote})$ and acknowledgements $\text{Acks} = \{\texttt{vid}\}$; and

$\textsc{Write}(\texttt{sid}, \texttt{vid}, m, \texttt{ReplyTo})$: for chat version `vid`, message m and set `ReplyTo` of prior commands to be explicitly acknowledged, issues a WRITE with command $(\texttt{vid}, \mathsf{Msg}, m, \texttt{ReplyTo})$ and a set of acknowledgements that includes each command in `ReplyTo` (i.e. $\texttt{ReplyTo} \subseteq \text{Acks}$).

5.1 The Unanimous Policy $\mathfrak{U}$

The first step in constructing a messenger is defining a policy to parameterize chat sessions; UatChat's policy—defined in Algorithm 7—is denoted $\mathfrak{U}$.

To define $\mathfrak{U}$ we rely on a helpful definition:

Definition 3. *For digraph $\mathcal{G} = (V, E)$ and node $v \in V$, the v*-sourced subgraph *of $\mathcal{G}$, denoted* $\mathsf{Sourced}(\mathcal{G}, v)$*, is the subgraph of $\mathcal{G}$ induced by the set of vertices $u \in V$ that are reachable from v—i.e. to which there is a directed path in $\mathcal{G}$ starting in v—plus node v itself.*

UatChat allows for five types of commands: Create, Add, Rm, Vote and Msg. Only commands of type Create, Add or Rm may be roots; specifically, for chat identifier $\mathtt{sid}$, sender S, group vector $\vec{G}$ and receiver vector $\vec{V}$—where S must be an element of the group, i.e. $S \in \mathrm{Set}(\vec{G})$, and $\vec{G}$ has no duplicate parties, i.e. $|\vec{G}| = |\mathrm{Set}(\vec{G})|$:

- $(\mathsf{Create}, \vec{G})$ is valid if the receiver vector matches the group vector, i.e. $\vec{V} = \vec{G}$;
- $(\cdot, \mathsf{proposal} \in \{\mathsf{Add}, \mathsf{Rm}\}, \vec{G}, P)$ is valid if P is not in the group and the receiver vector matches the group vector with P appended, i.e. $\vec{V} = \vec{G} \mid\mid P$.

A Vote command $(\mathtt{vid}, \mathsf{Vote})$ is valid if $\mathtt{vid}$ is a WRITE operation identifier for a root that is either an Add or a Rm proposal—which requires parties to agree on the proposal—and the set of acknowledgements is just the proposal node itself, i.e. $\mathrm{Acks} = \{\mathtt{vid}\}$. Finally, a Write command $(\mathtt{vid}, (\mathsf{Msg}, \cdot, \mathtt{ReplyTo}))$ is valid if: 1. $\mathtt{vid}$ is the identifier of a root; 2. every node in $\mathtt{ReplyTo}$ is being acknowledged (i.e. $\mathtt{ReplyTo} \subseteq \mathrm{Acks}$); 3. every node in Acks is in the subgraph sourced by $\mathtt{vid}$; and 4. if node corresponding to $\mathtt{vid}$ is an Add or a Rm proposal, then Acks includes a vote from each party whose vote is required for the proposal to take effect. This last condition is what enforces the unanimity policy: a proposal can only take effect if all parties agree on it. Theorem 2 trivially follows by inspection of $\mathfrak{U}$'s definition (Algorithm 7).

Theorem 2. $\mathfrak{U}$ *satisfies Requirements 1, 2 and 3.*

5.2 Defining UatChat

While policy $\mathfrak{U}$ already gives most of the guarantees we want from our messenger—by establishing which commands are valid via predicates ISROOT and ISVALID—one may want to require more for a root to be valid: Requirement 1 implies that for any $\mathcal{G}^+ = (V^+, E^+)$ and any set Acks, if a quadruple $(\mathtt{sid}, S, \vec{V}, \mathtt{cmd})$ is a root, then $\mathfrak{U}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$. To exemplify, we add such extra requirements to our messenger (see Algorithm 8). On the other hand, one may want the messenger to hide (to honest parties) certain nodes in a chat's graph; we also exemplify this with our messenger.

Algorithm 7 Unanimous policy $\mathfrak{U}$; Below, Sourced is as in Definition 3.

$\diamond$ $\textsc{IsRoot}(\texttt{sid}, S, \vec{V}, \texttt{cmd})$
  $(\text{Voters}, \cdot, \vec{G}, \cdot) \leftarrow \mathsf{RootCmdInfo}(\texttt{cmd})$
  if $(\text{Voters}, \cdot, \vec{G}, \cdot) = \perp$ **:**
    return 0
  return $\big(|\vec{G}| = |\text{Set}(\vec{G})|\big) \wedge \big(S \in \text{Voters}\big) \wedge \big(\vec{V} = \vec{G}\big)$

$\diamond$ $\textsc{IsValid}(\texttt{sid}, \mathcal{G}^+ = (V^+, E^+), S, \vec{V}, \texttt{cmd}, \text{Acks})$
  if $\textsc{IsRoot}(\texttt{sid}, S, \vec{V}, \texttt{cmd}) = 1$ **:** // Any root is a valid node.
    return 1
  if $(\text{Acks} \subseteq V^+) \wedge \texttt{cmd} = (\texttt{vid}, \cdot)$ **:**
    if $(\texttt{vid} \notin V^+) \vee (\mathsf{NodeIsRoot}(\texttt{vid}, V^+) = 0)$ **:**
      return 0
    $(\text{Voters}, \text{Votable}, \vec{G}_{\text{pre-vote}}, \vec{G}_{\text{post-vote}}) \leftarrow \mathsf{RootCmdInfo}(\mathsf{CmdOf}(\texttt{vid}, V^+))$
    if $\texttt{cmd} = (\cdot, \mathsf{Vote})$ **:**
      return $\text{Votable} \wedge \big(S \in \text{Voters}\big) \wedge \big(\vec{V} = \vec{G}_{\text{pre-vote}}\big) \wedge \big(\text{Acks} = \{\texttt{vid}\}\big)$
    else if $\texttt{cmd} = (\cdot, (\mathsf{Msg}, \cdot, \texttt{ReplyTo}))$ **:**
      Compute $\mathcal{G}^+_{\text{src}} := (V^+_{\text{src}}, E^+_{\text{src}}) \leftarrow \mathsf{Sourced}(\mathcal{G}^+, \texttt{vid})$
      return $\big(\texttt{ReplyTo} \subseteq \text{Acks} \subseteq V^+_{\text{src}}\big) \;\wedge\; \big(\mathsf{Voted}(\texttt{vid}, \text{Acks}, V^+_{\text{src}}) = \text{Voters}\big) \;\wedge\; \big(\vec{V} = \vec{G}_{\text{post-vote}}\big)$
  return 0

$\diamond$ $\mathsf{Voted}(\texttt{vid}, \text{Acks}, V^+)$
  $\text{Voted} \leftarrow \{\mathsf{SenderOf}(\texttt{vid}, V^+)\}$
  for $\texttt{id} \in \text{Acks}$ **with** $\mathsf{CmdOf}(\texttt{id}, V^+) = (\texttt{vid}, \mathsf{Vote})$ **:**
    $\text{Voted} \leftarrow \text{Voted} \cup \{\mathsf{SenderOf}(\texttt{id}, V^+)\}$
  return Voted

$\diamond$ $\mathsf{SenderOf}(\texttt{id}, V^+)$
  $(\langle S \to \vec{V} \rangle, \cdot) \leftarrow V^+[\texttt{id}]$
  return S

$\diamond$ $\mathsf{CmdOf}(\texttt{id}, V^+)$
  $(\cdot, (\cdot, \texttt{cmd}, \cdot)) \leftarrow V^+[\texttt{id}]$
  return cmd

$\diamond$ $\mathsf{NodeIsRoot}(\texttt{id}, V^+)$
  $(\langle S' \to \vec{V}' \rangle, (\texttt{sid}', \texttt{cmd}', \cdot)) \leftarrow V^+[\texttt{id}]$
  return $\textsc{IsRoot}(\texttt{sid}', S', \vec{V}', \texttt{cmd}')$

$\diamond$ $\mathsf{RootCmdInfo}(\texttt{cmd})$
  if $\texttt{cmd} = (\mathsf{Create}, \vec{G})$ **:**
    return $(\text{Set}(\vec{G}), 0, \vec{G}, \vec{G})$
  if $\texttt{cmd} = (\cdot, \mathsf{Add}, \vec{G}, P)$ **:**
    $\vec{G}' \leftarrow \vec{G} \,||\, P$
    return $(\text{Set}(\vec{G}), 1, \vec{G}', \vec{G}')$
  if $\texttt{cmd} = (\cdot, \mathsf{Rm}, \vec{G}, P)$ **:**
    $\vec{G}' \leftarrow \vec{G} \,||\, P$
    return $(\text{Set}(\vec{G}), 1, \vec{G}', \vec{G})$
  return $\perp$

Additional requirements for the validity of a root. Let $\mathcal{G}^+ := (V^+, E^+)$ be a proper graph; consider some tuple $(\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \text{Acks})$:

- if $\texttt{cmd} = (\mathsf{Create}, \vec{G})$, then Acks must be the empty set;
- if $\texttt{cmd} = (\texttt{vid}, \mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}, \vec{G}, P)$, then 1. vid must be in V^+; 2. vid's corresponding node (in V^+) must be a root (in the sense of $\mathfrak{U}$'s IsRoot predicate); 3. if vid's corresponding command is either Add or Rm, then Acks must contain a vote from each of the parties necessary to agreed on vid's proposal; and 4. the group vector $\vec{G}_{\texttt{vid}}$ corresponding to vid must be consistent with $\vec{G}$ (see Algorithm 8).

Hiding unwanted nodes. Generally, a node u is only visible to a party P if all of u's acknowledged nodes are already visible to P; the only case in which a node u is shown to a party P—despite u's acknowledged nodes not being visible to P—is when u's command is $(\texttt{sid}, \mathsf{Add}, \vec{G}, P)$: in this case u becomes visible to P as soon as P receives a corresponding vote from each of the parties in $\vec{G}$ needed for

Algorithm 8 Additional root requirements. Below, $\mathsf{Sourced}$ is as in Definition 3.

$\diamond$ $\textsc{IsRoot-Ext}(\mathtt{sid}, \mathcal{G}^+ = (V^+, E^+), S, \vec{V}, \mathtt{cmd}, \text{Acks})$
  if $\mathfrak{U}[\textsc{IsRoot}](\mathtt{sid}, S, \vec{V}, \mathtt{cmd}) = 0$ **:**
    return 0
  if $\mathtt{cmd} = (\mathsf{Create}, \vec{G})$ **:**
    return $\text{Acks} = \emptyset$
  if $(\mathtt{cmd} = (\mathtt{vid}, \mathsf{change}, \vec{G}, P)) \wedge (\mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}) \wedge (\text{Acks} \subseteq V^+)$ **:**
    if $(\mathtt{vid} \notin V^+) \vee (\mathsf{NodeIsRoot}(\mathtt{vid}, V^+) = 0)$ **:**
      return 0
    Compute $\mathcal{G}^+_{\text{src}} \coloneqq (V^+_{\text{src}}, E^+_{\text{src}}) \leftarrow \mathsf{Sourced}(\mathcal{G}^+, \mathtt{vid})$
    $(\text{Voters}, \text{Votable}, \cdot, \vec{G}_{\mathtt{vid}}) \leftarrow \mathsf{RootCmdInfo}(\mathsf{CmdOf}(\mathtt{vid}, V^+))$
    if $(\text{Votable} = 1) \wedge (\text{Voters} \neq \mathsf{Voted}(\mathtt{vid}, \text{Acks}, V^+))$ **:**
      return 0
    return $(\mathsf{change}, \vec{G}) \in \big\{(\mathsf{Add}, \vec{G}_{\mathtt{vid}}), (\mathsf{Rm}, \mathsf{RemoveFromVector}(\vec{G}_{\mathtt{vid}}, P))\big\}$
  return 0

an unanimous agreement (to add P to chat $\mathtt{sid}$). Proposals' votes only become visible after all votes that are necessary for an unanimous agreement have been received. Finally, proposals to add (resp. remove) a party P to (resp. from) a chat are kept hidden from P until all parties have agreed to the proposal. (This guarantees that an honest party P only sees that it was added to a chat once all the chat's participants agreed to P's addition.)

Consistency. Neither hiding unwanted nodes nor making further requirements for root nodes to be valid affect the consistency of our messenger, because honest parties only see a subgraph of what is output by the chat sessions abstraction (and therefore the subgraphs they read are consistent).

5.2.1 Constructing UatChat. Dishonest parties' capabilities are exactly the same in **ChatSessions**[$\mathfrak{U}$] and **UatChat**, and the same holds for interface $\textsc{Deliver}$ (see Algorithms 6 and 9). This means one can equivalently define the ideal **UatChat** resource via a converter $\mathsf{UatChatProt}$ run by each honest party and attaching it to **ChatSessions**[$\mathfrak{U}$]:

$$\mathsf{UatChatProt}^{\mathcal{M}^H} \cdot \mathbf{ChatSessions}[\mathfrak{U}] \equiv \mathbf{UatChat}.$$

(For completeness, we define converter $\mathsf{UatChatProt}$ in Appendix, Algorithm 16.)

6 The Modularity of ChatSessions[$\mathfrak{P}$]

We now extend **ChatSessions**[$\mathfrak{P}$] to provide various additional security properties, namely authenticity, Off-The-Record, confidentiality and anonymity. We focus on these guarantees because they match the ones captured in the model from [34] in the context of Multi-Designated Receiver Signed Public Key Encryption (MDRS-PKE) schemes [38,20,34]. In particular this allows us to follow [34]'s simple and intuitive modeling technique in the context of **ChatSessions**[$\mathfrak{P}$]. Furthermore, [34]'s MDRS-PKE application semantics are an exact match with the

Algorithm 9 The ideal **UatChat** application. The description below relies on a system **ChatSessions**[$\mathfrak{U}$] (see Algorithm 6). For simpler notation we write **CS**[$\mathfrak{U}$] instead of **ChatSessions**[$\mathfrak{U}$].

$\triangleright$ $(P \in \mathcal{M}^H)$-CreateChat($\mathtt{sid}, \vec{G} \in \mathcal{M}^+$)
Require: $\mathtt{sid} \notin$ **UatChat**-Read
  $\big(\vec{V}, \mathtt{cmd}, \mathrm{Acks}\big) \leftarrow \big(\vec{G}, (\mathsf{Create}, \vec{G}), \emptyset\big)$
Require: IsRoot-Ext($\mathtt{sid}, (\emptyset, \emptyset), P, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$
  Output(**CS**[$\mathfrak{U}$]-Write($\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks}$))

$\triangleright$ $(P \in \mathcal{M}^H)$-ProposeChange($\mathtt{sid}, \mathtt{vid}, \mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}, P' \in \mathcal{M}$)
Require: **BasicReqs**($\mathtt{sid}, \mathtt{vid}, P$)
  $(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} := (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \mathrm{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
  $\vec{G}' \leftarrow (\vec{G} \mid\mid P')$
  $\mathrm{LeafAcks} \leftarrow \{\mathtt{id} \mid (\exists(\mathtt{id}, (\cdot, (\cdot, (\mathtt{vid}, \cdot), \cdot))) \in V^+_{\text{src-vis}}) \wedge (\nexists(\mathtt{id}, \cdot) \in E^+_{\text{src-vis}})\}$
  $\big(\vec{V}, \mathtt{cmd}, \mathrm{Acks}\big) \leftarrow \big(\vec{G}', (\mathtt{vid}, \mathsf{change}, \vec{G}, P'), \mathrm{VoteAcks} \cup \mathrm{LeafAcks}\big)$
Require: IsRoot-Ext($\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$
  Output(**CS**[$\mathfrak{U}$]-Write($\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks}$))

$\triangleright$ $(P \in \mathcal{M}^H)$-Vote($\mathtt{sid}, \mathtt{vid}$)
Require: **BasicReqs**($\mathtt{sid}, \mathtt{vid}, P$)
  $(\vec{G}, \cdot, \mathcal{G}^+_{\text{src-vis}}, \mathrm{MissingVotes}, \cdot) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
Require: $P \in \mathrm{MissingVotes}$
  $\big(\vec{V}, \mathtt{cmd}, \mathrm{Acks}\big) \leftarrow \big(\vec{G}, (\mathtt{vid}, \mathsf{Vote}), \{\mathtt{vid}\}\big)$
Require: IsValid($\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$
  Output(**CS**[$\mathfrak{U}$]-Write($\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks}$))

$\triangleright$ $(P \in \mathcal{M}^H)$-Write($\mathtt{sid}, \mathtt{vid}, m, \mathtt{ReplyTo}$)
Require: **BasicReqs**($\mathtt{sid}, \mathtt{vid}, P$)
  $(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} := (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \mathrm{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
  $\big(\vec{V}, \mathtt{cmd}, \mathrm{Acks}\big) \leftarrow \big(\vec{G}, (\mathtt{vid}, \mathsf{Msg}, m, \mathtt{ReplyTo}), \mathrm{VoteAcks} \cup \mathtt{ReplyTo}\big)$
Require: IsValid($\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$
  Output(**CS**[$\mathfrak{U}$]-Write($\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks}$))

$\triangleright$ $(P \in \mathcal{M}^H)$-Read
  ChatGraphs $\leftarrow \emptyset$
  for $(\mathtt{sid}, \mathcal{G}^+) \in$ **CS**[$\mathfrak{U}$]-Read **with** $\mathsf{VisibleGraph}(\mathtt{sid}, \mathcal{G}^+, P) \neq (\emptyset, \emptyset)$ **:**
    ChatGraphs $\leftarrow$ ChatGraphs $\cup \{(\mathtt{sid}, \mathsf{VisibleGraph}(\mathtt{sid}, \mathcal{G}^+, P))\}$
  Output(ChatGraphs)

$\triangleright$ $(P \in \overline{\mathcal{P}^H})$-Write($\langle S \to \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})$): Output(**CS**[$\mathfrak{U}$]-Write($\langle S \to \vec{V} \rangle, m$))
$\triangleright$ $(P \in \overline{\mathcal{P}^H})$-Read: Output(**CS**[$\mathfrak{U}$]-Read)

$\triangleright$ Deliver($P, \mathtt{id}$): **CS**[$\mathfrak{U}$]-Deliver($P, \mathtt{id}$)

$\diamond$ **BasicReqs**($\mathtt{sid}, \mathtt{vid}, P$)
Require: $\mathtt{sid} \in$ **CS**[$\mathfrak{U}$]-Read
  $\mathcal{G}^+_{\text{vis}} := (V^+_{\text{vis}}, E^+_{\text{vis}}) \leftarrow \mathsf{VisibleGraph}(\mathtt{sid}, \mathbf{CS}[\mathfrak{U}]\text{-Read}[\mathtt{sid}], P)$
Require: $(\mathtt{vid} \in V^+_{\text{vis}}) \wedge (\mathsf{NodeIsRoot}(\mathtt{vid}, V^+_{\text{vis}}) = 1)$

Algorithm 10 Helper functions from **UatChat**'s description. Below, $\mathsf{Sourced}$ is as in Definition 3. For simpler notation we write $\mathbf{CS}[\mathfrak{U}]$ instead of $\mathbf{ChatSessions}[\mathfrak{U}]$.

