Stateful Communication with Malicious Parties

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. Cryptography's most common use is secure communication—e.g. Alice can use encryption to hide the contents of the messages she sends to Bob (confidentiality) and can use signatures to assure Bob she sent these messages (authenticity). While one typically considers *stateless* security guarantees—for example a channel that Alice can use to send messages securely to Bob—one can also consider *stateful* ones—e.g. 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.

We introduce a modular abstraction for stateful group communication, called *Chat Sessions*, which captures security guarantees that are achievable in fully asynchronous settings when one makes no party-honesty assumptions: anyone (including group members themselves) can be fully dishonest. Our abstraction is parameterized by (and enforces) a permissions policy that defines what operations parties have the right to perform in a given chat state. We show how to *construct*, *use* and *extend* Chat Sessions.

Our *construction* is fully decentralized (in particular, it need not a delivery service), does not incur additional interaction between chat participants (other than what is inherent from chat operations like sending a message) and liveness depends *solely* on messages being delivered.

A key feature of Chat Sessions is modularity: we *extend* Chat Sessions to capture *authenticity*, *confidentiality*, *anonymity* and *off-the-record*, and show our construction provides these guarantees if the underlying communication channels do too. We complement this by proving Maurer et al.'s Multi-Designated Receiver Signed Public Key Encryption scheme (Eurocrypt '22) constructs matching communication channels (i.e. with all these guarantees).

We *use* Chat Sessions to construct *UatChat*: a simple and equally modular messaging application. Since *UatChat* preserves each of the guarantees mentioned above, this means we give the first fully Off-The-Record messaging application: parties can plausibly deny not only having sent any messages but even of being aware of a chat's existence.

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

1 Introduction

Current works on secure messaging focus on so-called Forward Secrecy (FS) and Post-Compromise Security (PCS) notions [3–10, 15, 16, 18, 21, 28, 30] which aim at providing rather strong confidentiality guarantees in settings where users' devices may get compromised. Intuitively, these notions capture the secrecy 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 [32, Theorem 1]. Indeed, this is consistent with the setting considered in the messaging literature—which, despite significant progress on tolerating stronger and stronger attacks, still assumes the honesty of all group members.

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].

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 [4, 6–9, 15, 21]—the delivery server. 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 [4, 6–10, 15, 21, 30]. Even worse, this delivery party can also perform group-split attacks—even in works that consider malicious insiders such as [8, 10, 11]. This has naturally motivated the study of fully decentralized protocols (e.g. [11, 22, 43, 44]) that do not rely on a delivery party, thus avoiding such group-splitting/fork attacks. However, these protocols still do not prevent malicious parties from performing group-split attacks (i.e. forks) [11, 22, 44], leading to the following natural question: *What guarantees can a messenger achieve when any party (group member or not) can be fully malicious?*

This paper takes a fresh (and radically different) approach to the study of messaging applications. Our main contribution is the introduction of a new abstraction for stateful communication—*Chat Sessions*—that captures the evolving state of a chat. We give a formal definition of our abstraction—by means of a composable security notion—show how to construct, use and extend it, and in doing so we address many problems left open in the literature. Regarding messaging applications, our abstraction yields the first messenger that is fully off-the-record—wherein parties can not only plausibly deny having sent any particular message but also having participated in a chat altogether—that is fully decentralized—our construction does not require any central party (not even to order the messages sent by parties)—and the first which cryptographically enforces an arbitrary access control policy—guaranteeing chat members only perform the operations for which they have permissions. No messenger provides any single of these guarantees if an arbitrary subset of group members is malicious, whereas ours provides them all simultaneously.

1.1 Overview of Contributions

1.1.1 Chat Sessions Abstraction.

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, 11, 22, 43, 44]. 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 (i.e. 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.[4]

Consistency. For each chat session there is a unique digraph $\mathcal{G}_{\text{Global}} \coloneqq (V, E)$[5]—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

[4] 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.

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

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 for which all parties have permissions [4, 7, 10, 13, 42]. 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, 42]. 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 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).[6]

Fine-Grained Modularity. A central feature of our chat sessions abstraction is its modularity: while neither authenticity, confidentiality, anonymity nor off-the-record are captured by the base chat sessions abstraction, it is easy to extend it to provide (any subset of) such additional guarantees. For example, authenticity can be simply captured by disallowing dishonest parties to impersonate honest ones in writing messages, confidentiality can be captured by hiding contents of messages sent by an honest sender to a vector of all-honest receivers, and anonymity can be captured similarly, by hiding the identity of the sender and of each of the receivers. (In fact by also following this modular approach to model off-the-record, we obtain *simpler yet stronger*, relative to [34], composable security notions for MDRS-PKE schemes, as explained ahead). This type of modularity not only allows our abstraction to be clean and easy to reason about (without compromising on generality—e.g. in contrast to current messaging applications, we do not fix a particular group management policy—and without compromising on the security guarantees it provides) but it also allows for a cleaner understanding (and modeling) of the additional guarantees.

Stronger Security Statements: Another advantage of this modularity is that it allows for stronger statements (regarding the additional security guarantees) with surprisingly simple proofs.[7] In fact, one can think of our results as showing that the security guarantees from the underlying (stateless) communication channels are *lifted* to chat sessions (Figure 2 in the appendix illustrates this); the only

[6] 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.

[7] See, for example, the proof of Corollary 3, which states that our construction preserves the confidentiality and anonymity guarantees of the underlying (assumed) channels (proof in Appendix Section C.4).

assumptions that seem inherently necessary (from the underlying channels) are consistency and replay-protection.[8]

Post-Compromise Forward Secrecy (PCFS): Despite being outside the scope of our work, we note that PCFS is a type of confidentiality guarantee, and as such it can be modeled similarly to how we model confidentiality and anonymity—although the resulting security model will be inherently more involved due to its significantly more complex setting and confidentiality guarantees. While to the best of our knowledge there has been no work modeling PCFS (for groups) in a composable framework[9]—let alone a model additionally capturing consistency and replay protection—we note that if one models (and achieves, via a construction) such type of confidentiality, then such guarantee can be lifted to chat sessions (similarly to the basic type of confidentiality guarantee we consider).

Efficiency Advantages: An important property one expects from a messenger is efficiency, both in terms of the time to encrypt and decrypt ciphertexts, but also, and, perhaps, especially, in terms of achieving ciphertext sizes that scale (ideally) logarithmically with the size of a group. Indeed efficiency has been a main focus in the messaging literature [3, 4, 8–10, 21, 30]. On this regard we make two points. First, our chat sessions construction is very efficient: it does not use any cryptographic primitives nor requires any expensive computation. Second, while we only provide a single construction of the communication channels used by our chat sessions construction, and this construction is based on MDRS-PKE schemes—for which ciphertext sizes (and hence encryption and decryption times) are inherently linear with the number of receivers (see [23, Theorem 1])—we note that if one is not willing to pay the extra price that is required for such strong security guarantees, and thanks to modularity, 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). All in all, what modularity buys us here is some sort of independence from the underlying types of scheme being used.

1.1.2 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 guaranteed to be consistent (in the sense explained before). Our messenger allows parties to 1. create chats with a given roster of participants (defined by a vector of parties); 2. propose adding/removing parties to/from existing chats; 3. vote on proposals; and 4. write messages—which may include a set of prior commands the message is "replying to". As required by our chat sessions abstraction, UatChat defines a

[8] Note that an insecure channel provides both these guarantees, as does the type of authenticated channel one can construct from strongly unforgeable signatures (with deterministic verification).

[9] For the case of two parties there are works in both the Universal Composability and Constructive Cryptography frameworks [16, 18, 29].

permissions policy, denoted $\mathfrak{U}$, which at a high level enforces all group members must unanimously agree on a group modification proposal for it to take effect. For example, to add a party P' to a chat, say with current roster $\vec{G}$, a party $P \in \vec{G}$ needs to propose adding P' and then all parties in $\vec{G}$ 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.)

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.[10] 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 would only consider himself part of the group once Alice would acknowledge it, which would never occur (so Bob would not succeed in deceiving 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, suppose that a party P_1 just created a chat with a vector of (all honest) parties $\vec{G} = (P_1, P_2, P_3, P_4)$. 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 [12, 25, 26, 40], 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 to take 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

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

messaging applications, perhaps by leveraging what is known from communities working on concurrent/parallel computing.

1.1.3 Application Semantics of Maurer et al.'s MDRS-PKE [35].

We introduce the first composable notions for MDRS-PKE schemes. Our notions are defined in (a rather simplified variant of) the Constructive Cryptography framework[11] and capture security in the (rather strong) setting considered in [19]—where even the secret keys of honest senders leak to a judge. We introduce these notions to obtain an instantiation of Chat Sessions providing authenticity, off-the-record, confidentiality and anonymity, which in turn yields the first fully off-the-record messaging application wherein parties can plausibly deny not only having sent any message, but also having the knowledge of a chat's existence. We emphasize this is only possible thanks to the modularity of our Chat Sessions abstraction, of our MDRS-PKE composable notions and of Uatchat. Concretely, it follows from Theorem 1 and Corollaries 1, 2 and 3 that when Chat Sessions is built on the MDRS-PKE ideal functionality we define it provides off-the-record, confidentiality and anonymity guarantees in case the judge is dishonest, and additionally gives authenticity guarantees if the judge is honest. As explained in Section 5.2.2, all these guarantees analogously carry over to Uatchat (and hence our claim of giving the first off-the-record messenger).