$\diamond$ $\mathsf{MissingVotes}(\mathtt{vid}, V^+)$
$\quad(\text{Voters}, \text{Votable}, \cdot, \cdot) \leftarrow \mathsf{RootCmdInfo}(\mathsf{CmdOf}(\mathtt{vid}, V^+))$
$\quad$**if** $\text{Votable} = 1$ **:**
$\quad\quad\text{Voted} \leftarrow \{\mathsf{SenderOf}(\mathtt{vid}, V^+)\} \cup \{S \mid \exists(\cdot, (\langle S \rightarrow \vec{R}\rangle, (\cdot, (\mathtt{vid}, \mathsf{Vote}), \cdot))) \in V^+\}$
$\quad$**else**
$\quad\quad\text{Voted} \leftarrow \text{Voters}$
$\quad$**return** $\text{Voters} \setminus \text{Voted}$

$\diamond$ $\mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\quad\mathcal{G}^+ := (V^+, E^+) \leftarrow \mathbf{CS}[\mathfrak{U}]\text{-}\textsc{Read}[\mathtt{sid}]$
$\quad\text{MissingVotes} \leftarrow \mathsf{MissingVotes}(\mathtt{vid}, V^+)$
$\quad(\cdot, \cdot, \vec{G}_{\text{pre-vote}}, \vec{G}_{\text{pos-vote}}) \leftarrow \mathsf{RootCmdInfo}(\mathsf{CmdOf}(\mathtt{vid}, V^+))$
$\quad\mathcal{G}^+_{\text{src-vis}} := (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}) \leftarrow \mathsf{VisibleGraph}(\mathsf{Sourced}(\mathcal{G}^+, \mathtt{vid}), P)$
$\quad\text{VoteNodes} \leftarrow \{\mathtt{id} \mid (\mathtt{id}, (\cdot, (\cdot, (\mathtt{vid}, \mathsf{Vote}), \cdot))) \in V^+_{\text{src-vis}}\}$
$\quad$**return** $(\vec{G}_{\text{pre-vote}}, \vec{G}_{\text{pos-vote}}, \mathcal{G}^+_{\text{src-vis}}, \text{MissingVotes}, \text{VoteNodes})$

$\diamond$ $\mathsf{AckedNodes}(\mathcal{G}^+ := (V^+, E^+), P)$
$\quad V^+_{\text{acked}} \leftarrow V^+$
$\quad$**for** $u := (\mathtt{id}, (\cdot, (\cdot, \mathtt{cmd}, \text{Acks}))) \in V^+$ **with** $\mathsf{NodeIsRoot}(\mathtt{id}, V^+) \wedge (\text{Acks} \not\subseteq V^+)$ **:**
$\quad\quad$**if** $\mathtt{cmd} \neq (\cdot, \mathsf{Add}, \cdot, P)$ **:**
$\quad\quad\quad\text{Compute } \mathcal{G}^+_{\text{src}} := (V^+_{\text{src}}, \cdot) \leftarrow \mathsf{Sourced}(\mathcal{G}^+, \mathtt{id})$
$\quad\quad\quad V^+_{\text{acked}} \leftarrow V^+_{\text{acked}} \setminus V^+_{\text{src}}$
$\quad$**return** V^+_{acked}

$\diamond$ $\mathsf{VisibleGraph}(\mathtt{sid}, \mathcal{G}^+ := (V^+, E^+), P)$
$\quad V^+_{\text{vis}} \leftarrow \mathsf{AckedNodes}(\mathcal{G}^+, P)$
$\quad$**for** $u := (\mathtt{id}, (\langle S \rightarrow \vec{V}\rangle, (\cdot, \mathtt{cmd}, \text{Acks}))) \in V^+_{\text{vis}}$ **with** $\mathsf{NodeIsRoot}(\mathtt{id}, V^+_{\text{vis}})$ **:**
$\quad\quad\text{Compute } \mathcal{G}^+_{\text{src}} := (V^+_{\text{src}}, \cdot) \leftarrow \mathsf{Sourced}(\mathcal{G}^+, \mathtt{id})$
$\quad\quad$**if** $\textsc{IsRoot-Ext}(\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \text{Acks}) = 0$ **:**
$\quad\quad\quad V^+_{\text{vis}} \leftarrow V^+_{\text{vis}} \setminus V^+_{\text{src}}$
$\quad\quad$**else if** $(\mathtt{cmd} = (\cdot, \mathsf{change}, \vec{G}, P')) \wedge (\mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}) \wedge (\mathsf{MissingVotes}(\mathtt{id}, V^+_{\text{vis}}) \neq \emptyset)$ **:**
$\quad\quad\quad V^+_{\text{vis}} \leftarrow V^+_{\text{vis}} \setminus V^+_{\text{src}}$
$\quad\quad\quad$**if** $P' \neq P$ **:**
$\quad\quad\quad\quad V^+_{\text{vis}} \leftarrow V^+_{\text{vis}} \cup \{u\}$
$\quad E^+_{\text{vis}} \leftarrow E^+ \cap (V^+_{\text{vis}} \times V^+_{\text{vis}})$
$\quad$**return** $\mathcal{G}^+_{\text{vis}} := (V^+_{\text{vis}}, E^+_{\text{vis}})$

repositories we assume for the construction of **ChatSessions**[$\mathfrak{P}$], which allows us to obtain an instantiation of our abstraction with all these extra guarantees.

We begin by defining [34]'s MDRS-PKE application semantics; in doing so we introduce their modeling technique, which we use to extend **ChatSessions**[$\mathfrak{P}$]'s guarantees. Next we model each additional guarantee and show our construction preserves them. Finally, we also explain why UatChat also preserves these guarantees as well.

6.1 Application Semantics of Multi-Designated Receiver Signed Public Key Encryption [34]

In the following, we consider a set of senders $\mathcal{S} = \{A_1, \ldots, A_l\}$, and a set of receivers $\mathcal{R} = \{B_1, \ldots, B_n\}$; we assume $\mathcal{R}^H$, $\overline{\mathcal{R}^H}$, $\mathcal{S}^H$ and $\overline{\mathcal{S}^H}$ are all non-empty. We also consider a set $\mathcal{F}$ that includes all senders and receivers, i.e. $\mathcal{F} \coloneqq \mathcal{S} \cup \mathcal{R}$. Finally, we consider a judge J(-udy) who is not a sender nor a receiver. The set of parties is $\mathcal{P} = \{A_1, \ldots, A_l, B_1, \ldots, B_n, J\}$.

The MDRS-PKE model from [34] provides different application semantics depending on the honesty of the judge J(-udy): if dishonest, their model provides consistency, Off-The-Record, confidentiality and anonymity; if honest, it additionally provides authenticity. Their model also provides confidentiality and anonymity for messages sent by honest senders to vectors of all-honest receivers.

Remark 2 (Authenticity and J's honesty). The reason why the model from [34] only provides authenticity for honest J is that they consider the setting from [20] where J is given access to the secret keys of honest senders (which in particular means she can impersonate them). On the other hand, if J is honest then she is not given access to the secret keys of honest senders; this is the only case in which authenticity may be possible.

6.1.1 Application Semantics for MDRS-PKE [34]

The MDRS-PKE model from [34] is defined upon the repository model we introduced in Section 3.2. Their application semantics include, for each sender $A_i \in \mathcal{S}$ and vector of receivers $\vec{V} \in \mathcal{R}^+$, a repository

$$\langle A_i \to \vec{V} \rangle^{\{A_i\} \cup \overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}}$$

to which A_i and any dishonest party can write to, and from which dishonest parties and the ones in $\vec{V}$ can read. Note that this naturally captures a stateless consistency guarantee because for each repository $\langle A_i \to \vec{V} \rangle$, either there is a register with identifier `id`—in which case each $B_j \in \vec{V}$ gets the same tuple upon a Read operation—or there is not—in which case no $B_j \in \vec{V}$ obtains a tuple with identifier `id`.

Off-The-Record. Their application semantics captures Off-The-Record by including, for each sender A_i and receiver vector $\vec{V}$, an additional repository $\langle[\text{Forge}]A_i \to \vec{V}\rangle$ to which parties from $\mathcal{F}$ write forged (i.e. "fake") messages; the readers of these repositories are only the dishonest parties because honest ones only read real (non-forged) messages; this means $\langle[\text{Forge}]A_i \to \vec{V}\rangle :=$ $\langle[\text{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}$. Put together with the repositories from above and attaching the network converter Net:[20]

$$\begin{bmatrix} \mathsf{Net} \cdot \left[\langle A_i \to \vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{\text{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \\ \left[\langle[\text{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix}. \tag{6.1}$$

Recall from the repository semantics (Algorithm 3) that READ operations output the atomic repository associated with each message output. This means that when a dishonest party reads from the resource above it still learns which messages are real ones—i.e. written to a repository $\langle A_i \to \vec{V}\rangle$—and which ones are "fake"—i.e. written to a repository $\langle[\text{Forge}]A_i \to \vec{V}\rangle$. To avoid this, they introduce a converter Otr [34] (Algorithm 11) which connects to the dishonest parties' READ interfaces of the resource above and hides from them where each message comes from.

Algorithm 11 Converter Otr from [34].

```
▷ (P ∈ \overline{P^H})-READ
  list ← ∅
  for (id, (⟨[Forge]A_i → V⃗⟩, m)) ∈ READ : list ← list ∪ {(id, (⟨A_i → V⃗⟩, m))}
  for (id, (⟨A_i → V⃗⟩, m)) ∈ READ : list ← list ∪ {(id, (⟨A_i → V⃗⟩, m))}
  OUTPUT(list)
```

Confidentiality and Anonymity. Their approach to capturing confidentiality and anonymity is similar: they define a converter $\mathsf{ConfAnon}$ (Algorithm 12) that limits dishonest parties reading capabilities [34].

Algorithm 12 Converter $\mathsf{ConfAnon}$ from [34].

```
▷ (P ∈ \overline{P^H})-READ
  list ← ∅
  for (id, (⟨A_i → V⃗⟩, m)) ∈ READ with {A_i} ∪ Set(V⃗) ⊆ P^H : list ← list ∪ {(id, (|V⃗|, |m|))}
  for (id, (⟨A_i → V⃗⟩, m)) ∈ READ with {A_i} ∪ Set(V⃗) ⊈ P^H : list ← list ∪ {(id, (⟨A_i → V⃗⟩, m))}
  OUTPUT(list)
```

Application Semantics for Dishonest J(-udy). Putting things together, their MDRS-PKE application semantics for the case of a dishonest judge J are given

[20] As noted in [34], Net need not be attached to $\langle[\text{Forge}]A_i \to \vec{V}\rangle$ because the readers are dishonest.

by the ideal resource $\mathbf{S}$ below [34]:

$$\mathbf{S} \coloneqq \left(\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \ \mathsf{Otr}^{\overline{\mathcal{P}^H}}\right) \cdot \begin{bmatrix} \mathsf{Net} \ \cdot \ \left[\langle A_i \to \vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \\ \left[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix}. \tag{6.2}$$

Application Semantics for Honest J(-udy). Their application semantics for the case of an honest judge is similar. They capture authenticity by removing the Write sub-interfaces that dishonest parties could use to write on behalf of honest senders [34]. Concretely, denoting these (sub-)interfaces by

$$\mathsf{Auth\text{-}Intf} \coloneqq \overline{\mathcal{P}^H}\text{-}\textsc{Write}(\langle \mathcal{S}^H \to \mathcal{R}^+\rangle, \cdot), \tag{6.3}$$

their application semantics are given by the following ideal resource $\mathbf{T}$:

$$\mathbf{T} \coloneqq \begin{pmatrix} \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \\ \cdot\, \mathsf{Otr}^{\overline{\mathcal{P}^H}} \end{pmatrix} \cdot \begin{bmatrix} \mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \left[\langle A_i \to \vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{\substack{A_i\in\mathcal{S}\\ \vec{V}\in\mathcal{R}^+}} \\ \left[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix} \tag{6.4}$$

where $\perp$ is a dummy converter which provides no output interface; attaching $\perp$ to $\mathsf{Auth\text{-}Intf}$ disables the interface that dishonest parties could use to write on behalf of honest ones.

6.2 Extending ChatSessions[$\mathfrak{P}$] to Provide Extra Guarantees

Having introduce the MDRS-PKE application semantics from [34], we are now set to extend the guarantees captured by **ChatSessions**[$\mathfrak{P}$]. We will prove that each of these guarantees is preserved.

6.2.1 Authenticity. We extend **ChatSessions**[$\mathfrak{P}$] to provide authenticity following the same technique from [34], i.e. by attaching converter $\perp^{\mathsf{Auth\text{-}Intf}}$ so dishonest parties cannot impersonate honest ones. ($\mathsf{Auth\text{-}Intf}$ are the Write sub-interfaces defined in Equation 6.3.) The ideal system is then

$$\mathbf{AuthChatSessions}[\mathfrak{P}] \coloneqq \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{ChatSessions}[\mathfrak{P}]. \tag{6.5}$$

The real world is as in Equation 4.2 with converter $\perp$ attached to interfaces $\mathsf{Auth\text{-}Intf} \coloneqq \overline{\mathcal{P}^H}\text{-}\textsc{Write}(\langle \mathcal{S}^H \to \mathcal{R}^+\rangle, \cdot)$ of **REP**:

$$\mathbf{R_{Auth}}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}). \tag{6.6}$$

Corollary 1 follows from Theorem 1.

Corollary 1. *For any $\mathfrak{P}$ satisfying Requirements 1, 2 and 3:*

$$\mathbf{R_{Auth}}[\mathfrak{P}] \equiv \mathbf{AuthChatSessions}[\mathfrak{P}].$$

Proof.

$$
\begin{aligned}
\mathbf{AuthChatSessions}[\mathfrak{P}] &= \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{ChatSessions}[\mathfrak{P}] \\
&\equiv \perp^{\mathsf{Auth\text{-}Intf}} \cdot (\mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AREP}) && (1) \\
&= \perp^{\mathsf{Auth\text{-}Intf}} \cdot (\mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP})) \\
&\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathsf{Net} \cdot \mathbf{REP}) && (2) \\
&\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}) && (3) \\
&= \mathbf{R_{Auth}}[\mathfrak{P}].
\end{aligned}
$$

(1): From Theorem 1;
(2): Commutativity of converter application at disjoint interfaces;
(3): By Equation 6.7 (see below).

It only remains to prove

$$\perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathsf{Net} \cdot \mathbf{REP} \equiv \mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}. \tag{6.7}$$

Converter $\perp$ disables the interfaces it is attached to. Attaching $\perp^{\mathsf{Auth\text{-}Intf}}$ to $\mathsf{Net} \cdot \mathbf{REP}$ disallows dishonest parties from issuing Write operations for labels $\langle S \to \vec{V} \rangle$ with $S \in \mathcal{S}^H$ (since $\mathsf{Auth\text{-}Intf} \coloneqq \overline{\mathcal{P}^H}\text{-}\textsc{Write}(\langle \mathcal{S}^H \to \mathcal{R}^+ \rangle, \cdot)$). The definition of converter Net depends on the repositories to which it connects (Algorithm 3); in particular it only allows a party P to issue a Write operation for a repository $\mathbf{rep_i} \coloneqq \mathbf{rep_i}_{\mathcal{R}_i}^{\mathcal{W}_i}$ if $P \in \mathcal{W}_i$, i.e. if P has write permissions—because the description of Net specifies that the party's interface of Net at which the Write operation was issued matches the one that Net uses to issue the corresponding Write operation to the repository. This then implies Equation 6.7.

□

Algorithm 13 The **FakeChatSessions** system to which fake messages (i.e. invisible to honest parties) are written. Below, **FAKE-REP** $\coloneqq [\langle [\text{Forge}]P \to \vec{V} \rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+}$.

▷ $(P \in \mathcal{M})$-Write$(S, \texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  Output(**FAKE-REP**-Write$(\langle [\text{Forge}]S \to \vec{V} \rangle, m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks}))$)

▷ $(P \in \overline{\mathcal{P}^H})$-Read: Output(**FAKE-REP**-Read)

6.2.2 Off-The-Record. As for authenticity, we follow the same modeling technique from [34]. Concretely, we extend **AuthChatSessions**$[\mathfrak{P}]$ via parallel composition with **FakeChatSessions**—defined in Algorithm 13—which provides 1. an interface Write that allows parties to write fake messages, and 2. an interface Read from which dishonest parties can read these fake messages—and then attach converter Otr (Algorithm 11) to the interfaces of dishonest parties that hides (from dishonest parties) which messages are real—i.e. written to **AuthChatSessions**[$\mathfrak{P}$]—and which ones are fake—not visible to honest parties, i.e. written to **FakeChatSessions**. The ideal world is then

$$\mathbf{OTR\text{-}ChatSessions}[\mathfrak{P}] \coloneqq \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathbf{AuthChatSessions}[\mathfrak{P}] \\ \mathbf{FakeChatSessions} \end{bmatrix}. \tag{6.8}$$

Algorithm 14 Converter $\mathsf{ChatSessionsForgeProt}$.

$\triangleright$ $(P \in \mathcal{M})$-$\textsc{Write}(S, \mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks})$
  $\textsc{Output}\Big(([\langle[\mathrm{Forge}]P \to \vec{R}\rangle]_{P\in\mathcal{M}, \vec{R}\in\mathcal{M}^+})$-$\textsc{Write}(\langle[\mathrm{Forge}]S \to \vec{V}\rangle, m \coloneqq (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))\Big)$

The assumed resources are similar to the ones for authenticity (Equation 6.6), but now also include repositories $\left[\langle[\mathrm{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{P\in\mathcal{M}, \vec{V}\in\mathcal{M}^+}$ to which parties write fake messages, plus converter Otr. Regarding the protocol, honest parties $\mathcal{M}^H$ run converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$, and additionally all (honest and dishonest) parties in $\mathcal{M}$ run converter $\mathsf{ChatSessionsForgeProt}$ (Algorithm 14) which allows writing fake messages. The real world resource is then

$$\mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}] \coloneqq \begin{pmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \\ \cdot\, \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \end{pmatrix} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \left[\langle[\mathrm{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{\substack{P\in\mathcal{M} \\ \vec{V}\in\mathcal{M}^+}} \end{bmatrix}. \tag{6.9}$$

Corollary 2 follows from Corollary 1.

Corollary 2. *For any $\mathfrak{P}$ satisfying Requirements 1, 2 and 3:*

$$\mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}] \equiv \mathbf{OTR\text{-}ChatSessions}[\mathfrak{P}].$$

Proof. Consider the definitions of **FakeChatSessions** (Algorithm 13), of protocol $\mathsf{ChatSessionsForgeProt}$ (Algorithm 14) and of $\left[\langle[\mathrm{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{P\in\mathcal{M}, \vec{V}\in\mathcal{M}^+}$ (Algorithm 2). We have

$$\mathbf{FakeChatSessions} \equiv \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \cdot \left[\langle[\mathrm{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{P\in\mathcal{M}, \vec{V}\in\mathcal{M}^+}. \tag{P.1}$$

It then follows

$$\begin{aligned}
&\mathbf{OTR\text{-}ChatSessions}[\mathfrak{P}] = \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathbf{AuthChatSessions}[\mathfrak{P}] \\ \mathbf{FakeChatSessions} \end{bmatrix}. \\
&\equiv \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathbf{R_{Auth}}[\mathfrak{P}] \\ \mathbf{FakeChatSessions} \end{bmatrix} && (1) \\
&\equiv \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}) \\ \mathbf{FakeChatSessions} \end{bmatrix} \\
&\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \mathbf{FakeChatSessions} \end{bmatrix} && (2) \\
&\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \\
&\quad \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \cdot \left[\langle [\mathrm{Forge}] P \to \vec{V} \rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}} \right]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+} \end{bmatrix} && (3) \\
&\equiv \begin{pmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \\ \cdot \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \end{pmatrix} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \left[\langle [\mathrm{Forge}] P \to \vec{V} \rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}} \right]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+} \end{bmatrix} && (4) \\
&= \mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}].
\end{aligned}$$

(1): Corollary 1;
(2): Commutativity of converter application at disjoint interfaces;
(3): By Equation P.1;
(4): Commutativity of converter application at disjoint interfaces.

□

6.2.3 Confidentiality and Anonymity. Finally, we also follow [34] to capture confidentiality and anonymity, i.e. capture these guarantees via converter ConfAnon (Algorithm 12); consider any two resources $\mathbf{AR}[\mathfrak{P}]$ and $\mathbf{V}[\mathfrak{P}]$ such that

$$\mathbf{V}[\mathfrak{P}] \equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AR}[\mathfrak{P}] \tag{6.10}$$

which have $\overline{\mathcal{P}^H}$-Read interfaces suitable for converter ConfAnon. ($\mathbf{V}[\mathfrak{P}]$ could be, e.g. $\mathbf{ChatSessions}[\mathfrak{P}]$, $\mathbf{AuthChatSessions}[\mathfrak{P}]$ or $\mathbf{OTR\text{-}ChatSessions}[\mathfrak{P}]$.) The ideal resource capturing confidentiality and anonymity is

$$\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{V}[\mathfrak{P}]. \tag{6.11}$$

The real world resource is

$$\mathbf{R}_{\mathsf{ConfAnon}}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{AR}[\mathfrak{P}]), \tag{6.12}$$

where $(\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{AR}[\mathfrak{P}])$ is the assumed resource for the construction. The following then establishes our claim that if the real world resource gives confidentiality and anonymity guarantees, then so does the corresponding ideal world.

(We state Corollary 3 abstractly because we want the result to hold for any suitable real world and ideal world resources.)

Corollary 3. *For any* $\mathfrak{P}$ *satisfying Requirements 1, 2 and 3 and any resources* $\mathbf{AR}[\mathfrak{P}]$ *and* $\mathbf{V}[\mathfrak{P}]$ *satisfying Equation 6.10 that have* $\overline{\mathcal{P}^H}$-Read *interfaces suitable for converter* ConfAnon *(Algorithm 12),*

$$\mathbf{R}_{\mathsf{ConfAnon}}[\mathfrak{P}] \equiv \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{V}[\mathfrak{P}].$$

Proof.

$$\begin{aligned}\mathbf{R}_{\mathsf{ConfAnon}}[\mathfrak{P}] &= \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{AR}[\mathfrak{P}]) \\ &\equiv \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot (\mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AR}[\mathfrak{P}]) && (1) \\ &\equiv \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot (\mathbf{V}[\mathfrak{P}]). && (2)\end{aligned}$$

(1): Commutativity of converter application at disjoint interfaces;
(2): Assumption stated in Equation 6.10. □

6.3 Uatchat

One can capture authenticity, confidentiality, anonymity and Off-The-Record analogously to how we captured these guarantees for **ChatSessions**[$\mathfrak{P}$]; corollaries analogous to Corollaries 1, 2 and 3 also hold for **UatChat** (and also follow trivially from the commutativity of converter application at disjoint interfaces). Regarding Off-The-Record, in Algorithm 15 we define **FakeUatChat** to which parties write fake commands; as for **ChatSessions**[$\mathfrak{P}$], the ideal **OTR-UatChat** is then the parallel composition of **UatChat** and **FakeUatChat** with converter Otr attached.

Algorithm 15 System **FakeUatChat**.