We introduce Forgery Invalidity—a game-based security notion that seems necessary for giving application semantics to the MDRS-PKE game-based notions from [19][12]—and give tight reductions from breaking the composable security of an MDRS-PKE scheme to breaking one of the game-based notions from [19] or Forgery Invalidity. Finally, we give tight reductions for the Forgery Invalidity of Maurer et al.'s MDRS-PKE construction [35], thus filling-in the gap of [19] by proving the existence of a tightly secure MDRS-PKE scheme—namely, Maurer et al.'s MDRS-PKE construction [35]—providing all the necessary guarantees for composable security; concretely, we: 1. introduce an analogous Forgery Invalidity notion for MDVS schemes; 2. prove Maurer et al.'s scheme [35] satisfies the new notion if the underlying MDVS scheme satisfies the analogous one; and 3. give tight reductions from breaking the Forgery Invalidity of Chakraborty et al.'s MDVS [19] to the security of their construction's building blocks.[13] Put together, this means we give a full-fledged and fully decentralized messaging application providing strong security guarantees (but not PCFS) that can be instantiated from building blocks all of which known to have instantiations with tight reductions to standard assumptions. (Figure 1 illustrates these contributions.)

Related Work: [34] introduces composable notions for Multi-Designated Verifier Signatures (MDVS)—a closely related type of scheme that does not provide

[11] We do not know how to define them in other frameworks; see [34] for a discussion.

[12] In the sense that we do not know how to prove the composable security of an MDRS-PKE scheme without relying on Forgery Invalidity for the setting where the secret keys of honest senders leak [19].

[13] Our reduction need not any additional guarantees from the scheme's building blocks.

confidentiality nor anonymity guarantees [35]. Our composable notions are related to the ones from [34], but have the following key differences: 1. we capture a stronger off-the-record guarantee—honest parties are allowed to read incoming messages, in contrast to [34]; 2. we consider the stronger setting introduced in [19] where secret keys of honest senders leak; and 3. our notions are significantly simpler [34].

2 Preliminaries

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 $\vec{x}$: $\mathrm{Set}(\vec{x}) \coloneqq \{x_i \mid x_i \in \vec{x}\}$. For a set/alphabet S, we denote the set of non-empty vectors/strings over S by S^+. For a vector of parties $\vec{V}$, we denote by $\vec{v}$ the corresponding vector of public keys, and for $i \in \{1, \ldots, |\vec{V}|\}$, v_i is party V_i's public key. 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 Application Semantics for MDRS-PKE Schemes

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

Modeling Access Control via Repositories. Similarly to [34], we model access control via repositories. A repository contains a set of registers and a (corresponding) set of register identifiers IdSet; a register is a pair $\texttt{reg} = (\texttt{id}, m)$, where $\texttt{id}$ is the register's identifier (uniquely identifying it among all repositories), and m is a message. We consider two types of repository access rights: *read access* and *write access*.[14] We denote by $\mathcal{W}$ and $\mathcal{R}$ the sets of parties with write and read access to a repository **rep**, respectively; to make the access permissions explicit we write $\mathbf{rep}_{\mathcal{R}}^{\mathcal{W}}$, but otherwise simply write **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 $\mathbf{AUT}_{\{B,E\}}^{\{A\}}$. The semantics of atomic repositories is defined in Algorithm 1.[15]

[14] This is in contrast to [34], which additionally considers *copy access*.

[15] As needed to capture the Off-The-Record guarantee, the repository semantics capture sender anonymity: for a repository $\mathbf{rep}_{\mathcal{R}}^{\mathcal{W}}$, readers do not learn who wrote (among the parties in $\mathcal{W}$) each of the repository's messages.