▷ $(P \in \mathcal{M})$-FakeCreateChat$(S, \mathtt{sid}, \vec{G} \in \mathcal{M}^+)$
$\left(\vec{V}, \mathtt{cmd}, \text{Acks}\right) \leftarrow \left(\vec{G}, (\mathsf{Create}, \vec{G}), \emptyset\right)$
Output(**FakeChatSessions**-Write$(S, \mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$)

▷ $(P \in \mathcal{M})$-FakeProposeChange$(S, \mathtt{sid}, \mathtt{vid}, \mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}, P' \in \mathcal{M})$
$(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} \coloneqq (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\vec{G}' \leftarrow (\vec{G} \mid\mid P')$
$\text{LeafAcks} \leftarrow \{\mathtt{id} \mid (\exists(\mathtt{id}, (\cdot, (\cdot, (\mathtt{vid}, \cdot), \cdot))) \in V^+_{\text{src-vis}}) \wedge (\nexists(\mathtt{id}, \cdot) \in E^+_{\text{src-vis}})\}$
$\left(\vec{V}, \mathtt{cmd}, \text{Acks}\right) \leftarrow \left(\vec{G}', (\mathtt{vid}, \mathsf{change}, \vec{G}, P'), \text{VoteAcks} \cup \text{LeafAcks}\right)$
Output(**FakeChatSessions**-Write$(S, \mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$)

▷ $(P \in \mathcal{M})$-FakeVote$(S, \mathtt{sid}, \mathtt{vid})$
$(\vec{G}, \cdot, \mathcal{G}^+_{\text{src-vis}}, \text{MissingVotes}, \cdot) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\left(\vec{V}, \mathtt{cmd}, \text{Acks}\right) \leftarrow \left(\vec{G}, (\mathtt{vid}, \mathsf{Vote}), \{\mathtt{vid}\}\right)$
Output(**FakeChatSessions**-Write$(S, \mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$)

▷ $(P \in \mathcal{M})$-FakeWrite$(S, \mathtt{sid}, \mathtt{vid}, m, \mathtt{ReplyTo})$
$(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} \coloneqq (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\left(\vec{V}, \mathtt{cmd}, \text{Acks}\right) \leftarrow \left(\vec{G}, (\mathtt{vid}, \mathsf{Msg}, m, \mathtt{ReplyTo}), \text{VoteAcks} \cup \mathtt{ReplyTo}\right)$
Output(**FakeChatSessions**-Write$(S, \mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$)

References

1. Signal Messenger: Speak Freely — signal.org. https://signal.org/, [Accessed 02-10-2024]
2. WhatsApp — Secure and Reliable Free Private Messaging and Calling — whatsapp.com. https://www.whatsapp.com/, [Accessed 02-10-2024]
3. Alwen, J., Auerbach, B., Noval, M.C., Klein, K., Pascual-Perez, G., Pietrzak, K., Walter, M.: CoCoA: Concurrent continuous group key agreement. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 815–844. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_28
4. Alwen, J., Auerbach, B., Noval, M.C., Klein, K., Pascual-Perez, G., Pietrzak, K.: DeCAF: Decentralizable CGKA with fast healing. In: Galdi, C., Phan, D.H. (eds.) SCN 24, Part II. LNCS, vol. 14974, pp. 294–313. Springer, Cham (Sep 2024). https://doi.org/10.1007/978-3-031-71073-5_14
5. Alwen, J., Coretti, S., Dodis, Y.: The double ratchet: Security notions, proofs, and modularization for the Signal protocol. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 129–158. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17653-2_5
6. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Security analysis and improvements for the IETF MLS standard for group messaging. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 248–277. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_9
7. Alwen, J., Coretti, S., Dodis, Y., Tselekounis, Y.: Modular design of secure group messaging protocols and the security of MLS. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 1463–1483. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484820
8. Alwen, J., Coretti, S., Jost, D., Mularczyk, M.: Continuous group key agreement with active security. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 261–290. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_10
9. Alwen, J., Hartmann, D., Kiltz, E., Mularczyk, M.: Server-aided continuous group key agreement. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 69–82. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560632
10. Alwen, J., Jost, D., Mularczyk, M.: On the insider security of MLS. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 34–68. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_2
11. Alwen, J., Mularczyk, M., Tselekounis, Y.: Fork-resilient continuous group key agreement. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 396–429. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_13
12. Attiya, H., Guerraoui, R., Hendler, D., Kuznetsov, P., Michael, M.M., Vechev, M.T.: Laws of order: expensive synchronization in concurrent algorithms cannot be eliminated. In: Ball, T., Sagiv, M. (eds.) Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011. pp. 487–498. ACM (2011). https://doi.org/10.1145/1926385.1926442, https://doi.org/10.1145/1926385.1926442
13. Balbás, D., Collins, D., Vaudenay, S.: Cryptographic administration for secure group messaging. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 1253–1270. USENIX Association (Aug 2023)

14. Barnes, R., Beurdouche, B., Robert, R., Millican, J., Omara, E., Cohn-Gordon, K.: The Messaging Layer Security (MLS) Protocol. RFC 9420 (Jul 2023). https://doi.org/10.17487/RFC9420, https://www.rfc-editor.org/info/rfc9420
15. Bellare, M., Boldyreva, A., Staddon, J.: Randomness re-use in multi-recipient encryption schemeas. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 85–99. Springer, Berlin, Heidelberg (Jan 2003). https://doi.org/10.1007/3-540-36288-6_7
16. Bhargavan, K., Barnes, R., Rescorla, E.: TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris (May 2018), https://inria.hal.science/hal-02425247
17. Bienstock, A., Fairoze, J., Garg, S., Mukherjee, P., Raghuraman, S.: A more complete analysis of the Signal double ratchet algorithm. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 784–813. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15802-5_27
18. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
19. Canetti, R., Jain, P., Swanberg, M., Varia, M.: Universally composable end-to-end secure messaging. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 3–33. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_1
20. Chakraborty, S., Hofheinz, D., Maurer, U., Rito, G.: Deniable authentication when signing keys leak. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 69–100. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30620-4_3
21. Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., Milner, K.: On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1802–1819. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243747
22. Cong, K., Eldefrawy, K., Smart, N.P., Terner, B.: The key lattice framework for concurrent group messaging. In: Pöpper, C., Batina, L. (eds.) ACNS 24International Conference on Applied Cryptography and Network Security, Part II. LNCS, vol. 14584, pp. 133–162. Springer, Cham (Mar 2024). https://doi.org/10.1007/978-3-031-54773-7_6
23. Damgård, I., Haagh, H., Mercer, R., Nitulescu, A., Orlandi, C., Yakoubov, S.: Stronger security and constructions of multi-designated verifier signatures. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 229–260. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_9
24. Devigne, J., Duguey, C., Fouque, P.A.: MLS group messaging: How zero-knowledge can secure updates. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021, Part II. LNCS, vol. 12973, pp. 587–607. Springer, Cham (Oct 2021). https://doi.org/10.1007/978-3-030-88428-4_29
25. Fiat, A., Naor, M.: Broadcast encryption. In: Stinson, D.R. (ed.) CRYPTO'93. LNCS, vol. 773, pp. 480–491. Springer, Berlin, Heidelberg (Aug 1994). https://doi.org/10.1007/3-540-48329-2_40
26. Herlihy, M., Shavit, N.: The art of multiprocessor programming. Morgan Kaufmann (2008)
27. Herlihy, M., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst. **12**(3), 463–492 (1990). https://doi.org/10.1145/78969.78972, https://doi.org/10.1145/78969.78972

28. Jost, D., Maurer, U.: Overcoming impossibility results in composable security using interval-wise guarantees. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 33–62. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_2
29. Jost, D., Maurer, U., Mularczyk, M.: Efficient ratcheting: Almost-optimal guarantees for secure messaging. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 159–188. Springer, Cham (May 2019). https://doi.org/10.1007/978-3-030-17653-2_6
30. Jost, D., Maurer, U., Mularczyk, M.: A unified and composable take on ratcheting. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 180–210. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_7
31. Klein, K., Pascual-Perez, G., Walter, M., Kamath, C., Capretto, M., Cueto, M., Markov, I., Yeo, M., Alwen, J., Pietrzak, K.: Keep the dirt: Tainted TreeKEM, adaptively and actively secure continuous group key agreement. In: 2021 IEEE Symposium on Security and Privacy. pp. 268–284. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00035
32. Kurosawa, K.: Multi-recipient public-key encryption with shortened ciphertext. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 48–63. Springer, Berlin, Heidelberg (Feb 2002). https://doi.org/10.1007/3-540-45664-3_4
33. Liu-Zhang, C.D., Maurer, U.: Synchronous constructive cryptography. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 439–472. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64378-2_16
34. Liu-Zhang, C.D., Portmann, C., Rito, G.: Simpler and stronger models for deniable authentication. Cryptology ePrint Archive, Report 2025/204 (2025), https://eprint.iacr.org/2025/204
35. Matt, C., Maurer, U.: The one-time pad revisited. In: Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013. pp. 2706–2710. IEEE (2013). https://doi.org/10.1109/ISIT.2013.6620718, https://doi.org/10.1109/ISIT.2013.6620718
36. Maurer, U.: Constructive cryptography—a new paradigm for security definitions and proofs. In: Proceedings of Theory of Security and Applications, TOSCA 2011. Lecture Notes in Computer Science, vol. 6993, pp. 33–56. Springer (2012). https://doi.org/10.1007/978-3-642-27375-9_3
37. Maurer, U., Portmann, C., Rito, G.: Giving an adversary guarantees (or: How to model designated verifier signatures in a composable framework). In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 189–219. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_7
38. Maurer, U., Portmann, C., Rito, G.: Multi-designated receiver signed public key encryption. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 644–673. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_22
39. Maurer, U., Renner, R.: Abstract cryptography. In: Chazelle, B. (ed.) ICS 2011. pp. 1–21. Tsinghua University Press (Jan 2011)
40. Maurer, U.M.: Indistinguishability of random systems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 110–132. Springer, Berlin, Heidelberg (Apr / May 2002). https://doi.org/10.1007/3-540-46035-7_8
41. Maurer, U.M., Pietrzak, K., Renner, R.: Indistinguishability amplification. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 130–149. Springer, Berlin, Heidelberg (Aug 2007). https://doi.org/10.1007/978-3-540-74143-5_8

42. Papadimitriou, C.H.: The serializability of concurrent database updates. J. ACM **26**(4), 631–653 (1979). https://doi.org/10.1145/322154.322158, https://doi.org/10.1145/322154.322158
43. Wallez, T., Protzenko, J., Beurdouche, B., Bhargavan, K.: TreeSync: Authenticated group management for messaging layer security. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 1217–1233. USENIX Association (Aug 2023)
44. Weidner, M.: Group messaging for secure asynchronous collaboration. Master's thesis (2019)
45. Weidner, M., Kleppmann, M., Hugenroth, D., Beresford, A.R.: Key agreement for decentralized secure group messaging with strong security guarantees. In: Vigna, G., Shi, E. (eds.) ACM CCS 2021. pp. 2024–2045. ACM Press (Nov 2021). https://doi.org/10.1145/3460120.3484542

Appendix

A Definition of UatChatProt (Algorithm 16)

Algorithm 16 Description of the $\mathsf{UatChatProt}$ converter run by each honest party $P \in \mathcal{M}^H$ for constructing **UatChat** (see Algorithm 9). We rely on the helper functions from Algorithm 10.

$\textsc{CreateChat}(\mathtt{sid}, \vec{G} \in \mathcal{M}^+)$
Require: $\mathtt{sid} \notin \mathsf{UatChatProt}\text{-}\textsc{Read}$
$\big(\vec{V}, \mathtt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}, (\mathsf{Create}, \vec{G}), \emptyset\big)$
Require: $\textsc{IsRoot-Ext}(\mathtt{sid}, (\emptyset, \emptyset), P, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$
$\textsc{Output}(\mathbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Write}(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks}))$

$\textsc{ProposeChange}(\mathtt{sid}, \mathtt{vid}, \mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}, P' \in \mathcal{M})$
Require: $\mathbf{BasicRequirements}(\mathtt{sid}, \mathtt{vid}, P)$
$(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} := (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\vec{G}' \leftarrow (\vec{G} \mid\mid P')$
$\text{LeafAcks} \leftarrow \{\mathtt{id} \mid (\exists(\mathtt{id}, (\cdot, (\cdot, (\mathtt{vid}, \cdot), \cdot))) \in V^+_{\text{src-vis}}) \wedge (\nexists(\mathtt{id}, \cdot) \in E^+_{\text{src-vis}})\}$
$\big(\vec{V}, \mathtt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}', (\mathtt{vid}, \mathsf{change}, \vec{G}, P'), \text{VoteAcks} \cup \text{LeafAcks}\big)$
Require: $\textsc{IsRoot-Ext}(\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$
$\textsc{Output}(\mathbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Write}(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks}))$

$\textsc{Vote}(\mathtt{sid}, \mathtt{vid})$
Require: $\mathbf{BasicRequirements}(\mathtt{sid}, \mathtt{vid}, P)$
$(\vec{G}, \cdot, \mathcal{G}^+_{\text{src-vis}}, \text{MissingVotes}, \cdot) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
Require: $P \in \text{MissingVotes}$
$\big(\vec{V}, \mathtt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}, (\mathtt{vid}, \mathsf{Vote}), \{\mathtt{vid}\}\big)$
Require: $\textsc{IsValid}(\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$
$\textsc{Output}(\mathbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Write}(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks}))$

$\textsc{Write}(\mathtt{sid}, \mathtt{vid}, m, \mathtt{ReplyTo})$
Require: $\mathbf{BasicRequirements}(\mathtt{sid}, \mathtt{vid}, P)$
$(\cdot, \vec{G}, \mathcal{G}^+_{\text{src-vis}} := (V^+_{\text{src-vis}}, E^+_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \mathsf{HelperFunction}(P, \mathtt{sid}, \mathtt{vid})$
$\big(\vec{V}, \mathtt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}, (\mathtt{vid}, \mathsf{Msg}, m, \mathtt{ReplyTo}), \text{VoteAcks} \cup \mathtt{ReplyTo}\big)$
Require: $\textsc{IsValid}(\mathtt{sid}, \mathcal{G}^+_{\text{src-vis}}, P, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$
$\textsc{Output}(\mathbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Write}(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks}))$

$\textsc{Read}$
$\text{ChatGraphs} \leftarrow \emptyset$
for $(\mathtt{sid}, \mathcal{G}^+) \in \mathbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Read}$ **with** $\mathsf{VisibleGraph}(\mathtt{sid}, \mathcal{G}^+, P) \neq (\emptyset, \emptyset)$ **:**
$\quad\text{ChatGraphs} \leftarrow \text{ChatGraphs} \cup \{(\mathtt{sid}, \mathsf{VisibleGraph}(\mathtt{sid}, \mathcal{G}^+, P))\}$
$\textsc{Output}(\text{ChatGraphs})$

B Proof of Theorem 1

B.1 Proof Structure

We will proceed via two sequences of hybrids, one starting from the real world system $\mathbf{R}[\mathfrak{P}]$ (Equation 4.2), defined $\mathbf{R}[\mathfrak{P}] := \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP})$:

$$\mathbf{R}[\mathfrak{P}] \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{1}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{2}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{3}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{4}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{5}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{6}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}},$$

and the other from the ideal $\mathbf{ChatSessions}[\mathfrak{P}]$ resource

$$\mathbf{ChatSessions}[\mathfrak{P}] \rightsquigarrow \mathbf{H}_{\mathbf{1}}^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_{\mathbf{2}}^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_{\mathbf{3}}^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_{\mathbf{4}}^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}.$$

The last hop of the proof is then $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}} \rightsquigarrow \mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$. More concretely, our proof will establish that all of these are *statistically* the same, i.e.

$$\begin{aligned}\mathbf{R}[\mathfrak{P}] &\equiv \mathbf{H}_{\mathbf{1}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{2}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{3}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{4}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{5}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{6}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}} \\ &\equiv \mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}} \equiv \mathbf{H}_{\mathbf{4}}^{\mathbf{IW}} \equiv \mathbf{H}_{\mathbf{3}}^{\mathbf{IW}} \equiv \mathbf{H}_{\mathbf{2}}^{\mathbf{IW}} \equiv \mathbf{H}_{\mathbf{1}}^{\mathbf{IW}} \equiv \mathbf{ChatSessions}[\mathfrak{P}].\end{aligned}$$

B.2 Helper Propositions

We now establish some useful propositions.

Proposition 1. *Consider any proper graph $\mathcal{G}^+ = (V^+, E^+)$. For any (extended) node $u := (\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))) \in V^+$:*

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1.$$

See Section B.4.1 for the proof of Proposition 1.

Proposition 2. *Consider any proper extended graph $\mathcal{G}^+ = (V^+, E^+)$. Consider function f that maps extended nodes $u := (\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))) \in V^+$ to sets of edges, defined as $f(u) := \mathrm{Acks} \times \{\mathtt{id}\}$. Then,*

$$E^+ = \bigcup_{u \in V^+} f(u).$$

See Section B.4.2 for the proof of Proposition 2.

Proposition 3. *Consider any proper extended graph $\mathcal{G}_0^+ = (V_0^+, E_0^+)$. If the corresponding non-extended $\mathcal{G}_0$ is input to* $\mathsf{UpdatedGraph}$ *(Algorithm 6), then the extended version of each intermediate graph $\mathcal{G}_i$ computed in* $\mathsf{UpdatedGraph}$ *is proper, and so is the extended version of the graph that is output.*

See Section B.4.3 for the proof of Proposition 3.

Proposition 4. *In* $\mathbf{ChatSessions}[\mathfrak{P}]$ *(Algorithm 6), if the extended version of graph $\mathcal{G} = (V, E)$ on which* $\mathsf{InducedPartyGraph}^+$ *computes is proper, then the output extended graph is proper.*

See Section B.4.4 for the proof of Proposition 4.

The following is a direct consequence of Proposition 4.

Corollary 4. *In* $\mathbf{ChatSessions}[\mathfrak{P}]$ *(Algorithm 6), if the extended version of every graph stored in* SessionGraphs *is proper, then for every $P \in \mathcal{P}^H$ and for any* $\mathtt{sid}$*, the extended graph output by* $\mathsf{InducedPartyGraph}^+$ *is proper.*

Proposition 5. *In* $\mathbf{ChatSessions}[\mathfrak{P}]$*, the extended versions of the graphs in* SessionGraphs *are proper.*

See Section B.4.5 for the proof of Proposition 5.

Proposition 6. *In* $\mathsf{ChatSessionsProt}[\mathfrak{P}]$*, the extended versions of the graphs in* SessionGraphs *are proper.*

See Section B.4.6 for the proof of Proposition 6.

Proposition 7. *In* $\mathbf{H_4^{RW}}$*, for each* $\mathtt{sid}$ *and* $P \in \mathcal{M}^H$*,* $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ *is proper.*

See Section B.4.7 for the proof of Proposition 7.

Proposition 8. *Consider some proper graph* $\mathcal{G}$ *and set of nodes* S*, and let*

$$(\mathcal{G}', S') \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S).$$

Then $S' = \mathcal{G}'.V \cap S$*.*

See Section B.4.8 for the proof of Proposition 8.

Proposition 9. *Consider some proper graph* $\mathcal{G} = (V, E)$ *and set of nodes* S*. Let*

$$(\mathcal{G}_S, \cdot) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S)$$

and for any set $V_S \subseteq V$*, let*

$$(\mathcal{G}_{V_S}, \cdot) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S \cup V_S).$$

Then $\mathcal{G}_S = \mathcal{G}_{V_S}$*.*

See Section B.4.9 for the proof of Proposition 9.

Proposition 10. *Consider any proper extended graph* $\mathcal{G}^+ = (V^+, E^+)$ *and any set* S *of nodes such that* $(S \cup V) \subseteq$ *Contents. For any positive* $n \in \mathbb{N}$*, consider any* n *sets* $S_1, \ldots, S_n$ *such that*

$$S = \bigcup_{i=1,\ldots,n} S_i.$$

Let

$$\mathcal{G}_1 \coloneqq \mathcal{G},$$
$$S'_1 \coloneqq S_1,$$

for $i = 1, \ldots, n$*, let*

$$(\mathcal{G}_{i+1}, S''_{i+1}) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}_i, S'_i),$$
$$S'_{i+1} \coloneqq S_{i+1} \cup (S'_i \setminus S''_{i+1}),$$

and let

$$S'' \coloneqq \bigcup_{i \in \{1,\ldots,n\}} S''_{i+1}.$$

Then,

$$(\mathcal{G}_{n+1}, S'') = \mathsf{UpdatedGraph}(\mathcal{G}, S).$$

See Section B.4.10 for the proof of Proposition 10.

Proposition 11. *Consider some proper graph $\mathcal{G} := (V, E)$, set of nodes S, and let $(\mathcal{G}', S') := \mathsf{UpdatedGraph}(\mathcal{G}, S)$. Then, for every node*

$$u := (\mathtt{id}, (\langle P \to \vec{R} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))) \in S \setminus S'$$

we have

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, P, \vec{R}, \mathtt{cmd}, \mathrm{Acks}) = 0.$$

See Section B.4.11 for the proof of Proposition 11.

Proposition 12. *Consider some proper graph $\mathcal{G} := (V, E)$, set of nodes S', and any tuple*

$$u := (\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})))$$

corresponding to a Write *operation, such that*

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1.$$

Then, for $\mathcal{G}' := (V \cup \{\mathtt{id}\}, E \cup (\mathrm{Acks} \times \{\mathtt{id}\}))$, and letting

$$(\mathcal{G}_1, S_1'') := \mathsf{UpdatedGraph}(\mathcal{G}', S')$$
$$(\mathcal{G}_2, S_2'') := \mathsf{UpdatedGraph}(\mathcal{G}, S' \cup \{\mathtt{id}\}),$$

we have $(\mathcal{G}_1, S_1'' \cup \{\mathtt{id}\}) = (\mathcal{G}_2, S_2'')$.

See Section B.4.12 for the proof of Proposition 12.

Proposition 13. *In $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, for each $\mathtt{sid}$, $\mathrm{SessionGraphs}_{\mathrm{Global}}[\mathtt{sid}]$ is proper.*
In $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$, for each $\mathtt{sid}$ and each $P \in \mathcal{M}^H$, $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ is proper.

See Section B.4.13 for the proof of Proposition 13.

Proposition 14. *Consider an execution of $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ or $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, and let V_O be the set of nodes in the graph output by $\mathsf{InducedPartyGraph}^+$. For any non-root $u \in V_O$, all nodes in u's acknowledgment set Acks are in V_O.*

See Section B.4.14 for the proof of Proposition 14.

B.3 Hybrid Sequence

In the hybrids' descriptions (Algorithms 18, 19, 20, 21, 22, 23, 24, 25, 26 and 27) we only show the differences relative to the previous hybrid (or relative to the ideal world system **ChatSessions**$[\mathfrak{P}]$ or real world system $\mathbf{R}[\mathfrak{P}]$). We will use blue highlights to emphasize changes to variables that are global (in the sense of being shared among all parties), yellow highlights to emphasize changes to variables that are local to each party, and red highlights to emphasize lines that were erased (from the description of the previous hybrid).

Algorithm 17 Hybrids $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. Below, non-highlighted lines correspond to parts of description that are common among the two hybrids, whereas highlighted ones correspond to parts of the description that only concern one of the hybrids: if green they concern $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$, and if purple they concern $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.

Initialization
  $(\mathsf{Net} \cdot \mathbf{REP})$-Initialization
  Contents, $\text{SessionGraphs}_{\text{Global}}$, $\text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{Sent}[P]$, SessionGraphs_P, ToHandle_P, Undelivered_P, $\text{Delivered}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-Write$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$
  $\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$
Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathcal{G}^+, P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Sent}[P] \leftarrow \text{Sent}[P] \cup \{\mathtt{id}\}$
  $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
  $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.
  $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \setminus \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.
  $\text{Delivered}_P[\mathtt{sid}] \leftarrow \text{Delivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.
  $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

$(P \in \mathcal{M}^H)$-Read
  Output$(\{(\mathtt{sid}, \mathcal{G}^+) \mid \mathcal{G}^+ = \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P) \wedge \mathcal{G}^+ \neq (\emptyset, \emptyset)\})$

$(P \in \overline{\mathcal{P}^H})$-Write$(\langle S \to \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle S \to \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
  $\forall P' \in \text{Set}(\vec{V})^H : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

$(P \in \overline{\mathcal{P}^H})$-Read
  Output$((\mathsf{Net} \cdot \mathbf{REP})\text{-Read})$

Deliver$(P, \mathtt{id})$
  $(\mathsf{Net} \cdot \mathbf{REP})$-Deliver$(P, \mathtt{id})$
  if $\exists \mathtt{sid}$ such that $\mathtt{id} \in \text{Undelivered}_P[\mathtt{sid}] \wedge P \in \mathcal{M}^H$ **:**
    $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \setminus \{\mathtt{id}\}$
    $\text{Delivered}_P[\mathtt{sid}] \leftarrow \text{Delivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
    $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$

$\mathsf{ProcessReceived}(P)$ // Not part of interface.
  for $\mathtt{sid} \in \text{ToHandle}_P$ **:**
    $(\mathcal{G}_{\text{upd}}, \text{Handled}) \leftarrow \mathsf{UpdatedGraph}(\text{SessionGraphs}_P[\mathtt{sid}], \text{ToHandle}_P[\mathtt{sid}])$
    $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \setminus \text{Handled}$
    $\text{SessionGraphs}_P[\mathtt{sid}] \leftarrow \mathcal{G}_{\text{upd}}$

$\mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$ // Not part of interface.
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \text{SessionGraphs}_P[\mathtt{sid}]$ // Hybrid $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$.
  $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}_{\text{Global}}[\mathtt{sid}]$ // Hybrid $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.
  $V_P \leftarrow V \cap \{\mathtt{id} \mid \mathtt{id} \in \text{Delivered}_P[\mathtt{sid}] \cup \text{Sent}[P]\}$ // Hybrid $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.
  $V_0 \leftarrow V_P \cap \{\mathtt{id} \mid \text{Contents}[\mathtt{id}] = (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \cdot)) \wedge \mathfrak{P}[\text{IsRoot}](\mathtt{sid}, S, \vec{V}, \mathtt{cmd})\}$
  $i \leftarrow 0$
  repeat
    $V_{i+1} \leftarrow V_i$
    for $\mathtt{id} \in V_P$ **:**
      $(\cdot, (\cdot, \cdot, \text{Acks})) \leftarrow \text{Contents}[\mathtt{id}]$
      if $\text{Acks} \subseteq V_i$ **:**
        $V_{i+1} \leftarrow V_{i+1} \cup \{\mathtt{id}\}$
    $i \leftarrow i + 1$
  until $V_i = V_{i-1}$
  $V_{E_P} := \{\mathtt{id} \mid (\mathtt{id}, \mathtt{id}') \in E_P\}$
  return $\mathsf{Extended}(\mathcal{G}_i := (V_i, E_P \cap (V_{E_P} \times V_i)))$

$\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$ // Not part of interface.
  $\text{ToHandle}_{\text{Global}}[\mathtt{sid}] \leftarrow \text{ToHandle}_{\text{Global}}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $(\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}], \text{Handled}) \leftarrow$
      $\mathsf{UpdatedGraph}(\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}], \text{ToHandle}_{\text{Global}}[\mathtt{sid}])$
  $\text{ToHandle}_{\text{Global}}[\mathtt{sid}] \leftarrow \text{ToHandle}_{\text{Global}}[\mathtt{sid}] \setminus \text{Handled}$

Algorithm 18 Hybrid $\mathbf{H}_1^{\mathbf{RW}}$. In the description below we only show the differences relative to the real world $\mathbf{R}[\mathfrak{P}]$.

Initialization
  (Net · **REP**)-Initialization
  $\text{Contents} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{SessionGraphs}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-Write($\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks}$)
  ProcessReceived(P)
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \text{SessionGraphs}_P[\texttt{sid}]$
Require: $\mathfrak{P}[\text{IsValid}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \texttt{cmd}, \text{Acks})$
  $\texttt{id} \leftarrow$ (Net · **REP**)-Write($\langle P \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks})$)
  $\text{Contents}[\texttt{id}] \leftarrow (\langle P \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{SessionGraphs}_P[\texttt{sid}] \leftarrow (V_P \cup \{\texttt{id}\}, E_P \cup (\text{Acks} \times \{\texttt{id}\}))$
  Output($\texttt{id}$)

$(P \in \mathcal{M}^H)$-Read
  ProcessReceived(P)
  Output($\{(\texttt{sid}, \mathsf{Extended}(\mathcal{G})) \mid (\texttt{sid}, \mathcal{G}) \in \text{SessionGraphs}_P \wedge \mathcal{G} \neq (\emptyset, \emptyset)\}$)

$(P \in \overline{\mathcal{P}^H})$-Write($\langle S \to \vec{V}\rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks})$)
  $\texttt{id} \leftarrow$ (Net · **REP**)-Write($\langle S \to \vec{V}\rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks})$)
  $\text{Contents}[\texttt{id}] \leftarrow (\langle S \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  Output($\texttt{id}$)

ProcessReceived(P) // Not part of interface.
  $\text{ToHandle} \leftarrow \emptyset$
  for $(\texttt{id}, (\langle S \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))) \in$ (Net · **REP**)-Read **with** $\texttt{id} \notin \text{SessionGraphs}_P[\texttt{sid}].V$ **:**
    $\text{ToHandle}[\texttt{sid}] \leftarrow \text{ToHandle}[\texttt{sid}] \cup \{\texttt{id}\}$
  for $\texttt{sid} \in \text{ToHandle}$ **:**
    $(\mathcal{G}_{\text{upd}}, \cdot) \leftarrow \mathsf{UpdatedGraph}(\text{SessionGraphs}_P[\texttt{sid}], \text{ToHandle}[\texttt{sid}])$
    $\text{SessionGraphs}_P[\texttt{sid}] \leftarrow \mathcal{G}_{\text{upd}}$

Algorithm 19 Hybrid $\mathbf{H_2^{RW}}$. We only show the differences relative to $\mathbf{H_1^{RW}}$.

Initialization
  ($\mathsf{Net} \cdot \mathbf{REP}$)-Initialization
  $\text{Contents} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{SessionGraphs}_P,\ \ \text{ToHandle}_P,\ \ \text{Undelivered}_P,\ \ \text{Delivered}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-Write$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$
  $\mathsf{ProcessReceived}(P)$
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \text{SessionGraphs}_P[\mathtt{sid}]$
Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{SessionGraphs}_P[\mathtt{sid}] \leftarrow (V_P \cup \{\mathtt{id}\}, E_P \cup (\text{Acks} \times \{\mathtt{id}\}))$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

$(P \in \mathcal{M}^H)$-Read
  $\mathsf{ProcessReceived}(P)$
  Output$(\{(\mathtt{sid}, \mathsf{Extended}(\mathcal{G})) \mid (\mathtt{sid}, \mathcal{G}) \in \text{SessionGraphs}_P \wedge \mathcal{G} \neq (\emptyset, \emptyset)\})$

$(P \in \overline{\mathcal{P}^H})$-Write$(\langle S \to \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle S \to \vec{V} \rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\forall P' \in \text{Set}(\vec{V})^H : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

Deliver$(P, \mathtt{id})$
  ($\mathsf{Net} \cdot \mathbf{REP}$)-Deliver$(P, \mathtt{id})$
  if $\exists \mathtt{sid}$ such that $\mathtt{id} \in \text{Undelivered}_P[\mathtt{sid}] \wedge P \in \mathcal{M}^H$ **:**
    $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \setminus \{\mathtt{id}\}$
    $\text{Delivered}_P[\mathtt{sid}] \leftarrow \text{Delivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
    $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$

$\mathsf{ProcessReceived}(P)$ // Not part of interface.
  $\text{ToHandle} \leftarrow \emptyset$
  for $(\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))) \in (\mathsf{Net} \cdot \mathbf{REP})\text{-Read}$ **with** $\mathtt{id} \notin \text{SessionGraphs}_P[\mathtt{sid}].V$ **:**
    $\text{ToHandle}[\mathtt{sid}] \leftarrow \text{ToHandle}[\mathtt{sid}] \cup \{\mathtt{id}\}$ // Unused.
  for $\mathtt{sid} \in \text{ToHandle}_P$ **:**
    $(\mathcal{G}_{\text{upd}}, \text{Handled}) \leftarrow \mathsf{UpdatedGraph}(\text{SessionGraphs}_P[\mathtt{sid}], \text{ToHandle}_P[\mathtt{sid}])$
    $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \setminus \text{Handled}$
    $\text{SessionGraphs}_P[\mathtt{sid}] \leftarrow \mathcal{G}_{\text{upd}}$

Algorithm 20 Hybrid $\mathbf{H}_3^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_2^{\mathbf{RW}}$.

$(P \in \mathcal{M}^H)$-Write$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks})$
  [redacted]
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \mathrm{SessionGraphs}_P[\mathtt{sid}]$
Require: $\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \mathtt{cmd}, \mathrm{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-}\textsc{Write}(\langle P \rightarrow \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
  $\mathrm{Contents}[\mathtt{id}] \leftarrow (\langle P \rightarrow \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
  $\mathrm{SessionGraphs}_P[\mathtt{sid}] \leftarrow (V_P \cup \{\mathtt{id}\}, E_P \cup (\mathrm{Acks} \times \{\mathtt{id}\}))$
  $\mathsf{ProcessReceived}(P)$
  $\forall P' \in (\mathrm{Set}(\vec{V})^H \setminus \{P\}) : \mathrm{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \mathrm{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

$(P \in \mathcal{M}^H)$-Read
  [redacted]
  Output$(\{(\mathtt{sid}, \mathsf{Extended}(\mathcal{G})) \mid (\mathtt{sid}, \mathcal{G}) \in \mathrm{SessionGraphs}_P \wedge \mathcal{G} \neq (\emptyset, \emptyset)\})$

Deliver$(P, \mathtt{id})$
  $(\mathsf{Net} \cdot \mathbf{REP})\text{-}\textsc{Deliver}(P, \mathtt{id})$
  if $\exists \mathtt{sid}$ such that $\mathtt{id} \in \mathrm{Undelivered}_P[\mathtt{sid}] \wedge P \in \mathcal{M}^H$ **:**
    $\mathrm{Undelivered}_P[\mathtt{sid}] \leftarrow \mathrm{Undelivered}_P[\mathtt{sid}] \setminus \{\mathtt{id}\}$
    $\mathrm{Delivered}_P[\mathtt{sid}] \leftarrow \mathrm{Delivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
    $\mathrm{ToHandle}_P[\mathtt{sid}] \leftarrow \mathrm{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$

$\mathsf{ProcessReceived}(P)$ // Not part of interface.
  [redacted]
  for [redacted] **with** [redacted] **:**
    [redacted] // Unused.
  for $\mathtt{sid} \in \mathrm{ToHandle}_P$ **:**
    $(\mathcal{G}_{\mathrm{upd}}, \mathrm{Handled}) \leftarrow \mathsf{UpdatedGraph}(\mathrm{SessionGraphs}_P[\mathtt{sid}], \mathrm{ToHandle}_P[\mathtt{sid}])$
    $\mathrm{ToHandle}_P[\mathtt{sid}] \leftarrow \mathrm{ToHandle}_P[\mathtt{sid}] \setminus \mathrm{Handled}$
    $\mathrm{SessionGraphs}_P[\mathtt{sid}] \leftarrow \mathcal{G}_{\mathrm{upd}}$

Algorithm 21 Hybrid $\mathbf{H}_4^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_3^{\mathbf{RW}}$.

$(P \in \mathcal{M}^H)$-Write$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks})$
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \mathrm{SessionGraphs}_P[\mathtt{sid}]$
Require: $\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \mathtt{cmd}, \mathrm{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-}\textsc{Write}(\langle P \rightarrow \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
  $\mathrm{Contents}[\mathtt{id}] \leftarrow (\langle P \rightarrow \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
  [redacted]
  $\mathrm{ToHandle}_P[\mathtt{sid}] \leftarrow \mathrm{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$
  $\forall P' \in (\mathrm{Set}(\vec{V})^H \setminus \{P\}) : \mathrm{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \mathrm{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  Output$(\mathtt{id})$

Algorithm 22 Hybrid $\mathbf{H}_5^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_4^{\mathbf{RW}}$.

$(P \in \mathcal{M}^H)$-WRITE($\mathtt{sid}, \mathtt{cmd}, \vec{V}$, Acks)
  $\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$
Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathcal{G}^+, P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})$-WRITE($\langle P \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  OUTPUT($\mathtt{id}$)

$(P \in \mathcal{M}^H)$-READ
  OUTPUT($\{(\mathtt{sid}, \mathcal{G}^+) \mid \mathcal{G}^+ = \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P) \wedge \mathcal{G}^+ \neq (\emptyset, \emptyset)\}$)

$\mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$ // Not part of interface.
  $\mathcal{G}_P \coloneqq (V_P, E_P) \leftarrow \text{SessionGraphs}_P[\mathtt{sid}]$
  $V_0 \leftarrow V_P \cap \{\mathtt{id} \mid \text{Contents}[\mathtt{id}] = (\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \cdot)) \wedge \mathfrak{P}[\text{IsRoot}](\mathtt{sid}, S, \vec{V}, \mathtt{cmd})\}$
  $i \leftarrow 0$
  repeat
    $V_{i+1} \leftarrow V_i$
    for $\mathtt{id} \in V_P$ **:**
      $(\cdot, (\cdot, \cdot, \text{Acks})) \leftarrow \text{Contents}[\mathtt{id}]$
      if $\text{Acks} \subseteq V_i$ **:**
        $V_{i+1} \leftarrow V_{i+1} \cup \{\mathtt{id}\}$
    $i \leftarrow i + 1$
  until $V_i = V_{i-1}$
  $V_{E_P} \coloneqq \{\mathtt{id} \mid (\mathtt{id}, \mathtt{id}') \in E_P\}$
  return $\mathsf{Extended}(\mathcal{G}_i \coloneqq (V_i, E_P \cap (V_{E_P} \times V_i)))$

Algorithm 23 Hybrid $\mathbf{H}_6^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_5^{\mathbf{RW}}$.

INITIALIZATION
  $(\mathsf{Net} \cdot \mathbf{REP})$-INITIALIZATION
  Contents, $\text{SessionGraphs}_{\text{Global}}$, $\text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{Sent}[P]$, SessionGraphs_P, ToHandle_P, Undelivered_P, $\text{Delivered}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-WRITE($\mathtt{sid}, \mathtt{cmd}, \vec{V}$, Acks)
  $\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$
Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathcal{G}^+, P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})$-WRITE($\langle P \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\text{Sent}[P] \leftarrow \text{Sent}[P] \cup \{\mathtt{id}\}$
  $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
  $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}_{\text{Mid}}^{\text{RW}} \equiv \mathbf{H}_{\text{Mid}}^{\text{IW}}$.
  $\text{Undelivered}_P[\mathtt{sid}] \leftarrow \text{Undelivered}_P[\mathtt{sid}] \setminus \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}_{\text{Mid}}^{\text{RW}} \equiv \mathbf{H}_{\text{Mid}}^{\text{IW}}$.
  $\text{Delivered}_P[\mathtt{sid}] \leftarrow \text{Delivered}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$ // Helps in simplifying proof $\mathbf{H}_{\text{Mid}}^{\text{RW}} \equiv \mathbf{H}_{\text{Mid}}^{\text{IW}}$.
  $\text{ToHandle}_P[\mathtt{sid}] \leftarrow \text{ToHandle}_P[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $\mathsf{ProcessReceived}(P)$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  OUTPUT($\mathtt{id}$)

$(P \in \overline{\mathcal{P}^H})$-WRITE($\langle S \to \vec{V}\rangle, m \coloneqq (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
  $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})$-WRITE($\langle S \to \vec{V}\rangle, m \coloneqq (\mathtt{sid}, \mathtt{cmd}, \text{Acks})$)
  $\text{Contents}[\mathtt{id}] \leftarrow (\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
  $\forall P' \in \text{Set}(\vec{V})^H : \text{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \text{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  OUTPUT($\mathtt{id}$)

Hybrid Sequence: $\mathbf{R}[\mathfrak{P}] \rightsquigarrow \ldots \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$

$\mathbf{R}[\mathfrak{P}] \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{1}}$: The real world system $\mathbf{R}[\mathfrak{P}] := \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP})$—defined in Equation 4.2—and $\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}}$—defined in Algorithm 18—are different descriptions of the same system: the only difference is that now there is a unique variable Contents instead of one per converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$; since by the definition of $\mathbf{REP}$ (Algorithms 1 and 2) each $\mathtt{id}$ maps to a unique pair $(\mathbf{rep_i}, m)$, it then follows $\mathbf{R}[\mathfrak{P}] \equiv \mathbf{H}^{\mathbf{RW}}_{\mathbf{1}}$.

$\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$: The only differences between $\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}}$ and $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ (Algorithm 19) are:

- for each party $P \in \mathcal{M}^H$, $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ has additional variables Undelivered_P, Delivered_P and ToHandle_P; and
- in $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$, $\mathsf{ProcessReceived}$ uses set ToHandle_P instead of issuing a Read operation to $(\mathsf{Net} \cdot \mathbf{REP})$ and then excluding nodes already added to the (corresponding) graph.

To prove $\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}} \equiv \mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ it suffices to show that for each $\mathtt{sid}$, in hybrid $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ it holds that $\text{ToHandle}[\mathtt{sid}] = \text{ToHandle}_P[\mathtt{sid}]$; we now establish this.

Fix some $\mathtt{sid}$.

- Let $\text{Read}_P[\mathtt{sid}]$ be the set of ids output by a Read operation at P's interface of $(\mathsf{Net} \cdot \mathbf{REP})$, filtered by the fixed $\mathtt{sid}$. This means $\text{ToHandle}[\mathtt{sid}] = \text{Read}_P[\mathtt{sid}] \setminus \text{SessionGraphs}_P[\mathtt{sid}].V$.
- For any $\mathtt{id}$ and party $P \in \mathcal{M}^H$: $\mathtt{id} \in \text{Read}_P[\mathtt{sid}]$ *if and only if* there is a query $\textsc{Deliver}(P, \mathtt{id})$.
- By the semantics of $(\mathsf{Net} \cdot \mathbf{REP})$ (Algorithms 2 and 3), for any $\mathtt{id}$ (corresponding to a Write operation for the fixed $\mathtt{sid}$), $\mathtt{id}$ was added to variable set $\text{ToHandle}_P[\mathtt{sid}]$ *if and only if* there is a query $\textsc{Deliver}(P, \mathtt{id})$.
- For any $\mathtt{id}$, $\mathtt{id}$ was removed from variable set $\text{ToHandle}_P[\mathtt{sid}]$ *if and only if* there is a query $\mathsf{UpdatedGraph}(\text{SessionGraphs}_P[\mathtt{sid}], \text{ToHandle}_P[\mathtt{sid}])$ where $\mathtt{id} \in \text{ToHandle}_P[\mathtt{sid}]$ that output a pair $(\mathcal{G}_{\text{upd}}, \text{Handled})$ such that $\mathtt{id} \in \text{Handled}$. For that query, by the definition of $\mathsf{UpdatedGraph}$, $\mathtt{id} \in \mathcal{G}_{\text{upd}}.V$. And by definition of $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$, $\mathtt{id} \in \mathcal{G}_{\text{upd}}.V$ implies $\mathtt{id} \in \text{SessionGraphs}_P[\mathtt{sid}].V$.

This implies the two sets are the same, so $\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}} \equiv \mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$.