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

```
◇ INITIALIZATION
  IdSet ← $\emptyset$

▷ ($P \in \mathcal{W}$)-WRITE($m$)
  id ← NEWREGISTER($m$)
  IdSet ← IdSet $\cup$ {id}
  OUTPUT(id)

▷ ($P \in \mathcal{R}$)-READ
  list ← $\emptyset$
  for id $\in$ IdSet :
    list ← list $\cup$ {(id, GETMESSAGE(id))}
  OUTPUT(list)
```

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

```
  ▷ ($P \in \mathcal{P}$)-WRITE($\mathbf{rep_i}_{\mathcal{R}_i}^{\mathcal{W}_i}$, $m$)
Require: ($P \in \mathcal{W}_i$)
    OUTPUT($\mathbf{rep_i}$-WRITE($m$))

  ▷ ($P \in \mathcal{P}$)-READ
    list ← $\emptyset$
    for $\mathbf{rep_i} \in \mathbf{REP}$ with $P \in \mathcal{R}_i$ :
      for (id, $m$) $\in$ $\mathbf{rep_i}$-READ :
        list ← list $\cup$ (id, ($\mathbf{rep_i}$, $m$))
    OUTPUT(list)
```

Following [34], to model that parties may have access to multiple repositories—say $\mathbf{rep_1}_{\mathcal{R}_1}^{\mathcal{W}_1}, \ldots, \mathbf{rep_n}_{\mathcal{R}_n}^{\mathcal{W}_n}$—we define a new type of repository denoted $\mathbf{REP} = [\mathbf{rep_1}_{\mathcal{R}_1}^{\mathcal{W}_1}, \ldots, \mathbf{rep_n}_{\mathcal{R}_n}^{\mathcal{W}_n}]$, which is essentially 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.

Modeling an Asynchronous Network. To model an asynchronous network we define a network converter Net (in Algorithm 3), which provides an interface for message delivery 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 $(\mathtt{id}, (\mathbf{rep_i}, m)) \in \textsc{Read}$ :
    if $\mathtt{id} \in \text{Received}[P]$ :
      $\text{list} \leftarrow \text{list} \cup (\mathtt{id}, (\mathbf{rep_i}, m))$
  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$(P \in \mathcal{P}^H, \mathtt{id})$
  $\text{Received}[P] \leftarrow \text{Received}[P] \cup \{\mathtt{id}\}$
```

3.1 MDRS-PKE Application Semantics: A Modular Approach to Capturing Composable Security Guarantees

The guarantees one expects from an MDRS-PKE scheme depend on the honesty of the judge J(-udy): if dishonest, we expect consistency and Off-The-Record; if honest, we additionally expect authenticity. (Authenticity is not possible when J is dishonest because she has access to honest senders' secret keys and so she can impersonate them.) Finally, we also expect confidentiality and anonymity for messages sent by honest senders to vectors of all-honest receivers.

Dishonest Judy. For each sender $A_i \in \mathcal{S}$ and each vector of receivers $\vec{V} \in \mathcal{R}^+$, the ideal system includes a repository

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

to which A_i and any dishonest party can write to, and from which dishonest parties and the ones in $\vec{V}$ can read. Consistency is captured because for each such repository $\langle A_i \to \vec{V} \rangle$, either there is a register with identifier $\mathtt{id}$—in which case *all honest receivers* $B_j \in \vec{V}$ get the *same unique tuple* $(\mathtt{id}, (\langle A_i \to \vec{V} \rangle, m))$ as part of the output of a READ operation—or there is not—in which case *no honest receiver* $B_j \in \vec{V}$ gets a *tuple with a matching identifier* $\mathtt{id}$ (as part of the output of a READ operation). To capture Off-The-Record, for each sender A_i and vector of designated receivers $\vec{V}$ we consider an additional repository to which forged messages are written to:

$$\langle [\text{Forge}] A_i \to \vec{V} \rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}.$$

The ideal system includes a repository consisting of all these (atomic) repositories put together, with the Net converter attached, i.e.[16]

$$\begin{bmatrix} \mathsf{Net} \cdot \left[\langle A_i \to \vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \\ \left[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix}. \tag{3.1}$$

Algorithm 4 Converter $\mathsf{ConfAnon}$.

```
▷ $(P \in \overline{\mathcal{P}^H})$-Read
  list ← ∅
  for $(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m)) \in$ Read :
    if $(A_i, \vec{V}) \in \mathcal{S}^H \times (\mathcal{R}^H)^+$ :
      list ← list ∪ $\{(\mathtt{id}, (|\vec{V}|, |m|))\}$
    else
      list ← list ∪ $\{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
  Output(list)
```

Algorithm 5 Converter Otr.