$\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{3}}$: The only difference between $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ and $\mathbf{H}^{\mathbf{RW}}_{\mathbf{3}}$ (Algorithm 20) is that, for each party $P \in \mathcal{M}^H$: in the latter $\mathsf{ProcessReceived}(P)$ is called 1. upon each $\textsc{Delivery}(P, \cdot)$ query, and 2. on each Write query at P's interface, after adding the resulting node to $\text{SessionGraphs}_P[\mathtt{sid}]$; in the former it is called upon each Read or Write query at P's interface, at the beginning of the query. (Regarding the differences for $\mathsf{ProcessReceived}$, it is easy to see that these are only syntactical, not semantical, as variable ToHandle is not used.)

Consider the sequence of hybrids $\mathbf{R}[\mathfrak{P}] \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{1}} \rightsquigarrow \mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$: for each $P \in \mathcal{M}^H$ and each $\mathtt{sid}$, $\text{SessionGraphs}_P[\mathtt{sid}]$ in $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ is handled (i.e. read/written) exactly the same way as in $\mathbf{R}[\mathfrak{P}]$, and so it is proper *if and only if* in $\mathbf{R}[\mathfrak{P}]$ the graph SessionGraphs[$\texttt{sid}$] stored in P's converter is proper. By Proposition 6, in $\mathbf{R}[\mathfrak{P}]$, for each party $P \in \mathcal{M}^H$ and each $\texttt{sid}$, the graph SessionGraphs[$\texttt{sid}$] stored in P's converter is proper. Therefore, for each party $P \in \mathcal{M}^H$ and for each $\texttt{sid}$, $\text{SessionGraphs}_P[\texttt{sid}]$ in $\mathbf{H}_\mathbf{2}^{\mathbf{RW}}$ is proper. With this established, we can now use Proposition 10 to proceed via induction.

In the following, consider some party $P \in \mathcal{M}^H$ and some $\texttt{sid}$. To begin note that $\mathbf{H}_\mathbf{2}^{\mathbf{RW}}$ and $\mathbf{H}_\mathbf{3}^{\mathbf{RW}}$ may only differ upon either a Write query—with a matching input $\texttt{sid}$—or a Read query. To prove they do not differ, it suffices to show that for both Write and Read queries, graphs $\text{SessionGraphs}_P[\texttt{sid}]$ in $\mathbf{H}_\mathbf{2}^{\mathbf{RW}}$ after $\mathsf{ProcessReceived}(P)$ is called and at the beginning of the query in $\mathbf{H}_\mathbf{3}^{\mathbf{RW}}$ are exactly the same. In both $\mathbf{H}_\mathbf{2}^{\mathbf{RW}}$ and $\mathbf{H}_\mathbf{3}^{\mathbf{RW}}$, upon Initialization the following holds:

- $\texttt{sid} \notin \text{SessionGraphs}_P$, which implies $\text{SessionGraphs}_P[\texttt{sid}] = (\mathcal{G}_\emptyset \coloneqq (\emptyset, \emptyset))$;
- $\texttt{sid} \notin \text{ToHandle}_P$, and so $\text{ToHandle}_P[\texttt{sid}] = \emptyset$;[21]

We now proceed via induction on the state of both $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$ since the last query to either Write or Read; if there was no prior query to either interface, consider instead the state of $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$ right after Initialization, i.e. $\text{SessionGraphs}_P[\texttt{sid}] = \mathcal{G}_\emptyset$ and $\text{ToHandle}_P[\texttt{sid}] = \emptyset$, as described above. Let $q_1, \ldots, q_n$ denote, in order, the Deliver queries with input $(P, \texttt{id}_i)$ since the last query to either the Write or Read interfaces of P, or since the end of Initialization (if there was no prior Write or Read query). For $i = 1, \ldots, n$, let $\texttt{id}_i$ be the identifier input to query q_i, and define set D_i as

$$D_i \coloneqq \begin{cases} \{\texttt{id}_i\}, & \text{if } \texttt{id}_i \in \text{Undelivered}_P[\texttt{sid}] \text{ at the start of } q_i \\ \emptyset, & \text{otherwise.} \end{cases}$$

Letting

$$S \coloneqq S' \cup \left(\bigcup_{i=1,\ldots,n} D_i \right),$$

where S' is defined as the set $\text{ToHandle}_P[\texttt{sid}]$ at the end of the last Write or Read query, or as $\emptyset$ if there was no such prior query, and letting $\mathcal{G}'$ be the state of $\text{SessionGraphs}_P[\texttt{sid}]$ also at the end of such last query (or $\mathcal{G}_\emptyset$ if there was none), note that in $\mathbf{H}_\mathbf{2}^{\mathbf{RW}}$, $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$ are updated using $\mathsf{UpdatedGraph}$ with input graph $\mathcal{G}'$ and input set S, i.e. letting

$$(\mathcal{G}_{\text{new}}, \text{Handled}) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}', S),$$
$$\text{ToHandle}_{\text{new}} \coloneqq S \setminus \text{Handled},$$

in the new query to P's Read or Write interface, $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$ are set to, respectively, $\mathcal{G}_{\text{new}}$ and $\text{ToHandle}_{\text{new}}$ after $\mathsf{ProcessReceived}$

[21] This is by convention that if $\texttt{sid}$ is not currently mapped to a set, then it is the same as mapping to the empty set.

is called on the READ or WRITE query. But this means that we can now rely on Proposition 10 to conclude the proof; concretely:

– if the last query to P's interface was a WRITE query, say q_{WRITE}, then let $n' := n + 1$, let S_1 be the set of nodes in ToHandle$_P$[$\texttt{sid}$] right after $\mathsf{ProcessReceived}(P)$ is called in the beginning of query q_{WRITE}—i.e. $S_1 :=$ ToHandle$_P$[$\texttt{sid}$]—and for $i = 2, \ldots, n'$, let $S_i := D_{i-1}$;
– if the last query to P's interface was a READ query, say q_{READ}, then let $n' := n$, let S_1 be the set of nodes in ToHandle$_P$[$\texttt{sid}$] right after the call to $\mathsf{ProcessReceived}(P)$ in the beginning of query q_{READ} together with D_1—i.e. $S_1 :=$ ToHandle$_P$[$\texttt{sid}$] $\cup D_1$—and for $i = 2, \ldots, n'$, let $S_i := D_i$;
– if there was no prior query to P's READ or WRITE interfaces, then let $n' := n$, and for $i = 1, \ldots, n'$, let $S_i := D_i$.

Note that in all cases

$$S = \bigcup_{i=1,\ldots,n'} S_i$$

and so by Proposition 10 it then follows $\mathbf{H_2^{RW}} \equiv \mathbf{H_3^{RW}}$.

$\mathbf{H_3^{RW}} \rightsquigarrow \mathbf{H_4^{RW}}$**:** The only difference between $\mathbf{H_3^{RW}}$ and $\mathbf{H_4^{RW}}$ is that in $\mathbf{H_4^{RW}}$ (Algorithm 21), upon a query $(P \in \mathcal{M}^H)$-WRITE$(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$, instead of adding the resulting node directly to graph SessionGraphs$_P$[$\texttt{sid}$], the node is instead added to set ToHandle$_P$[$\texttt{sid}$]. However, it follows from Proposition 12 that in the two cases both SessionGraphs$_P$[$\texttt{sid}$] and ToHandle$_P$[$\texttt{sid}$] are still the same at the end of the $(P \in \mathcal{M}^H)$-WRITE query. Therefore, $\mathbf{H_3^{RW}} \equiv \mathbf{H_4^{RW}}$.

$\mathbf{H_4^{RW}} \rightsquigarrow \mathbf{H_5^{RW}}$**:** The only difference between $\mathbf{H_4^{RW}}$ and $\mathbf{H_5^{RW}}$ (Algorithm 22) is that for a party P and some $\texttt{sid}$, $\mathbf{H_5^{RW}}$ now computes $\mathsf{InducedPartyGraph}^+$ on SessionGraphs$_P$[$\texttt{sid}$] instead of simply using this graph. To prove $\mathbf{H_4^{RW}} \equiv \mathbf{H_5^{RW}}$ it suffices to show that when, in $\mathsf{InducedPartyGraph}^+$, graph $\mathcal{G}_P := (V_P, E_P)$ is set to SessionGraphs$_P$[$\texttt{sid}$], the output of function $\mathsf{InducedPartyGraph}^+(\texttt{sid}, P)$ is $\mathsf{Extended}$(SessionGraphs$_P$[$\texttt{sid}$]). To begin, note that from Proposition 7 each graph SessionGraphs$_P$[$\texttt{sid}$] in $\mathbf{H_4^{RW}}$ is proper. Furthermore, it is easy to see that the set of edges E of the graph $\mathcal{G} := (V, E)$ output by $\mathsf{InducedPartyGraph}^+$ is such that, for function f defined in Proposition 2—i.e. $f(u) := \text{Acks} \times \{\texttt{id}\}$—we have

$$E = \bigcup_{u \in V} f(u).$$

Therefore we only need to show that the set of vertices V of the graph output by $\mathsf{InducedPartyGraph}^+$ is the set of vertices of SessionGraphs$_P$[$\texttt{sid}$]. Below we prove V includes all nodes in SessionGraphs$_P$[$\texttt{sid}$] (the other direction follows trivially from inspection of $\mathsf{InducedPartyGraph}^+$).

Letting SessionGraphs$_P$[$\texttt{sid}$] $:= \mathcal{G}_P := (V_P, E_P)$, by Definition 2, for $n = |V_P|$, there is an ordered sequence of nodes $u_1, \ldots, u_n$ such that, letting

$$\mathcal{G}_0 := (V_0, E_0) = (\emptyset, \emptyset),$$

and letting for $i = 0, \ldots, n-1$,

$$\mathcal{G}_{i+1} \coloneqq (V_i \cup \{u_{i+1}.\mathtt{id}\}, E_i \cup (u_{i+1}.\mathrm{Acks} \times \{u_{i+1}.\mathtt{id}\})),$$

it holds that

$$\textsc{IsValid}(u_{i+1}.\mathtt{sid}, \mathcal{G}_i^+, u_{i+1}.S, u_{i+1}.\vec{V}, u_{i+1}.\mathtt{cmd}, u_{i+1}.\mathrm{Acks}) = 1,$$

and for $i = 0, \ldots, n$, graph $\mathcal{G}_i$ is proper. By definition of $\mathsf{InducedPartyGraph}^+$ all root nodes are in V_0 and thus are part of the output graph, so we only need to prove that all non-root nodes are also added. We proceed by contradiction: consider the first node u_j in the sequence $u_1, \ldots, u_n$ that is not added to the output graph. To begin, we have

$$\textsc{IsValid}(u_j.\mathtt{sid}, \mathcal{G}_{j-1}^+, u_j.S, u_j.\vec{V}, u_j.\mathtt{cmd}, u_j.\mathrm{Acks}) = 1.$$

Since u_j is not a root node, it follows from Requirement 2 that for each $\mathtt{id} \in u_j.\mathrm{Acks}$ there is a node $(\mathtt{id}, \cdot) \in \mathcal{G}_{j-1}^+.V^+$. By Requirement 3, for a proper graph $\mathcal{G} = (V, E)$ such that $\mathcal{G}_{j-1} = (V_{j-1}, E_{j-1})$ is a subgraph of $\mathcal{G}$, since $u_j.\mathrm{Acks} \subseteq V_{j-1}$,

$$\begin{aligned}&\textsc{IsValid}(u_j.\mathtt{sid}, \mathcal{G}^+, u_j.S, u_j.\vec{V}, u_j.\mathtt{cmd}, u_j.\mathrm{Acks})\\ &= \textsc{IsValid}(u_j.\mathtt{sid}, \mathcal{G}_{j-1}^+, u_j.S, u_j.\vec{V}, u_j.\mathtt{cmd}, u_j.\mathrm{Acks}),\end{aligned}$$

and so

$$\textsc{IsValid}(u_j.\mathtt{sid}, \mathcal{G}^+, u_j.S, u_j.\vec{V}, u_j.\mathtt{cmd}, u_j.\mathrm{Acks}) = 1.$$

This means it only remains to prove the graph output by $\mathsf{InducedPartyGraph}^+$ is proper to obtain a contradiction; but this follows from Proposition 4, so indeed $\mathbf{H_4^{RW}} \equiv \mathbf{H_5^{RW}}$.

$\mathbf{H_5^{RW}} \rightsquigarrow \mathbf{H_6^{RW}}$**:** The only difference between $\mathbf{H_5^{RW}}$ and $\mathbf{H_6^{RW}}$ (Algorithm 23) are the three new variables $\mathrm{SessionGraphs}_{\mathrm{Global}}$, $\mathrm{ToHandle}_{\mathrm{Global}}$ and $\mathrm{Sent}[P]$ (for each $P \in \mathcal{M}^H$) in $\mathbf{H_6^{RW}}$, and that now upon a $(P \in \mathcal{M}^H)$-$\textsc{Write}$ query the resulting $\mathtt{id}$ is added and removed from set $\mathrm{Undelivered}_P[\mathtt{sid}]$, and it is also added to set $\mathrm{Delivered}_P[\mathtt{sid}]$. First, note that the behavior of $\mathbf{H_6^{RW}}$ is independent of the two new variables and of $\mathrm{Delivered}_P[\mathtt{sid}]$, implying that adding $\mathtt{id}$ to $\mathrm{Delivered}_P[\mathtt{sid}]$ does not affect $\mathbf{H_6^{RW}}$'s behavior. Regarding adding and then removing $\mathtt{id}$ from set $\mathrm{Undelivered}_P[\mathtt{sid}]$:

- if $\mathtt{id}$ was not in set $\mathrm{Undelivered}_P[\mathtt{sid}]$ prior to the query, then adding and removing it from the set has no side-effects;
- if $\mathtt{id}$ was already in $\mathrm{Undelivered}_P[\mathtt{sid}]$ (prior to the query) then it is removed from the set. However, in this case the only difference is that, because $\mathtt{id}$ is removed, upon a query $\textsc{Deliver}(P, \mathtt{id})$, it is not added to sets $\mathrm{Delivered}_P[\mathtt{sid}]$ and $\mathrm{ToHandle}_P[\mathtt{sid}]$. However, on one hand, as we already explained $\mathbf{H_6^{RW}}$ is independent of $\mathrm{Delivered}_P[\mathtt{sid}]$, and on the other hand, $\mathtt{id}$ is added to $\mathrm{ToHandle}_P[\mathtt{sid}]$ and there is a call to $\mathsf{ProcessReceived}(P)$, so from Proposition 9 even in this case there is no difference in behavior of hybrid $\mathbf{H_6^{RW}}$.

It then follows $\mathbf{H_5^{RW}} \equiv \mathbf{H_6^{RW}}$.

$\mathbf{H}_6^{\mathbf{RW}} \rightsquigarrow \mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$**:** Note that $\mathbf{H}_6^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ (Algorithm 17) have the same description, so $\mathbf{H}_5^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$.

Algorithm 24 Hybrid $\mathbf{H}_1^{\mathbf{IW}}$ for the proof of Theorem 1. In the description below we only show what is different relative to **ChatSessions**[$\mathfrak{P}$].

INITIALIZATION
  ($\mathsf{Net} \cdot \mathbf{REP}$)-INITIALIZATION
  Contents, $\text{SessionGraphs}_{\text{Global}}$, $\text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{Sent}[P] \leftarrow \emptyset$

$\mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$ // Not part of interface.
  $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}_{\text{Global}}[\mathtt{sid}]$
  $V_P \leftarrow V \cap \{\mathtt{id} \mid (\mathtt{id}, (\cdot, (\mathtt{sid}, \cdot, \cdot))) \in (\mathsf{Net} \cdot \mathbf{REP})\text{-READ} \cup \text{Sent}[P]\}$
  $V_0 \leftarrow V_P \cap \{\mathtt{id} \mid \text{Contents}[\mathtt{id}] = (\langle S \rightarrow \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \cdot)) \wedge \mathfrak{P}[\text{ISROOT}](\mathtt{sid}, S, \vec{V}, \mathtt{cmd})\}$
  $i \leftarrow 0$
  repeat
    $V_{i+1} \leftarrow V_i$
    for $\mathtt{id} \in V_P$ **:**
      $(\cdot, (\cdot, \cdot, \text{Acks})) \leftarrow \text{Contents}[\mathtt{id}]$
      if $\text{Acks} \subseteq V_i$ **:**
        $V_{i+1} \leftarrow V_{i+1} \cup \{\mathtt{id}\}$
    $i \leftarrow i + 1$
  until $V_i = V_{i-1}$
  $V_E := \{\mathtt{id} \mid (\mathtt{id}, \mathtt{id}') \in E\}$
  return $\mathsf{Extended}(\mathcal{G}_i := (V_i, E \cap (V_E \times V_i)))$

$\diamond$ $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$ // Not part of interface.
  $\text{ToHandle}_{\text{Global}}[\mathtt{sid}] \leftarrow \text{ToHandle}_{\text{Global}}[\mathtt{sid}] \cup \{\mathtt{id}\}$
  $(\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}], \text{Handled}) \leftarrow$
      $\mathsf{UpdatedGraph}(\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}], \text{ToHandle}_{\text{Global}}[\mathtt{sid}])$
  $\text{ToHandle}_{\text{Global}}[\mathtt{sid}] \leftarrow \text{ToHandle}_{\text{Global}}[\mathtt{sid}] \setminus \text{Handled}$

Hybrid Sequence: $\mathbf{R}[\mathfrak{P}] \rightsquigarrow \ldots \rightsquigarrow \mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$

ChatSessions[$\mathfrak{P}$] $\rightsquigarrow \mathbf{H}_1^{\mathbf{IW}}$**: ChatSessions**[$\mathfrak{P}$] (Algorithm 6) and $\mathbf{H}_1^{\mathbf{IW}}$ (defined in Algorithm 24) only differ in the names of variables SessionGraphs and ToHandle, and so **ChatSessions**[$\mathfrak{P}$] $\equiv \mathbf{H}_1^{\mathbf{IW}}$.

$\mathbf{H}_1^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_2^{\mathbf{IW}}$**:** The only difference between $\mathbf{H}_1^{\mathbf{IW}}$ and $\mathbf{H}_2^{\mathbf{IW}}$ (Algorithm 25) is that in $\mathbf{H}_2^{\mathbf{IW}}$ there are, for each party $P \in \mathcal{M}^H$, additional variables ToHandle_P, Undelivered_P and Delivered_P. However, none of these variables have any effect in the behavior of $\mathbf{H}_2^{\mathbf{IW}}$, so $\mathbf{H}_1^{\mathbf{IW}} \equiv \mathbf{H}_2^{\mathbf{IW}}$.

$\mathbf{H}_2^{\mathbf{IW}} \rightsquigarrow \mathbf{H}_3^{\mathbf{IW}}$**:** Hybrid $\mathbf{H}_3^{\mathbf{IW}}$ (Algorithm 26) only differs from $\mathbf{H}_2^{\mathbf{IW}}$ in what the variable V_P in the $\mathsf{InducedPartyGraph}^+$ procedure is set to: for a party P, in $\mathbf{H}_2^{\mathbf{IW}}$, V_P is set to the union of the ids of the nodes output by P's READ operation from ($\mathsf{Net}\cdot\mathbf{REP}$) and $\text{Sent}[P]$, whereas in $\mathbf{H}_3^{\mathbf{IW}}$ it is set to the union of $\text{Delivered}_P[\mathtt{sid}]$ and $\text{Sent}[P]$. However, from inspection of $\mathbf{H}_3^{\mathbf{IW}}$ and by the definition of ($\mathsf{Net} \cdot \mathbf{REP}$) (Algorithms 2 and 3), for any party $P \in \mathcal{M}^H$ and any $\mathtt{id}$, we have

Algorithm 25 Hybrid $\mathbf{H}_2^{\mathbf{IW}}$. We only show the differences relative to $\mathbf{H}_1^{\mathbf{IW}}$.

INITIALIZATION
  $(\mathsf{Net} \cdot \mathbf{REP})$-INITIALIZATION
  $\text{Contents},\ \ \text{SessionGraphs}_{\text{Global}},\ \ \text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{Sent}[P],\ \ \text{ToHandle}_P,\ \ \text{Undelivered}_P,\ \ \text{Delivered}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-WRITE$(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  $\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\texttt{sid}, P)$
Require: $\mathfrak{P}[\text{IsValid}](\texttt{sid}, \mathcal{G}^+, P, \vec{V}, \texttt{cmd}, \text{Acks})$
  $\texttt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle P \rightarrow \vec{V} \rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Contents}[\texttt{id}] \leftarrow (\langle P \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Sent}[P] \leftarrow \text{Sent}[P] \cup \{\texttt{id}\}$
  $\mathsf{AddToGraph}(\texttt{sid}, \texttt{id})$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\texttt{sid}] \leftarrow \text{Undelivered}_{P'}[\texttt{sid}] \cup \{\texttt{id}\}$
  OUTPUT$(\texttt{id})$

$(P \in \overline{\mathcal{P}^H})$-WRITE$(\langle S \rightarrow \vec{V} \rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\texttt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-Write}(\langle S \rightarrow \vec{V} \rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Contents}[\texttt{id}] \leftarrow (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\mathsf{AddToGraph}(\texttt{sid}, \texttt{id})$
  $\forall P' \in \text{Set}(\vec{V})^H : \text{Undelivered}_{P'}[\texttt{sid}] \leftarrow \text{Undelivered}_{P'}[\texttt{sid}] \cup \{\texttt{id}\}$
  OUTPUT$(\texttt{id})$

DELIVER$(P, \texttt{id})$
  $(\mathsf{Net} \cdot \mathbf{REP})$-DELIVER$(P, \texttt{id})$
  if $\exists \texttt{sid}$ such that $\texttt{id} \in \text{Undelivered}_P[\texttt{sid}] \wedge P \in \mathcal{M}^H$ **:**
    $\text{Undelivered}_P[\texttt{sid}] \leftarrow \text{Undelivered}_P[\texttt{sid}] \setminus \{\texttt{id}\}$
    $\text{Delivered}_P[\texttt{sid}] \leftarrow \text{Delivered}_P[\texttt{sid}] \cup \{\texttt{id}\}$
    $\text{ToHandle}_P[\texttt{sid}] \leftarrow \text{ToHandle}_P[\texttt{sid}] \cup \{\texttt{id}\}$

Algorithm 26 Hybrid $\mathbf{H}_3^{\mathbf{IW}}$. We only show the differences relative to $\mathbf{H}_2^{\mathbf{IW}}$.