```
▷ $(P \in \overline{\mathcal{P}^H})$-Read
  list ← ∅
  for $(\mathtt{id}, (\mathbf{rep_i}, m)) \in$ Read :
    if $\mathbf{rep_i} = \langle[\mathrm{Forge}]A_i \to \vec{V}\rangle$ :
      list ← list ∪ $\{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
    else // $\mathbf{rep_i} = \langle A_i \to \vec{V}\rangle$.
      list ← list ∪ $\{(\mathtt{id}, (\mathbf{rep_i}, m))\}$
  Output(list)
```

The repositories in Equation 3.1 do not capture neither Off-The-Record, confidentiality nor anonymity guarantees, and we do not know how to model these only using the repository resources from before: regarding Off-The-Record and anonymity, Read operations leak the atomic repository from which each of the tuples output was read from, and so, for the case of OTR, a party can distinguish real messages—which would be paired with repositories labels of the form $\langle A_i \to \vec{V}\rangle$—from forged ones—which would be paired with labels of the form $\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle$; regarding confidentiality, either a party has read access to an atomic repository—in which case Read operations output all of the repository's messages in plain—or the party has no read access to the atomic repository—in which case the party does not learn anything about the messages written in that repository: not their length nor how many there are. We model Off-The-Record via a converter Otr that is attached to dishonest parties'

[16] Net need not be attached to $\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle$ since all readers are dishonest.

READ interfaces and limits their (reading) capabilities. Otr—formally defined in Algorithm 5—captures Off-The-Record because it ensures (dishonest) parties do not know whether they are reading real messages—written to $\langle A_i \to \vec{V} \rangle$—or forged ones—written to $\langle [\text{Forge}] A_i \to \vec{V} \rangle$. Confidentiality and anonymity are modeled similarly—by attaching a converter $\mathsf{ConfAnon}$ (defined in Algorithm 4) to the READ interfaces of dishonest parties that similarly limits their reading capabilities: if an honest sender $A_i \in \mathcal{S}^H$ sends a message m to a vector of receivers $\vec{V}$ all of whom are honest (i.e. $\vec{V} \in \mathcal{R}^{H^+}$), then $\mathsf{ConfAnon}$ only leaks $|\vec{V}|$ and $|m|$ to dishonest parties (instead of $\langle A_i \to \vec{V} \rangle$ and m). The ideal world system $\mathbf{S}$ is then

$$\mathbf{S} := \left(\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \; \mathsf{Otr}^{\overline{\mathcal{P}^H}} \right) \cdot \begin{bmatrix} \mathsf{Net} \cdot \left[\langle A_i \to \vec{V} \rangle^{\{A_i\} \cup \overline{\mathcal{P}^H}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{A_i \in \mathcal{S}, \vec{V} \in \mathcal{R}^+} \\ \left[\langle [\text{Forge}] A_i \to \vec{V} \rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}} \right]_{A_i \in \mathcal{S}, \vec{V} \in \mathcal{R}^+} \end{bmatrix}. \tag{3.2}$$

Honest Judy. To capture authenticity, the ideal system $\mathbf{T}$ includes (atomic) repositories

$$\left[\langle A_i \to \vec{V} \rangle^{\{A_i\}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{A_i \in \mathcal{S}^H, \vec{V} \in \mathcal{R}^+}$$

to which only the intended (honest) sender A_i can write. For dishonest senders $A_i \in \overline{\mathcal{S}^H}$ we only expect consistency, which can be captured as before; $\mathbf{T}$ then includes all such atomic repositories with converter Net attached:

$$\mathsf{Net} \cdot \begin{bmatrix} \left[\langle A_i \to \vec{V} \rangle^{\{A_i\}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{A_i \in \mathcal{S}^H, \vec{V} \in \mathcal{R}^+} \\ \left[\langle A_i \to \vec{V} \rangle^{\overline{\mathcal{P}^H}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{A_i \in \overline{\mathcal{S}^H}, \vec{V} \in \mathcal{R}^+} \end{bmatrix} \tag{3.3}$$

As one might note, and similarly to how we captured Off-The-Record, confidentiality and anonymity, we can alternatively capture authenticity by also attaching a converter to dishonest parties interfaces. Concretely, we attach converter $\perp$ to the WRITE sub-interfaces of dishonest parties' for which the sender A_i in the input label $\langle A_i \to \vec{V} \rangle$ is honest (i.e. $A_i \in \mathcal{S}^H$). Denoting these (sub-)interfaces by $\mathsf{Auth\text{-}Intf} := \overline{\mathcal{P}^H}\text{-WRITE}(\langle \mathcal{S}^H \to \mathcal{R}^+ \rangle, \cdot)$, this means

$$\perp^{\mathsf{Auth\text{-}Intf}} \cdot \left[\langle A_i \to \vec{V} \rangle^{\{A_i\} \cup \overline{\mathcal{P}^H}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{\substack{A_i \in \mathcal{S}^H \\ \vec{V} \in \mathcal{R}^+}} \equiv \left[\langle A_i \to \vec{V} \rangle^{\{A_i\}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{\substack{A_i \in \mathcal{S}^H \\ \vec{V} \in \mathcal{R}^+}}. \tag{3.4}$$

By capturing Off-The-Record, confidentiality and anonymity as before, the ideal system $\mathbf{T}$ is then defined as:

$$\mathbf{T} := \begin{pmatrix} \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \\ \cdot \, \mathsf{Otr}^{\overline{\mathcal{P}^H}} \end{pmatrix} \cdot \begin{bmatrix} \mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \left[\langle A_i \to \vec{V} \rangle^{\{A_i\} \cup \overline{\mathcal{P}^H}}_{\text{Set}(\vec{V}) \cup \overline{\mathcal{P}^H}} \right]_{\substack{A_i \in \mathcal{S} \\ \vec{V} \in \mathcal{R}^+}} \\ \left[\langle [\text{Forge}] A_i \to \vec{V} \rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}} \right]_{A_i \in \mathcal{S}, \vec{V} \in \mathcal{R}^+} \end{bmatrix}. \tag{3.5}$$

4 Chat Sessions

We denote the set of messaging parties by $\mathcal{M} = \{P_1, \ldots, P_n\}$ (being that each party $P_i \in \mathcal{M}$ can both send and receive messages) and assume $\mathcal{M}^H$ and $\overline{\mathcal{M}^H}$ are non-empty. Set $\mathcal{P}$ also includes the judge J, i.e. $\mathcal{P} = \mathcal{M} \cup \{J\}$.

4.1 Ideal Chat Sessions

The chat sessions functionality, denoted **ChatSessions**$[\mathfrak{P}]$, allows parties to perform $\textsc{Read}$ and $\textsc{Write}$ operations. When a party $P \in \mathcal{M}^H$ issues a $\textsc{Read}$ operation (which takes no input), **ChatSessions**$[\mathfrak{P}]$ outputs a set of pairs $(\texttt{sid}, \mathcal{G}^+)$, where $\texttt{sid}$ 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. $\textsc{Write}$ operations are uniquely identified by an $\texttt{id}$ and have an associated writer/sender S, vector of receivers $\vec{V}$, and message $m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks})$—a triple comprising an $\texttt{sid}$, a command $\texttt{cmd}$ and a set Acks of (prior) $\textsc{Write}$ operation identifiers to acknowledge. These operations take as input an $\texttt{sid}$, a vector of receivers $\vec{V}$, a command $\texttt{cmd}$ and a set of acknowledgements Acks, and output their own identifier $\texttt{id}$.

As one of the main goals of our abstraction is capturing consistency, our ideal functionality keeps track of a global directed graph (digraph) $\mathcal{G} = (V, E)$ for each existing chat session $\texttt{sid}$.[17] Each node $v \in V$ of this global graph is the identifier $\texttt{id}$ of a $\textsc{Write}$ operation, and for any node $v \in V$, we have that $(u, v) \in E$ if and only if the (message) triple $m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks})$ corresponding to ($\textsc{Write}$) v is such that $u \in \text{Acks}$. The elements $\mathcal{G}^+ = (V^+, E^+)$ output by $\textsc{Read}$ operations are of a different type than the global digraphs: on one hand, each $u \in V^+$ is of the form $(\texttt{id}, (\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks})))$—with $\texttt{id}$ being a $\textsc{Write}$ 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 $\texttt{sid}$, $\texttt{cmd}$ and Acks being, respectively, the session identifier, the command and the set of $\textsc{Write}$ operations acknowledged; on the other hand, the elements of E^+ (i.e. edges) are pairs $(\texttt{id}, \texttt{id}')$ of $\textsc{Write}$ operation identifiers. Since there are no two different tuples $u, v \in V^+$ with the same $\textsc{Write}$ operation identifier (i.e. $\forall u, v \in V^+, u \neq v \to u.\texttt{id} \neq v.\texttt{id}$), one can alternatively think of $\mathcal{G}^+$ as a triple $\mathcal{G}^+ = (\mathcal{G}' = (V', E'), \texttt{f})$ where $\mathcal{G}'$ is (informally) a subgraph of the global digraph from before and $\texttt{f}$ is a function—with domain V'—mapping each $\textsc{Write}$ operation $\texttt{id} \in V'$ to a tuple $(\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$. Throughout the rest of the paper we will call these the extended digraphs/graphs; we will 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.

The **ChatSessions**$[\mathfrak{P}]$ functionality is parameterized by a policy $\mathfrak{P}$ which defines two (deterministic) predicates: $\textsc{IsRoot}$ and $\textsc{IsValid}$. $\textsc{IsRoot}$ takes as input a session identifier $\texttt{sid}$, a sender S, a vector of receivers $\vec{V}$ and a command $\texttt{cmd}$ and outputs whether a corresponding node is a root node. $\textsc{IsValid}$ takes as

[17] Each $\mathcal{G} = (V, E)$ is not necessarily a digraph: there may be edges $(u, v) \in E$ for which $u \notin V$. For simplicity, however, we will still refer to $\mathcal{G}$ as being a digraph.

input a session identifier $\texttt{sid}$, an extended graph $\mathcal{G}^+ = (V^+, E^+)$—corresponding to a party's view of that session's graph—a sender S, a vector of receivers $\vec{V}$, a command $\texttt{cmd}$ and a set Acks of WRITE operation $\texttt{id}$s to acknowledge, and outputs whether $\texttt{cmd}$ is a valid command (with respect to the given input).

To simplify the description of our ideal messenger, we rely on a repository **REP** (see Algorithm 2) with a network converter Net (see Algorithm 3) attached. Every party in $\mathcal{M}$ can both send and receive messages; to capture this, each $P_i \in \mathcal{M}$ plays both the role of a sender $A_i \in \mathcal{S}$ and of a receiver $B_i \in \mathcal{R}$ (see Section 3). So **REP** is defined as

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