$\mathsf{InducedPartyGraph}^+(\texttt{sid}, P)$ // Not part of interface.
  $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$
  $V_P \leftarrow V \cap \{\texttt{id} \mid \texttt{id} \in \text{Delivered}_P[\texttt{sid}] \cup \text{Sent}[P]\}$
  $V_0 \leftarrow V_P \cap \{\texttt{id} \mid \text{Contents}[\texttt{id}] = (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \cdot)) \wedge \mathfrak{P}[\text{IsRoot}](\texttt{sid}, S, \vec{V}, \texttt{cmd})\}$
  $i \leftarrow 0$
  repeat
    $V_{i+1} \leftarrow V_i$
    for $\texttt{id} \in V_P$ **:**
      $(\cdot, (\cdot, \cdot, \text{Acks})) \leftarrow \text{Contents}[\texttt{id}]$
      if $\text{Acks} \subseteq V_i$ **:**
        $V_{i+1} \leftarrow V_{i+1} \cup \{\texttt{id}\}$
    $i \leftarrow i + 1$
  until $V_i = V_{i-1}$
  $V_E := \{\texttt{id} \mid (\texttt{id}, \texttt{id}') \in E\}$
  return $\mathsf{Extended}(\mathcal{G}_i := (V_i, E \cap (V_E \times V_i)))$

Algorithm 27 Hybrid $\mathbf{H}_4^{\mathbf{IW}}$. We only show the differences relative to $\mathbf{H}_3^{\mathbf{IW}}$.

Initialization
  (Net · **REP**)-Initialization
  Contents, $\text{SessionGraphs}_{\text{Global}}$, $\text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ **:**
    $\text{Sent}[P]$, SessionGraphs_P, ToHandle_P, Undelivered_P, $\text{Delivered}_P \leftarrow \emptyset$

$(P \in \mathcal{M}^H)$-Write(sid, cmd, $\vec{V}$, Acks)
  $\mathcal{G}^+ \leftarrow$ InducedPartyGraph$^+$(sid, P)
Require: $\mathfrak{P}$[IsValid](sid, $\mathcal{G}^+$, P, $\vec{V}$, cmd, Acks)
  id $\leftarrow$ (Net · **REP**)-Write($\langle P \to \vec{V} \rangle$, $m :=$ (sid, cmd, Acks))
  Contents[id] $\leftarrow$ ($\langle P \to \vec{V} \rangle$, (sid, cmd, Acks))
  Sent[P] $\leftarrow$ Sent[P] $\cup$ {id}
  AddToGraph(sid, id)
  Undelivered_P[sid] $\leftarrow$ Undelivered_P[sid] $\cup$ {id} // Helps in simplifying proof $\mathbf{H}^{\text{RW}}_{\text{Mid}} \equiv \mathbf{H}^{\text{IW}}_{\text{Mid}}$.
  Undelivered_P[sid] $\leftarrow$ Undelivered_P[sid] $\setminus$ {id} // Helps in simplifying proof $\mathbf{H}^{\text{RW}}_{\text{Mid}} \equiv \mathbf{H}^{\text{IW}}_{\text{Mid}}$.
  Delivered_P[sid] $\leftarrow$ Delivered_P[sid] $\cup$ {id} // Helps in simplifying proof $\mathbf{H}^{\text{RW}}_{\text{Mid}} \equiv \mathbf{H}^{\text{IW}}_{\text{Mid}}$.
  ToHandle_P[sid] $\leftarrow$ ToHandle_P[sid] $\cup$ {id}
  ProcessReceived(P)
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\})$: $\text{Undelivered}_{P'}$[sid] $\leftarrow$ $\text{Undelivered}_{P'}$[sid] $\cup$ {id}
  Output(id)

Deliver(P, id)
  (Net · **REP**)-Deliver(P, id)
  if $\exists$sid such that id $\in$ Undelivered_P[sid] $\wedge$ $P \in \mathcal{M}^H$ **:**
    Undelivered_P[sid] $\leftarrow$ Undelivered_P[sid] $\setminus$ {id}
    Delivered_P[sid] $\leftarrow$ Delivered_P[sid] $\cup$ {id}
    ToHandle_P[sid] $\leftarrow$ ToHandle_P[sid] $\cup$ {id}
  ProcessReceived(P)

ProcessReceived(P) // Not part of interface.
  for sid $\in$ ToHandle_P **:**
    ($\mathcal{G}_{\text{upd}}$, Handled) $\leftarrow$ UpdatedGraph(SessionGraphs_P[sid], ToHandle_P[sid])
    ToHandle_P[sid] $\leftarrow$ ToHandle_P[sid] $\setminus$ Handled
    SessionGraphs_P[sid] $\leftarrow$ $\mathcal{G}_{\text{upd}}$

$\mathtt{id} \in \text{Delivered}_P[\mathtt{sid}]$ *if and only if* there is a node $u \coloneqq (\mathtt{id}, (\cdot, (\mathtt{sid}, \cdot, \cdot)))$ that is output by P-$(\mathsf{Net} \cdot \mathbf{REP})$-Read. This then implies $\mathbf{H_2^{IW}} \equiv \mathbf{H_3^{IW}}$.

$\mathbf{H_3^{IW}} \rightsquigarrow \mathbf{H_4^{IW}}$**:** The only difference between $\mathbf{H_3^{IW}}$ and $\mathbf{H_4^{IW}}$ (Algorithm 27) is the additional variable SessionGraphs_P (see Algorithm 27), that upon a $(P \in \mathcal{M}^H)$-Write query the resulting id is added and removed from set $\text{Undelivered}_P[\mathtt{sid}]$, and it is added to sets $\text{Delivered}_P[\mathtt{sid}]$ and $\text{ToHandle}_P[\mathtt{sid}]$, and that in $(P \in \mathcal{M}^H)$-Write and $\text{Deliver}(P, \cdot)$ queries, $\mathsf{ProcessReceived}(P, \cdot)$ is called. First, note that $\text{ToHandle}_P[\mathtt{sid}]$ may only affect $\text{SessionGraphs}_P[\mathtt{sid}]$, and in turn $\text{SessionGraphs}_P[\mathtt{sid}]$ does not have any effect in the behavior of $\mathbf{H_4^{IW}}$; regarding the change in variable $\text{Undelivered}_P[\mathtt{sid}]$, note that adding and then removing id may only have side effects if id is already in $\text{Undelivered}_P[\mathtt{sid}]$ as this may prevent a later $\text{Deliver}(P, \mathtt{id})$ query from adding id to variable $\text{Delivered}_P[\mathtt{sid}]$—if id is not in $\text{Undelivered}_P[\mathtt{sid}]$, then adding and removing it has no side effects. However, even in case id is in $\text{Undelivered}_P[\mathtt{sid}]$, noting that id is added to $\text{Delivered}_P[\mathtt{sid}]$, it follows that there are no side-effects to the behavior of $\mathbf{H_4^{IW}}$. It then follows $\mathbf{H_3^{IW}} \equiv \mathbf{H_4^{IW}}$.

$\mathbf{H_4^{IW}} \rightsquigarrow \mathbf{H_{Mid}^{IW}}$**:** Systems $\mathbf{H_4^{IW}}$ and $\mathbf{H_{Mid}^{IW}}$ (Algorithm 17) have the same description, so $\mathbf{H_4^{IW}} \equiv \mathbf{H_{Mid}^{IW}}$.

(Final Hop) $\mathbf{H_{Mid}^{RW}} \rightsquigarrow \mathbf{H_{Mid}^{IW}}$**:** As is clear in the description of $\mathbf{H_{Mid}^{RW}}$ and $\mathbf{H_{Mid}^{IW}}$ (Algorithm 17), the only difference between these systems is the set of nodes V_P on which $\mathsf{InducedPartyGraph}^+$ computes. It suffices to show that in both cases $\mathsf{InducedPartyGraph}^+$ outputs the same graph.

To begin, note that Proposition 13 already establishes:

- in $\mathbf{H_{Mid}^{IW}}$, for each sid, $\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}]$ is proper; and
- in $\mathbf{H_{Mid}^{RW}}$, for each sid and each $P \in \mathcal{M}^H$, $\text{SessionGraphs}_P[\mathtt{sid}]$ is proper.

We need to show that, for each Write and Read query at the interface of an honest party $P \in \mathcal{M}^H$, the output of $\mathsf{InducedPartyGraph}^+$ is the same independently of whether it is computed as in $\mathbf{H_{Mid}^{RW}}$ or as in $\mathbf{H_{Mid}^{IW}}$. For both $\mathbf{H_{Mid}^{RW}}$ and $\mathbf{H_{Mid}^{IW}}$, the graph $\mathcal{G}^+ \coloneqq (V^+, E^+)$ output by $\mathsf{InducedPartyGraph}^+$ is such that

$$E^+ = \bigcup_{u \in V^+} f(u)$$

where f is the function defined in Proposition 2, i.e.

$$f(u \coloneqq (\mathtt{id}, (\cdot, (\cdot, \cdot, \text{Acks})))) \coloneqq \text{Acks} \times \{\mathtt{id}\};$$

therefore showing the set of nodes is the same in both cases is sufficient (as it implies the graph is also the same).

Fix some sid and some party $P \in \mathcal{P}^H$. In the following, let

$$\begin{aligned} V^{\text{Global}} &\coloneqq \text{SessionGraphs}_{\text{Global}}[\mathtt{sid}].V, \\ V_P^{\text{Global}} &\coloneqq V^{\text{Global}} \cap (\text{Delivered}_P[\mathtt{sid}] \cup \text{Sent}[P]), \\ V_P^{\text{Local}} &\coloneqq \text{SessionGraphs}_P[\mathtt{sid}].V. \end{aligned}$$

Before moving on with the proof, we first establish a few helpful facts regarding both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.

Helpful Facts.

Fact 1. *Any node added to* $\text{ToHandle}_P[\texttt{sid}]$ *is in* $\text{Delivered}_P[\texttt{sid}]$.

Proof (Fact 1). Follows from inspection of Deliver and $(P \in \mathcal{M}^H)$-Write in hybrids $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ (Algorithm 17). □

Fact 2. *Any node in* $\text{Delivered}_P[\texttt{sid}]$ *was added to* $\text{ToHandle}_P[\texttt{sid}]$.

Proof (Fact 2). Follows from inspection of Deliver and $(P \in \mathcal{M}^H)$-Write in hybrids $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ (Algorithm 17). □

Fact 3. *Any root that was added to* $\text{ToHandle}_P[\texttt{sid}]$ *is added to* V_P^{Local}.

Proof (Fact 3). From inspection of both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, whenever a node is added to ToHandle_P there is a subsequent call—during the same query—to $\mathsf{ProcessReceived}(P)$.

Consider any root node

$$u \coloneqq (\texttt{id}, (\langle S \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))).$$

First note that $\text{SessionGraphs}_P[\texttt{sid}]$ is proper, and that $\mathsf{UpdatedGraph}$ constructs the output graph following the definition of proper graph (Definition 2); in particular, note that each intermediate graph $\mathcal{G}_i$ is proper. It then follows from Requirement 1 that for each such intermediate graph $\mathcal{G}_i$ we have

$$\mathfrak{P}[\text{IsValid}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}_i), S, \vec{V}, \texttt{cmd}, \text{Acks}) = 1.$$

By inspection of $\mathsf{ProcessReceived}$ and in particular of $\mathsf{UpdatedGraph}$, it follows that u is added to the output graph, and therefore was added to V_P^{Local}. □

Fact 4. *Any node in* V_P^{Local} *was added to* $\text{ToHandle}_P[\texttt{sid}]$.

Proof (Fact 4). From inspection, the only place where nodes may be added to V_P^{Local} is in $\mathsf{ProcessReceived}$; in turn, in $\mathsf{ProcessReceived}$ only nodes in ToHandle_P may be added to V_P^{Local} (Proposition 8), so the statement holds. □

Fact 5. *Any node added to* $\text{ToHandle}_P[\texttt{sid}]$ *was previously in* $\text{Undelivered}_P[\texttt{sid}]$.

Proof (Fact 5). Follows from inspection of Deliver and $(P \in \mathcal{M}^H)$-Write in hybrids $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ (Algorithm 17). □

Fact 6. *Any node in* $\text{Undelivered}_P[\texttt{sid}]$ *was added to* $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$.

Proof (Fact 6). From inspection of hybrids $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ (Algorithm 17), nodes are only added to $\text{Undelivered}_P[\texttt{sid}]$ in Write operations (at both the interfaces of honest and dishonest parties). However, in both cases they are also added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$, so the statement holds. □

Fact 7. *Any node added to* $\text{ToHandle}_P[\texttt{sid}]$ *was previously added to* $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$.

Proof (Fact 7). Follows from Facts 5 and 6. □

Fact 8. *Any root that was added to* $\text{ToHandle}_{Global}[\texttt{sid}]$ *is added to* V^{Global}.

Proof (Fact 8). From inspection of both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, a node is only added to $\text{ToHandle}_{\text{Global}}$ in $\mathsf{AddToGraph}$ calls. Furthermore, in such calls there is always a subsequent query to $\mathsf{UpdatedGraph}$.

One can establish this fact by following arguments similar to the ones used in the proof of Fact 3. □

Fact 9. *Any node in* V_P^{Local} *was added to* $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$.

Proof (Fact 9). Every node in V_P^{Local} was previously added to $\text{ToHandle}_P[\texttt{sid}]$. Fact 7 then implies the result. □

Fact 10. *Any node* $u \coloneqq (\texttt{id}, (\langle S \to \vec{R}\rangle, (\texttt{sid}', \texttt{cmd}, \text{Acks})))$ *in* $\text{Sent}[P]$ *is also in* $\text{Delivered}_P[\texttt{sid}']$.

Proof (Fact 10). From inspection of $(P \in \mathcal{M}^H)$-$\textsc{Write}$ in Algorithm 17, whenever a node with a given $\texttt{sid}'$ is added to $\text{Sent}[P]$, it is subsequently added to $\text{Delivered}_P[\texttt{sid}']$, so the statement holds. □

Fact 11. *After any query to any of the interfaces of* $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ *or* $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, *every node*

$$u \coloneqq (\texttt{id}, (\langle S \to \vec{R}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))),$$

in $\text{ToHandle}_P[\texttt{sid}]$ *but not in* V_P^{Local}*—i.e.* $u \in (\text{ToHandle}_P[\texttt{sid}] \setminus V_P^{\text{Local}})$*—is such that*

$$\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathcal{G}^+_{Local}, S, \vec{R}, \texttt{cmd}, \text{Acks}) = 0,$$

where $\mathcal{G}^+_{Local}$ *is the extended graph corresponding to set of nodes* V_P^{Local} *(see Proposition 2).*

Proof (Fact 11). By inspection of $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, for each node added to $\text{ToHandle}_P[\texttt{sid}]$ there is a subsequent update of $\text{SessionGraphs}_P[\texttt{sid}]$ via $\mathsf{UpdatedGraph}$ where the input set $\text{ToHandle}_P[\texttt{sid}]$ contains the new node. Since every node in the set output by $\mathsf{UpdatedGraph}$ is then removed from $\text{ToHandle}_P[\texttt{sid}]$, it then follows from Proposition 11 that the fact holds. □

We start by showing that any root is in V_P^{Global} if and only if it is also in V_P^{Local} (i.e. V_P^{Global} and V_P^{Local} contain exactly the same set of root nodes). Note that, by inspection of $\mathsf{InducedPartyGraph}^+$ (Algorithm 17), this implies that the set of root nodes output by $\mathsf{InducedPartyGraph}^+$ in both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ is exactly the same—because all root nodes are in the initial set V_0 for both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. So, once this is established we only need to consider non-roots.

Roots. Take any root node $u \in V_P^{\text{Global}}$. By definition of V_P^{Global} we have $u \in \text{Delivered}_P[\texttt{sid}]$. From Fact 2 we know u was added to $\text{ToHandle}_P[\texttt{sid}]$, and from Fact 3 it then follows that u was added to V_P^{Local}. As for the converse direction, take any root $u \in V_P^{\text{Local}}$. From Fact 4 we know u was added to $\text{ToHandle}_P[\texttt{sid}]$, and from Fact 1 we know $u \in \text{Delivered}_P[\texttt{sid}]$. From Fact 9 it follows that u was added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$. From Fact 8 we know u was added to V^{Global}. At this point we have established that $u \in \text{Delivered}_P[\texttt{sid}]$ and $u \in V^{\text{Global}}$, which by definition of V_P^{Global} implies $u \in V_P^{\text{Global}}$.

Non-root Nodes. We prove that in both $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$ it always holds (i.e. between queries to the interfaces) that:

S.1 $V_P^{\text{Local}} \subseteq V^{\text{Global}}$;
S.2 $V_P^{\text{Local}} \subseteq V_P^{\text{Global}}$; and
S.3 *(incomplete paths)* for every $u \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$, there is a (possibly root) node

$$v \in V^{\text{Global}} \setminus V_P^{\text{Global}}$$

such that there is a path from v to u

$$v \rightsquigarrow \ldots \rightsquigarrow u$$

where each node in the path is not a root.

Note that **S.2** and **S.3** (proven below) imply that the set of nodes output by function $\mathsf{InducedPartyGraph}^+$ is the same in both $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$: **S.2** implies that every node in the graph output by $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ is also in the graph output by $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$; from (a recursive application of) Proposition 14 we have that **S.3** implies that every node in $V_P^{\text{Global}} \setminus V_P^{\text{Local}}$ is not in the graph output by $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$. So, establishing **S.2** and **S.3** implies $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}} \equiv \mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$.

We first prove **S.3**. From definition of V_P^{Global} and Fact 10, we can restate it:

S.3' for every $u \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$, there is a (possibly root) node $v \in V^{\text{Global}} \setminus \text{Delivered}_P[\texttt{sid}]$ such that there is a path from v to u ($v \rightsquigarrow \ldots \rightsquigarrow u$) where each node in the path is not a root.

Since $\text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$ is proper, there is a sequence of nodes

$$v_1, \ldots, v_n$$

as in Definition 2. Assume for contradiction there is a node $u' \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$ such that for every (possibly root) node $v \in V^{\text{Global}} \setminus \text{Delivered}_P[\texttt{sid}]$ there is no path from v to u' ($v \rightsquigarrow \ldots \rightsquigarrow u'$) where each node in the path is not a root. Note that each node v_i in the sequence above is in V^{Global}, and by definition of V_P^{Global}, so is each node u'. Now take the least $j \in \{1, \ldots, n\}$ such that $v_j \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$ and for every (possibly root) node $v \in V^{\text{Global}} \setminus \text{Delivered}_P[\texttt{sid}]$ there is no path from v to v_j ($v \rightsquigarrow \ldots \rightsquigarrow v_j$) where each node in the path is not a root. Say $v_j := (\mathtt{id}, (\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})))$. We already know v_j cannot be a root—because we already established the subsets of V_P^{Local} and V_P^{Global} consisting of root nodes are the same. Since $v_j \in V_P^{\mathrm{Global}}$, it follows from definition of V_P^{Global} and from Fact 10 that $v_j \in V^{\mathrm{Global}}$ and $v_j \in \mathrm{Delivered}_P[\mathtt{sid}]$. From Fact 2, we have v_j was added to $\mathrm{ToHandle}_P[\mathtt{sid}]$. Since a node is only removed from $\mathrm{ToHandle}_P[\mathtt{sid}]$ when it is added to V_P^{Local}, and $v_j \in V_P^{\mathrm{Global}} \setminus V_P^{\mathrm{Local}}$, it follows v_j is currently in $\mathrm{ToHandle}_P[\mathtt{sid}]$. Since $v_j \in V^{\mathrm{Global}}$ and nodes are only added to V^{Global} via $\mathsf{UpdatedGraph}$—which constructs the output graph following Definition 2 (Proposition 3)—and since v_j is not a root node, then Requirement 2 implies every node in the set of v_j's acknowledgments was in either the input graph or some intermediate graph on which $\mathsf{UpdatedGraph}$ was computing, say $\mathcal{G}_i := (V_i, E_i)$: in other words, the fact that v_j was added implies

$$\textsc{IsValid}(\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_i), S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1,$$

and since v_j is not a root, from Requirement 2 we have $\mathrm{Acks} \subseteq V_i$. On the other hand, the fact that $v_j \in \mathrm{ToHandle}_P[\mathtt{sid}]$ but $v_j \notin V_P^{\mathrm{Local}}$ implies, from Fact 11:

$$\textsc{IsValid}(\mathtt{sid}, \mathsf{Extended}(\mathrm{SessionGraphs}_P[\mathtt{sid}]), S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 0.$$

We have already established $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ is proper; since $\mathcal{G}_i$ is also proper, it follows from Requirement 3 that $\mathrm{Acks} \not\subseteq V_P^{\mathrm{Local}}$. By Definition 2, since v_j is not a root and from Requirement 3, every node in $x \in \mathrm{Acks}$ must appear before v_j in the sequence $v_1, \ldots, v_n$. Taking any such $x \in \mathrm{Acks} \setminus V_P^{\mathrm{Local}}$ (which exists because we already concluded $\mathrm{Acks} \not\subseteq V_P^{\mathrm{Local}}$) we know $x = v_l$ for some $l < j$. To conclude the proof of **S.3'** (and therefore of **S.3**), consider two cases:

- $v_l \in \mathrm{ToHandle}_P[\mathtt{sid}]$, or
- $v_l \notin \mathrm{ToHandle}_P[\mathtt{sid}]$.