Authenticity, Off-The-Record, confidentiality and anonymity guarantees are defined similarly to Section 3; our security statements imply the additional guarantees if the assumed repositories provide them too (see Section 4.3).

4.1.1 Policy requirements.

One of the goals of our ideal abstraction is capturing consistency, meaning that it needs to ensure all parties have a consistent view of each session's chat graph—even if their views are partial due to undelivered messages. As one may note, this is not possible for every possible policy; we now move to define the requirements one needs to make on a policy parameterizing the chat sessions ideal functionality. Throughout the following, we consider a fixed policy $\mathfrak{P}$, and refer to the two predicates defined by $\mathfrak{P}$ simply as ISROOT and ISVALID, thus omitting $\mathfrak{P}$.

For some chat session identifier $\texttt{sid}$, command $\texttt{cmd}$, sender $S \in \mathcal{M}$ and vector of receivers $\vec{V} \in \mathcal{M}^+$, we call $(\texttt{sid}, S, \vec{V}, \texttt{cmd})$ a root if $\textsc{IsRoot}(\texttt{sid}, S, \vec{V}, \texttt{cmd}) = 1$. 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 1 (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* $\langle S \to \vec{V}\rangle$*, any triple* $(\texttt{sid}, \texttt{cmd}, \mathrm{Acks})$*, and any* $\texttt{id}$ *for a corresponding* WRITE *operation, if* $\textsc{IsValid}(\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \mathrm{Acks}) = 1$*, then* $\mathcal{G}^{+\prime} = (V^{+\prime}, E^{+\prime})$ *is* proper*, where* $V^{+\prime} := V^+ \cup \{(\texttt{id}, (\langle S \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \mathrm{Acks})))\}$*, and* $E^{+\prime} := E^+ \cup (\mathrm{Acks} \times \{\texttt{id}\})$*.*

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* $(\texttt{sid}, S, \vec{V}, \texttt{cmd})$ *and any finite set of* WRITE *operation identifiers* Acks*:* $\textsc{IsValid}(\texttt{sid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \mathrm{Acks}) = 1$*.*

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* $(\texttt{sid}, S, \vec{V}, \texttt{cmd})$ *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* $(\texttt{id}, \cdot) \in V^+$*.*

At a high level, the following requirement captures that the validity of a command 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*

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

4.1.2 The Definition. ChatSessions$[\mathfrak{P}]$ is formally defined in Algorithm 6; consistency is captured by the existence of a unique graph $\mathcal{G}$ (for each chat session) such that honest parties see subgraphs of $\mathcal{G}$ induced by what they received.

4.2 Constructing Chat Sessions

The assumed resource for our construction is a repository **REP** (matching the ideal world's **REP**) with the network converter Net attached: $(\mathsf{Net} \cdot \mathbf{REP})$; honest parties run converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ (Algorithm 7), which is parameterized by a policy $\mathfrak{P}$. The real world system is

$$\mathbf{R}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP}). \tag{4.2}$$

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 C.1 for the proof.)

4.3 Authenticity, Off-The-Record, Confidentiality and Anonymity

We model these guarantees like in Section 3 and prove that if the real world provides (any subset of) these guarantees, so does the ideal chat sessions system.

Authenticity. We attach converter $\perp^{\mathsf{Auth\text{-}Intf}}$ so dishonest parties cannot impersonate honest ones. The ideal system is then

$$\mathbf{AuthChatSessions}[\mathfrak{P}] \coloneqq \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{ChatSessions}[\mathfrak{P}]. \tag{4.3}$$

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**:

$$\mathbf{R}_{\mathbf{Auth}}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \perp^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}). \tag{4.4}$$

Corollary 1 follows from Theorem 1 (see Appendix Section C.2 for a proof).

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

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

▷ $(P \in \mathcal{M}^H)$-WRITE$(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$
  $\mathcal{G}^+ \leftarrow \mathsf{InducedPartyGraph}^+(\mathtt{sid}, P)$
Require: $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathcal{G}^+, P, \vec{V}, \mathtt{cmd}, \text{Acks})$
  $\mathtt{id} \leftarrow (\mathsf{Net}\cdot\mathbf{REP})$-WRITE$(\langle P \to \vec{V}\rangle, m := (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  Contents$[\mathtt{id}] \leftarrow (\langle P \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks}))$
  Sent$[P] \leftarrow$ Sent$[P] \cup \{\mathtt{id}\}$
  $\mathsf{AddToGraph}(\mathtt{sid}, \mathtt{id})$
  OUTPUT$(\mathtt{id})$

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

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

▷ DELIVER$(P, \mathtt{id})$
  $(\mathsf{Net}\cdot\mathbf{REP})$-DELIVER$(P, \mathtt{id})$

▷ $(P \in \overline{\mathcal{P}^H})$-READ
  OUTPUT$((\mathsf{Net}\cdot\mathbf{REP})$-READ$)$
```