We now obtain a contradiction for both of these cases.

$v_l \in \mathrm{ToHandle}_P[\mathtt{sid}]$: from Fact 1 we know $v_l \in \mathrm{Delivered}_P[\mathtt{sid}]$, and since $v_l \in V^{\mathrm{Global}}$, we also know $v_l \in V_P^{\mathrm{Global}}$. Furthermore, we know $v_l \notin V_P^{\mathrm{Local}}$, which implies $v_l \in V_P^{\mathrm{Global}} \setminus V_P^{\mathrm{Local}}$. However, this is now a contradiction with our assumption v_j was the first node in the sequence $v_1, \ldots, v_n$ (because $l < j$).

$v_l \notin \mathrm{ToHandle}_P[\mathtt{sid}]$: from Fact 1 we know $v_l \notin \mathrm{Delivered}_P[\mathtt{sid}]$, and since $v_l \in V^{\mathrm{Global}}$, then $v_l \in V^{\mathrm{Global}} \setminus \mathrm{Delivered}_P[\mathtt{sid}]$. However this is a contradiction with our assumption because $v_j \in V_P^{\mathrm{Global}} \setminus V_P^{\mathrm{Local}}$ and yet there is a node in $V^{\mathrm{Global}} \setminus \mathrm{Delivered}_P[\mathtt{sid}]$, namely v_l, for which there is a path from v_l to v_j where every node in the path is not a root (this is the case, since there is an edge from v_l to v_j, so there are no nodes in the path).

To prove **S.1** and **S.2** we use induction on the queries made to $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$.

Base case: Upon $\textsc{Initialization}$

$$\text{SessionGraphs}_{\text{Global}}[\texttt{sid}] = \text{SessionGraphs}_P[\texttt{sid}] = \mathcal{G}_\emptyset,$$

so **S.1** and **S.2** trivially hold.

Induction Step: Suppose **S.1** and **S.2** hold. We prove that after any query:

- $(P' \in \mathcal{M}^H)$-$\textsc{Write}(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$,
- $(P' \in \mathcal{M}^H)$-$\textsc{Read}$,
- $(P' \in \overline{\mathcal{P}^H})$-$\textsc{Write}(\langle S \to \vec{V} \rangle, m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks}))$,
- $(P' \in \overline{\mathcal{M}^H})$-$\textsc{Read}$, or
- $\textsc{Deliver}(P', \texttt{id})$

S.1 and **S.2** still hold.

Queries $(P' \in \mathcal{P})$**-**$\textsc{Read}$**.** For both honest and dishonest parties, $\textsc{Read}$ queries have no side-effects (i.e. in the description of $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and of $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ no variable's value is modified). Therefore, after any $\textsc{Read}$ query the claim still holds.

Queries $(P' \in \mathcal{M}^H)$**-**$\textsc{Write}(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$**, for** $P' \neq P$ **.** Neither V_P^{Global} or V_P^{Local} change, as the resulting node is not added to either $\text{SessionGraphs}_P[\texttt{sid}]$, $\text{Sent}[P]$ nor $\text{Delivered}_P[\texttt{sid}]$. Therefore **S.2** holds. Since V_P^{Local} does not change, **S.1** also still holds because no node is removed from V^{Global}.

Queries $(P' \in \overline{\mathcal{P}^H})$**-**$\textsc{Write}(\langle S \to \vec{V} \rangle, m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks}))$**.** Analogous to the case of queries $(P' \in \mathcal{M}^H)$-$\textsc{Write}(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$ where $P' \neq P$. Therefore **S.1** and **S.2** hold.

Queries P**-**$\textsc{Write}(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$**.** We have already seen **S.3** holds; from induction hypothesis, **S.2** also holds, and so the graph output by $\mathsf{InducedPartyGraph}^+$ is the same for both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. This means that for both cases the $\textsc{Write}$ requirement is exactly the same, so the set of valid inputs for these queries (i.e. their domains) in $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ are the same.

Now note that the new node resulting from the query, say u, is added to both V_P^{Global} and V_P^{Local}: u is added to both $\text{Delivered}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$; by the $\textsc{Write}$ requirement u must be valid; Proposition 12 and the fact that the graph output by $\mathsf{UpdatedGraph}$ contains the input graph imply the new node is added to $\text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$—and since $u \in \text{Delivered}_P[\texttt{sid}]$, to V_P^{Global}—and to $\text{SessionGraphs}_P[\texttt{sid}]$—so, to V_P^{Local}. At this point, to establish **S.1** and **S.2** still hold after the query, it remains to argue that for any node $u' \in \text{ToHandle}_P[\texttt{sid}]$, if $u' \notin V_P^{\text{Global}}$ then u' is not added to V_P^{Local}. First note, Fact 1 implies any node in $\text{ToHandle}_P[\texttt{sid}]$ is also in $\text{Delivered}_P[\texttt{sid}]$, and it therefore suffices to prove that if $u' \notin V^{\text{Global}}$ then u' is not added to V_P^{Local}. Second, we already established that any root node is in V_P^{Global} if and only if it is also in V_P^{Local}, and so we only need to consider non-root nodes.

By induction hypothesis, **S.1** holds prior to the query (this allows us to rely on Requirement 3). From Fact 7 we know that every node in $\text{ToHandle}_P[\texttt{sid}]$ was previously added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$. Noting that in the two hybrids $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$ is only modified inside $\mathsf{AddToGraph}$, and that after (possibly) new nodes being added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$ the global graph is updated via $\mathsf{UpdatedGraph}$—all nodes in $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$ being input to $\mathsf{UpdatedGraph}$—and since a node may only be added to V_P^{Local} also via $\mathsf{UpdatedGraph}$, it follows from Requirement 3[22] that if any node is added to V_P^{Local} then it is also added to V^{Global}. This establishes **S.1** and **S.2** still hold after the query.

Queries $\textsc{Deliver}(P', \texttt{id})$**.** The only interesting case is when $P' = P$. Upon such query graph $\text{SessionGraphs}_P[\texttt{sid}]$ (and so V_P^{Local}) may only be modified via $\mathsf{UpdatedGraph}$; the set of nodes input to such $\mathsf{UpdatedGraph}$ is $\text{ToHandle}_P[\texttt{sid}]$; by Fact 1 all these nodes are in $\text{Delivered}_P[\texttt{sid}]$. It then suffices to prove that any node added to V_P^{Local} was already in V^{Global}. We proceed by contradiction: take the first node $u \in \text{ToHandle}_P[\texttt{sid}]$ that is added to V_P^{Local} during this $\textsc{Deliver}$ query but is not in V^{Global}. Here, by first we mean the first node that is not in V^{Global} but is added by $\mathsf{UpdatedGraph}$. By Fact 7, every node added to $\text{ToHandle}_P[\texttt{sid}]$ was previously added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$ in a prior query, since $\textsc{Deliver}$ queries do not modify $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$. In that prior query, $\text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$ was updated via $\mathsf{UpdatedGraph}$ with set of nodes $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$; we therefore know that $u \in \text{ToHandle}_{\text{Global}}[\texttt{sid}]$ during such query, because we assumed u was not added to V^{Global} but have already concluded that u was added to $\text{ToHandle}_{\text{Global}}[\texttt{sid}]$. This implies that in the last iteration of $\mathsf{UpdatedGraph}$, u was not valid according to predicate $\mathfrak{P}[\textsc{IsValid}]$ (otherwise u would have been added to V^{Global}. However, this is now a contradiction: since u is the first node which is not in V^{Global} that is added to V_P^{Local}, then u was valid according to predicate $\mathfrak{P}[\textsc{IsValid}]$ for that graph, say $\mathcal{G}_j$ (which is proper, because as already explained all intermediate graphs computed in $\mathsf{UpdatedGraph}$ are proper), and yet u was not valid according that predicate ($\mathfrak{P}[\textsc{IsValid}]$) for $\text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$, which from induction hypothesis **S.1** (and the fact that u is the first node added) is a supergraph of $\mathcal{G}_j$. It follows $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. ◻

B.4 Proofs of Helper Propositions

B.4.1 Proof of Proposition 1.

Proof. Consider any $u \coloneqq (\texttt{id}, (\langle S \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))) \in V^+$. Definition 2 implies there is a proper subgraph of $\mathcal{G}^+$, say $\mathcal{G'}^+ = (V'^+, E'^+)$, such that

$$\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathcal{G'}^+, S, \vec{V}, \texttt{cmd}, \text{Acks}) = 1.$$

[22] Note that graphs $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{SessionGraphs}_{\text{Global}}[\texttt{sid}]$ are proper, and that $\mathsf{UpdatedGraph}$ always constructs graphs following the definition of proper graph, Definition 2.

By Requirement 3 and since both $\mathcal{G}^+$ and $\mathcal{G}'^+$ are proper,

$$\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \text{Acks}) = \mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathcal{G}'^+, S, \vec{V}, \texttt{cmd}, \text{Acks}).$$

□

B.4.2 Proof of Proposition 2.

Proof. Follows from the definition of proper graph (Definition 2). □

B.4.3 Proof of Proposition 3.

Proof. We prove this by induction on i; for $i = 0$, the extended version of $\mathcal{G}_i$ is proper by the assumption on the input graph. For any $i \in \mathbb{N}$, assume the extended version of $\mathcal{G}_i = (V_i, E_i)$, i.e. $\mathcal{G}_i^+ = (V_i^+, E_i^+)$ is proper. We show $\mathcal{G}_{i+1}^+$ is also proper. Initially, $\mathcal{G}_{i+1}^+$ is set to $\mathcal{G}_i^+$, and therefore by assumption it is proper. For each extended node

$$u \coloneqq (\texttt{id}, (\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))),$$

helper function $\mathsf{UpdatedGraph}$ only adds u to $\mathcal{G}_{i+1}^+$ if

$$\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}_{i+1}), S, \vec{V}, \texttt{cmd}, \text{Acks}) = 1.$$

By Definition 2, since $\mathsf{Extended}(\mathcal{G}_{i+1})$ is proper, then the updated extended graph

$$\mathcal{G}_{i+1}^+ \coloneqq (\mathcal{G}_{i+1}.V^+ \cup \{u\}, \mathcal{G}_{i+1}.E^+ \cup (\text{Acks} \times \{\texttt{id}\}))$$

is also proper. □

B.4.4 Proof of Proposition 4.

Proof. First note that from Algorithm 6 the graph output by $\mathsf{InducedPartyGraph}^+$ on input $(\texttt{sid}, P)$ is a subgraph of (the extended version of) SessionGraphs[$\texttt{sid}$], which by assumption is proper. It then remains to show that this (extended) subgraph is proper, which we do via induction by analyzing Algorithm 6. Concretely, we show for each $i \in \mathbb{N}$ that the extended version of graph $\mathcal{G}_i \coloneqq (V_i, E_i \coloneqq E \cap (V_E \times V_i))$ is proper. Noting that $E_i = \bigcup_{u \in V_i} f(u)$ for $f(u) \coloneqq \text{Acks} \times \{\texttt{id}\}$ (defined as in Proposition 2), it then suffices to show that the graph induced by set of edges V_i is proper, which the rest of this proof will establish. Throughout the proof, we denote the extended version of graph SessionGraphs[$\texttt{sid}$] by $\mathcal{G}_{\texttt{sid}}^+ = (V_{\texttt{sid}}^+, E_{\texttt{sid}}^+)$.

To begin, note that V_P is a subset of the vertices of graph SessionGraphs[$\texttt{sid}$], and V_0 is the subset of V_P containing only root nodes. By Requirement 1 all root nodes are valid; by inductively applying Definition 2 to each node in V_0 and noting that $E_0 = \bigcup_{u \in V_0} f(u)$ (for f defined as in Proposition 2) it then follows that $\mathcal{G}_0^+$ (the extended version of $\mathcal{G}_0$) is proper.

Assume that, for some $i \in \mathbb{N}$, $\mathcal{G}_i^+ = (V_i^+, E_i^+)$ is proper. We now establish that $\mathcal{G}_{i+1}^+ = (V_{i+1}^+, E_{i+1}^+)$ is proper. Take any node $u \in (V_{i+1}^+ \setminus V_i^+)$; say

$$u \coloneqq (\mathtt{id}, (\langle S \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))).$$

By Algorithm 6, all of u's acknowledged nodes (i.e. Acks) are in $\mathcal{G}_i^+$. More, $\mathcal{G}_i^+$ is a subgraph of $\mathcal{G}_{\mathtt{sid}}^+$ and both extended graphs are proper. By Requirement 3 it then follows

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}_{\mathtt{sid}}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = \mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}_i^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}).$$

By Proposition 1,

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}_{\mathtt{sid}}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1,$$

so

$$\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}_i^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1,$$

which by Definition 2 implies $\mathcal{G}_i^{+\prime} = (V_i^+ \cup \{u\}, E_i^+ \cup (\mathrm{Acks} \times \{\mathtt{id}\}))$ is proper. Via an induction argument (using Definition 2) over all remaining nodes in $V_{i+1}^+ \setminus V_i^+$, the statement then follows. □

B.4.5 Proof of Proposition 5.

Proof. We proceed by induction. As base case, note that initially SessionGraphs is the empty set, and in this case all graphs are proper.

Assume that all (extended versions of the) graphs stored in SessionGraphs are proper. We will show that after any possible interaction with the ideal **ChatSessions**[$\mathfrak{P}$], all graphs stored in SessionGraphs are still proper. First, from Algorithm 6 we have that no query to interfaces $(P \in \mathcal{M}^H)$-Read, $(P \in \overline{\mathcal{M}^H})$-Read and Deliver modifies any graph stored in SessionGraphs. We now consider the two remaining cases: queries to $(P \in \mathcal{M}^H)$-Write and queries to $(P \in \overline{\mathcal{M}^H})$-Write.

Consider any query to $(P \in \mathcal{M}^H)$-Write, and let $(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \mathrm{Acks})$ be the input to the query. By assumption all graphs in SessionGraphs before the query are proper, and only SessionGraphs[sid] is modified. Concretely, the new value of SessionGraphs[sid] is the graph output by UpdatedGraph. Since the graph input to UpdatedGraph is SessionGraphs[sid], which by induction hypothesis was proper at the beginning of the query, it follows from Proposition 3 that all graphs in SessionGraphs after such query are still proper.

Now consider any query to $(P \in \overline{\mathcal{M}^H})$-Write, and let $(\langle S \to \vec{V} \rangle, m \coloneqq (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$ be the corresponding input. By Proposition 3 and the assumption that all graphs in SessionGraphs before the query are proper, the output of UpdatedGraph is proper. Again, by the definition of **ChatSessions**[$\mathfrak{P}$] (Algorithm 6), only SessionGraphs[sid] may be modified; if it is modified, it is set to the output of UpdatedGraph, which is proper. It follows that after the query all graphs stored in SessionGraphs are still proper. □

B.4.6 Proof of Proposition 6.

Proof. One can prove this by following arguments similar to the ones from the proof of Proposition 5. For any party $P \in \mathcal{P}^H$, we proceed by induction on the state of SessionGraphs stored in P's $\mathsf{ChatSessionsProt}[\mathfrak{P}]$' converter. Initially SessionGraphs is empty and therefore all graphs are proper. Assume that all (extended versions of the) graphs stored in SessionGraphs are proper. We only need to show that after any WRITE or READ queries to P's $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ converter, all graphs stored in SessionGraphs are still proper.

For a query $\text{WRITE}(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$, one can follow the same arguments used in the proof of Proposition 5, and so it follows all graphs in SessionGraphs after such query are still proper after such query. Regarding READ queries one can follow the arguments used in the proof of Proposition 5 for the case of $(P \in \overline{\mathcal{P}^H})$-WRITE operations (over $\mathtt{sid}$ in the set ToHandle). ☐

B.4.7 Proof of Proposition 7.

Proof. Follows from an argument along the lines of the proof of Proposition 6. ☐

B.4.8 Proof of Proposition 8.

Proof. We prove the two directions:

$S' \subseteq \mathcal{G}'.V \cap S$: From inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), any node $u \in S'$ must be in set $\mathcal{G}'.V$ and in set S.

$\mathcal{G}'.V \cap S \subseteq S'$: Consider an arbitrary node $u \in \mathcal{G}'.V \cap S$; first, if $u \in \mathcal{G}.V$ then it follows from Proposition 1 and inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6) that $u \in S'$; second, if $u \notin \mathcal{G}.V$ then, since $u \in \mathcal{G}'.V$, u was added to $\mathcal{G}'.V$ by $\mathsf{UpdatedGraph}$ and therefore by inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), $u \in S'$. ☐

B.4.9 Proof of Proposition 9.

Proof. We prove this by contradiction. Since $\mathcal{G} = (V, E)$ is proper, letting $n = |V|$, by Definition 2 there is an ordered sequence of nodes $u_1, \ldots, u_n$ such that, letting $\mathcal{G}_0 \coloneqq (V_0, E_0) = (\emptyset, \emptyset)$, and letting for $i = 0, \ldots, n-1$,

$$\mathcal{G}_{i+1} \coloneqq (V_i \cup \{u_{i+1}.\mathtt{id}\}, E_i \cup (u_{i+1}.\text{Acks} \times \{u_{i+1}.\mathtt{id}\})),$$

it holds $\text{IsValid}(u_{i+1}.\mathtt{sid}, {\mathcal{G}_i}^+, u_{i+1}.S, u_{i+1}.\vec{V}, u_{i+1}.\mathtt{cmd}, u_{i+1}.\text{Acks}) = 1$. For some set $V_S \subseteq V$, let $(\mathcal{G}_S, S') \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S)$, and furthermore let $(\mathcal{G}_{V_S}, {S_{V_S}}') \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S \cup V_S)$. By inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), graphs $\mathcal{G}_S$ and $\mathcal{G}_{V_S}$ are constructed according to Definition 2, so there are sequences of nodes $u^S_{n+1}, \ldots, u^S_{(n+|S'|)}$ and $u^{S\cup V_S}_{n+1}, \ldots, u^{S\cup V_S}_{(n+|{S_{V_S}}'|)}$ (where each node u^S_j is in set S and each node $u_l^{S\cup V_S}$ is in set $S\cup V_S$) such that, for $j = n, \ldots, (n + |S'|)-1$ and for $l = n, \ldots, (n + |S_{V_S}{}'|) - 1$, letting

$$\mathcal{G}_{j+1}^S \coloneqq (V_j^S \cup \{u_{j+1}^S.\texttt{id}\}, E_j^S \cup (u_{j+1}^S.\text{Acks} \times \{u_{j+1}^S.\texttt{id}\})),$$
$$\mathcal{G}_{l+1}^{S\cup V_S} \coloneqq (V_l^{S\cup V_S} \cup \{u_{l+1}^{S\cup V_S}.\texttt{id}\}, E_l^{S\cup V_S} \cup (u_{l+1}^{S\cup V_S}.\text{Acks} \times \{u_{l+1}^{S\cup V_S}.\texttt{id}\})),$$

it holds that

$$\textsc{IsValid}(u_{j+1}^S.\texttt{sid}, \left(\mathcal{G}_j^S\right)^+, u_{j+1}^S.S, u_{j+1}^S.\vec{V}, u_{j+1}^S.\texttt{cmd}, u_{j+1}^S.\text{Acks}) = 1,$$
$$\textsc{IsValid}(u_{l+1}^{S\cup V_S}.\texttt{sid}, \left(\mathcal{G}_l^{S\cup V_S}\right)^+, u_{l+1}^{S\cup V_S}.S, u_{l+1}^{S\cup V_S}.\vec{V}, u_{l+1}^{S\cup V_S}.\texttt{cmd}, u_{l+1}^{S\cup V_S}.\text{Acks}) = 1.$$

For contradiction, assume $\mathcal{G}_S \neq \mathcal{G}_{V_S}$; so, either $V_{\mathcal{G}_S} \setminus V_{\mathcal{G}_{V_S}} \neq \emptyset$ or $V_{\mathcal{G}_{V_S}} \setminus V_{\mathcal{G}_S} \neq \emptyset$. We obtain a contradiction for each case.

$V_{\mathcal{G}_S} \setminus V_{\mathcal{G}_{V_S}} \neq \emptyset$: consider the first node u_j^S in the sequence $u_{n+1}^S, \ldots, u_{(n+|S'|)}^S$ that is not in $V_{\mathcal{G}_{V_S}}$; u_j^S is not a root because this would contradict the assumption that $\mathfrak{P}$ satisfies Requirement 1. Given u_j^S is not a root, since

$$\textsc{IsValid}(u_j^S.\texttt{sid}, \left(\mathcal{G}_{j-1}^S\right)^+, u_j^S.S, u_j^S.\vec{V}, u_j^S.\texttt{cmd}, u_j^S.\text{Acks}) = 1,$$

from Requirement 2 it follows $u_j^S.\text{Acks} \subseteq V_{j-1}^S$. By assumption u_j^S is the first node in the sequence, so all prior nodes are in $V_{\mathcal{G}_{V_S}}$, implying $V_{j-1}^S \subseteq V_{\mathcal{G}_{V_S}}$. Since both $\mathcal{G}_{V_S}$ and $\mathcal{G}_{j-1}^S$ are proper graphs, it then follows from Requirement 3

$$\begin{aligned}&\textsc{IsValid}(u_j^S.\texttt{sid}, \mathcal{G}_{V_S}{}^+, u_j^S.S, u_j^S.\vec{V}, u_j^S.\texttt{cmd}, u_j^S.\text{Acks})\\ &= \textsc{IsValid}(u_j^S.\texttt{sid}, \left(\mathcal{G}_{j-1}^S\right)^+, u_j^S.S, u_j^S.\vec{V}, u_j^S.\texttt{cmd}, u_j^S.\text{Acks})\end{aligned}$$

and so

$$\textsc{IsValid}(u_j^S.\texttt{sid}, \mathcal{G}_{V_S}{}^+, u_j^S.S, u_j^S.\vec{V}, u_j^S.\texttt{cmd}, u_j^S.\text{Acks}) = 1.$$

However, from inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6) this is a contradiction with the fact that in the last iteration of $\mathsf{UpdatedGraph}(\mathcal{G}, S \cup V_S)$ node u_j^S was not added to the output graph $\mathcal{G}_{V_S}$. □

$V_{\mathcal{G}_{V_S}} \setminus V_{\mathcal{G}_S} \neq \emptyset$: Follows from an argument analogous to the one for case above, noting that the graph input to $\mathsf{UpdatedGraph}$ is always a subgraph of the output graph (so each node $u \in V_S$ is in the output graph $\mathcal{G}_S$). □

□

B.4.10 Proof of Proposition 10.

Proof. It is sufficient to prove for the case of 2 sets as a simple induction argument then implies the case for $n > 2$. Consider some proper graph $\mathcal{G}_1 \coloneqq$

$(V_{\mathcal{G}_1}, E_{\mathcal{G}_1})$, some set S of nodes, and any two sets S_1 and S_2 such that $S = S_1 \cup S_2$. Furthermore, let

$$\begin{aligned}(\mathcal{G}_2 \coloneqq (V_{\mathcal{G}_2}, E_{\mathcal{G}_2}), {S_2}') &\coloneqq \mathsf{UpdatedGraph}(\mathcal{G}_1, S_1),\\ (\mathcal{G}_3 \coloneqq (V_{\mathcal{G}_3}, E_{\mathcal{G}_3}), {S_3}') &\coloneqq \mathsf{UpdatedGraph}(\mathcal{G}_2, S_2 \cup (S_1 \setminus {S_2}')),\\ (\mathcal{G}' \coloneqq (V_{\mathcal{G}'}, E_{\mathcal{G}'}), S') &\coloneqq \mathsf{UpdatedGraph}(\mathcal{G}_1, S).\end{aligned}$$

We want to show $(\mathcal{G}_3, {S_2}' \cup {S_3}') = (\mathcal{G}', S')$.

To start we show $\mathcal{G}' = \mathcal{G}_3$ implies $S' = {S_2}' \cup {S_3}'$. From Proposition 8

$$\begin{aligned}{S_3}' &= V_{\mathcal{G}_3} \cap \big(S_2 \cup (S_1 \setminus {S_2}')\big),\\ S' &= V_{\mathcal{G}'} \cap S = V_{\mathcal{G}'} \cap (S_1 \cup S_2).\end{aligned}$$

Noting that 1. from Proposition 8 ${S_2}' = V_{\mathcal{G}_2} \cap S_1$, and; 2. since the graph $\mathcal{G}_2$ input to $\mathsf{UpdatedGraph}$ is proper, then $V_{\mathcal{G}_2} \subseteq V_{\mathcal{G}_3}$:

$$\begin{aligned}{S_2}' \cup {S_3}' &= {S_2}' \cup \Big(V_{\mathcal{G}_3} \cap \big(S_2 \cup (S_1 \setminus {S_2}')\big)\Big)\\ &= (V_{\mathcal{G}_3} \cap S_2) \cup \Big(({S_2}' \cup V_{\mathcal{G}_3}) \cap ({S_2}' \cup (S_1 \setminus {S_2}'))\Big)\\ &\overset{(2)}{=} (V_{\mathcal{G}_3} \cap S_2) \cup \Big(V_{\mathcal{G}_3} \cap ({S_2}' \cup (S_1 \setminus {S_2}'))\Big)\\ &\overset{(1)}{=} (V_{\mathcal{G}_3} \cap S_2) \cup \big(V_{\mathcal{G}_3} \cap S_1\big)\\ &= V_{\mathcal{G}'} \cap (S_1 \cup S_2).\end{aligned}$$

At this point we only need to establish $V_{\mathcal{G}_3} = V_{\mathcal{G}'}$, as Proposition 2 then implies $\mathcal{G}_3 = \mathcal{G}'$. First note that since ${S_2}' \subseteq V_{\mathcal{G}_2}$, for

$$({\mathcal{G}_3}' \coloneqq (V_{{\mathcal{G}_3}'}, E_{{\mathcal{G}_3}'}), S'_{{\mathcal{G}_3}'}) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}_2, S_2 \cup S_1), \tag{B.1}$$

Proposition 9 implies ${\mathcal{G}_3}' = \mathcal{G}_3$. So, we only need to prove that $V_{{\mathcal{G}_3}'} = V_{\mathcal{G}'}$.

The argument used in the proof of Proposition 9 can be used here too. Since $\mathcal{G}_1$ is proper, and letting $n \coloneqq |V_{\mathcal{G}_1}|$, by Definition 2 there is an ordered sequence of nodes $u_1, \ldots, u_n$ such that, letting $\mathcal{G}_0 \coloneqq (V_0, E_0) = (\emptyset, \emptyset)$, and letting for $i = 0, \ldots, n-1$, $\mathcal{G}_{i+1} \coloneqq (V_i \cup \{u_{i+1}.\mathtt{id}\}, E_i \cup (u_{i+1}.\mathrm{Acks} \times \{u_{i+1}.\mathtt{id}\}))$, we have $\textsc{IsValid}(u_{i+1}.\mathtt{sid}, {\mathcal{G}_i}^+, u_{i+1}.S, u_{i+1}.\vec{V}, u_{i+1}.\mathtt{cmd}, u_{i+1}.\mathrm{Acks}) = 1$. By inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), graphs $\mathcal{G}'$, $\mathcal{G}_2$ and ${\mathcal{G}_3}'$ are constructed according to Definition 2, meaning there are sequences of nodes

$$\begin{aligned}&u^{S'}_{n+1}, \ldots, u^{S'}_{(n+|S'|)}\\ &u^{{S_2}'}_{n+1}, \ldots, u^{{S_2}'}_{(n+|{S_2}'|)}\\ &u^{{\mathcal{G}_3}'}_{(n+|{S_2}'|)+1}, \ldots, u^{{\mathcal{G}_3}'}_{(n+|{S_2}'|+|S'_{{\mathcal{G}_3}'}|)}\end{aligned}$$

($S'_{{\mathcal{G}_3}'}$ is defined in Equation B.1) such that, for $i = n, \ldots, (n + |S'|) - 1$, for $j = n, \ldots, (n + |{S_2}'|) - 1$, and for $l = (n + |{S_2}'|), \ldots, (n + |{S_2}'| + |S'_{{\mathcal{G}_3}'}|) - 1$,

$$\begin{aligned}\mathcal{G}^{S'}_{i+1} &\coloneqq (V^{S'}_i \cup \{u^{S'}_{i+1}.\mathtt{id}\}, E^{S'}_i \cup (u^{S'}_{i+1}.\mathrm{Acks} \times \{u^{S'}_{i+1}.\mathtt{id}\})),\\ \mathcal{G}^{{S_2}'}_{j+1} &\coloneqq (V^{{S_2}'}_j \cup \{u^{{S_2}'}_{j+1}.\mathtt{id}\}, E^{{S_2}'}_j \cup (u^{{S_2}'}_{j+1}.\mathrm{Acks} \times \{u^{{S_2}'}_{j+1}.\mathtt{id}\})),\\ \mathcal{G}^{{\mathcal{G}_3}'}_{l+1} &\coloneqq (V^{{\mathcal{G}_3}'}_l \cup \{u^{{\mathcal{G}_3}'}_{l+1}.\mathtt{id}\}, E^{{\mathcal{G}_3}'}_l \cup (u^{{\mathcal{G}_3}'}_{l+1}.\mathrm{Acks} \times \{u^{{\mathcal{G}_3}'}_{l+1}.\mathtt{id}\})),\end{aligned}$$

it holds that

$$\textsc{IsValid}(u_{i+1}^{S'}.\texttt{sid}, \left(\mathcal{G}_i^{S'}\right)^+, u_{i+1}^{S'}.S, u_{i+1}^{S'}.\vec{V}, u_{i+1}^{S'}.\texttt{cmd}, u_{i+1}^{S'}.\text{Acks}) = 1,$$
$$\textsc{IsValid}(u_{j+1}^{S_2{}'}.\texttt{sid}, \left(\mathcal{G}_j^{S_2{}'}\right)^+, u_{j+1}^{S_2{}'}.S, u_{j+1}^{S_2{}'}.\vec{V}, u_{j+1}^{S_2{}'}.\texttt{cmd}, u_{j+1}^{S_2{}'}.\text{Acks}) = 1,$$
$$\textsc{IsValid}(u_{l+1}^{\mathcal{G}_3{}'}.\texttt{sid}, \left(\mathcal{G}_l^{\mathcal{G}_3{}'}\right)^+, u_{l+1}^{\mathcal{G}_3{}'}.S, u_{l+1}^{\mathcal{G}_3{}'}.\vec{V}, u_{l+1}^{\mathcal{G}_3{}'}.\texttt{cmd}, u_{l+1}^{\mathcal{G}_3{}'}.\text{Acks}) = 1.$$

We now show $V_{\mathcal{G}_2} \setminus V_{\mathcal{G}'} = \emptyset$, $V_{\mathcal{G}_3{}'} \setminus V_{\mathcal{G}'} = \emptyset$ and $V_{\mathcal{G}'} \setminus V_{\mathcal{G}_3{}'} = \emptyset$. Note that $V_{\mathcal{G}_3{}'} \setminus V_{\mathcal{G}'} = \emptyset$ and $V_{\mathcal{G}'} \setminus V_{\mathcal{G}_3{}'} = \emptyset$ together imply $V_{\mathcal{G}_3} = V_{\mathcal{G}'}$. As in the proof of Proposition 9, we proceed by contradiction:

$V_{\mathcal{G}_2} \setminus V_{\mathcal{G}'} = \emptyset$: Suppose this is not the case and consider the first node $u_j^{S_2{}'}$ in the sequence $u_{n+1}^{S_2{}'}, \ldots, u_{(n+|S_2{}'|)}^{S_2{}'}$ such that $u_j^{S_2{}'} \notin V_{\mathcal{G}'}$. First, $u_j^{S_2{}'}$ cannot be a root node, as otherwise this would imply $\mathfrak{P}$ does not satisfy Requirement 1. Since $u_j^{S_2{}'}$ is not a root and noting that

$$\textsc{IsValid}(u_j^{S_2{}'}.\texttt{sid}, \left(\mathcal{G}_{j-1}^{S_2{}'}\right)^+, u_j^{S_2{}'}.S, u_j^{S_2{}'}.\vec{V}, u_j^{S_2{}'}.\texttt{cmd}, u_j^{S_2{}'}.\text{Acks}) = 1,$$

it follows from Requirement 2 that $u_j^{S_2{}'}.\text{Acks} \subseteq \mathcal{G}_{j-1}^{S_2{}'}.V$. Since by assumption $u_j^{S_2{}'}$ is the first node in the sequence, then all nodes in the sequence prior to $u_j^{S_2{}'}$ are in $V_{\mathcal{G}'}$, implying $V_{j-1}^{S_2{}'} \subseteq V_{\mathcal{G}'}$. Since both $\mathcal{G}'$ and $\mathcal{G}_{j-1}^{S_2{}'}$ are proper graphs, it then follows from Requirement 3

$$\begin{aligned}&\textsc{IsValid}(u_j^{S_2{}'}.\texttt{sid}, \mathcal{G}'^+, u_j^{S_2{}'}.S, u_j^{S_2{}'}.\vec{V}, u_j^{S_2{}'}.\texttt{cmd}, u_j^{S_2{}'}.\text{Acks})\\ &= \textsc{IsValid}(u_j^{S_2{}'}.\texttt{sid}, \left(\mathcal{G}_{j-1}^{S_2{}'}\right)^+, u_j^{S_2{}'}.S, u_j^{S_2{}'}.\vec{V}, u_j^{S_2{}'}.\texttt{cmd}, u_j^{S_2{}'}.\text{Acks}).\end{aligned}$$

and so

$$\textsc{IsValid}(u_j^{S_2{}'}.\texttt{sid}, \mathcal{G}'^+, u_j^{S_2{}'}.S, u_j^{S_2{}'}.\vec{V}, u_j^{S_2{}'}.\texttt{cmd}, u_j^{S_2{}'}.\text{Acks}) = 1.$$

However, from inspection of UpdatedGraph (Algorithm 6) this is a contradiction with the fact that in the last iteration of UpdatedGraph node $u_j^{S_2{}'}$ was not added (because $u_j^{S_2{}'} \in S_1 \subseteq S$). $\square$

$V_{\mathcal{G}_3{}'} \setminus V_{\mathcal{G}'} = \emptyset$: One can prove this by noting that $V_{\mathcal{G}_2} \subseteq V_{\mathcal{G}_3{}'}$—which follows from inspection of UpdatedGraph, Algorithm 6—by relying on the fact that $V_{\mathcal{G}_2} \subseteq V_{\mathcal{G}'}$ (proven above)—and following an argument analogous to the one above. $\square$

$V_{\mathcal{G}'} \setminus V_{\mathcal{G}_3{}'} = \emptyset$: Similar to the step above. $\square$

$\square$

B.4.11 Proof of Proposition 11.

Proof. Follows from inspection of UpdatedGraph (Algorithm 6): consider any node

$$u \coloneqq (\texttt{id}, (\langle P \to \vec{R}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))) \in S \setminus S'.$$

If it were the case that

$$\mathfrak{P}[\textsc{IsValid}](\texttt{sid}, \mathcal{G}^+, P, \vec{R}, \texttt{cmd}, \text{Acks}) = 1,$$

u would be added in the last iteration of UpdatedGraph. $\square$

B.4.12 Proof of Proposition 12.

Proof. First, by assumption we know $\mathcal{G} \coloneqq (V, E)$ is proper. In the following, let $(\mathcal{G}_{\mathtt{id}} \coloneqq (V_{\mathtt{id}}, E_{\mathtt{id}}), S_{\mathtt{id}}) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, \{\mathtt{id}\})$. By inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), since $\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \mathrm{Acks}) = 1$, $\mathtt{id}$ is added to both the output graph—i.e. $\mathtt{id} \in V_{\mathtt{id}}$—and the output set $S_{\mathtt{id}}$. Since only nodes in the set input to $\mathsf{UpdatedGraph}$ may be added to the output graph, we have $V_{\mathtt{id}} = V \cup \{\mathtt{id}\}$ and $S_{\mathtt{id}} = \{\mathtt{id}\}$; by definition of $\mathsf{UpdatedGraph}$ we also have $E_{\mathtt{id}} = E \cup (\mathrm{Acks} \times \{\mathtt{id}\})$, and therefore $\mathcal{G}_{\mathtt{id}} = \mathcal{G}'$. By definition of $\mathcal{G}_1$ and S_1'', we have $(\mathcal{G}_1, S_1'') \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}', S')$. The result then follows from Proposition 10 by considering sets $S_1 \coloneqq \{\mathtt{id}\}$ and $S_2 \coloneqq S'$. ◻

B.4.13 Proof of Proposition 13.

Proof. Regarding $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ it follows from Proposition 5 and by following the sequence of hybrids

$$\mathbf{ChatSessions}[\mathfrak{P}] \rightsquigarrow \mathbf{H}^{\mathbf{IW}}_{\mathbf{1}} \rightsquigarrow \mathbf{H}^{\mathbf{IW}}_{\mathbf{2}} \rightsquigarrow \mathbf{H}^{\mathbf{IW}}_{\mathbf{3}} \rightsquigarrow \mathbf{H}^{\mathbf{IW}}_{\mathbf{4}} \rightsquigarrow \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$$

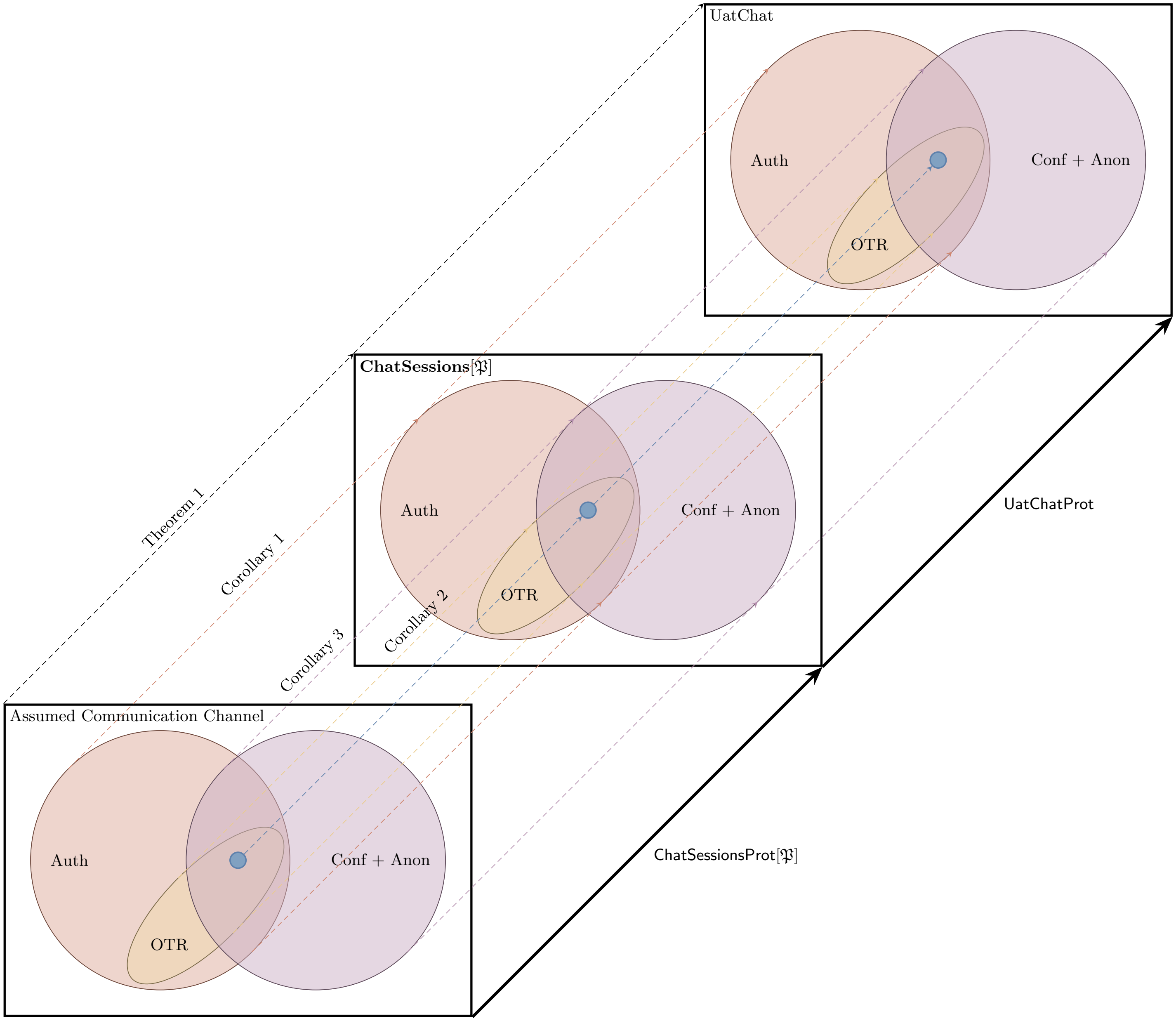
that for each $\mathtt{sid}$, graph $\mathrm{SessionGraphs}_{\mathrm{Global}}[\mathtt{sid}]$ is proper.

Regarding $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$, we prove by induction on the queries made to $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$. Upon $\textsc{Initialization}$, for each party $P \in \mathcal{M}^H$, we have $\mathrm{SessionGraphs}_P = \emptyset$, so trivially all graphs are proper. Consider any query to one of $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$'s interfaces. First, note that only $(P \in \mathcal{M}^H)$-$\textsc{Write}$ and $\textsc{Deliver}$ queries may actually modify any graph $\mathrm{SessionGraphs}_P[\mathtt{sid}]$. Note that if any such graph is modified, then it is set to the graph output by $\mathsf{UpdatedGraph}$; note also that the graph input to $\mathsf{UpdatedGraph}$ is proper (induction hypothesis). It then follows from Proposition 3 that after any such query, each graph $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ is still proper. ◻

B.4.14 Proof of Proposition 14.

Proof. Follows from the definition of $\mathsf{InducedPartyGraph}^+$: non-root nodes are only added to the output set if all their predecessors are already in that set. ◻

C "Zoomed-in" version of Figure 1



UatChat
Auth
Conf + Anon
OTR
ChatSessions[𝔓]
Auth
Conf + Anon
OTR
Assumed Communication Channel
Auth
Conf + Anon
OTR
Theorem 1
Corollary 1
Corollary 2
Corollary 3
UatChatProt
ChatSessionsProt[𝔓]