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

◇ $\mathsf{Extended}(\mathcal{G} = (V, E))$  // Not part of interface.
  return $\mathcal{G}^+ := (\{(\mathtt{id}, \text{Contents}[\mathtt{id}]) \mid \mathtt{id} \in V\}, E)$

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

◇ $\mathsf{UpdatedGraph}(\mathcal{G}_0, \text{ToHandle})$  // Not part of interface.
  $i \leftarrow 0$, Handled $\leftarrow \emptyset$
  repeat
    $\mathcal{G}_{i+1} \leftarrow \mathcal{G}_i$
    for $\mathtt{id} \in$ ToHandle with $\mathtt{id} \notin$ Handled :
      $(\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \text{Acks})) \leftarrow \text{Contents}[\mathtt{id}]$
      if $\mathfrak{P}[\text{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_{i+1}), S, \vec{V}, \mathtt{cmd}, \text{Acks})$ :
        $\mathcal{G}_{i+1} \leftarrow (\mathcal{G}_{i+1}.V \cup \{\mathtt{id}\}, \mathcal{G}_{i+1}.E \cup (\text{Acks} \times \{\mathtt{id}\}))$
        Handled $\leftarrow$ Handled $\cup \{\mathtt{id}\}$
    $i \leftarrow i + 1$
  until $\mathcal{G}_i = \mathcal{G}_{i-1}$
  return $(\mathcal{G}_i, \text{Handled})$
```

Algorithm 7 Converter $\mathsf{ChatSessionsProt}[\mathfrak{P}]$.

```
◇ INITIALIZATION
  $\text{SessionGraphs}, \quad \text{Contents} \leftarrow \emptyset$

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

▷ $(P \in \mathcal{M}^H)$-WRITE$(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  ProcessReceived
  $\mathcal{G} := (V, E) \leftarrow \text{SessionGraphs}[\texttt{sid}]$ // If $\texttt{sid} \notin \text{SessionGraphs}$ then $\mathcal{G} = (\emptyset, \emptyset)$.
  Require: $\mathfrak{P}[\text{ISVALID}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}), P, \vec{V}, \texttt{cmd}, \text{Acks})$
  $\texttt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-WRITE}(\langle P \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Contents}[\texttt{id}] \leftarrow (\langle P \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{SessionGraphs}[\texttt{sid}] \leftarrow (V \cup \{\texttt{id}\}, E \cup (\text{Acks} \times \{\texttt{id}\}))$
  $\text{OUTPUT}(\texttt{id})$
-----------------------------------------------------------------
◇ ProcessReceived // Not part of interface.
  $\text{ToHandle} \leftarrow \emptyset$
  for $(\texttt{id}, (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))) \in (\mathsf{Net} \cdot \mathbf{REP})\text{-READ}$ with $\texttt{id} \notin \text{SessionGraphs}[\texttt{sid}].V$ :
    $\text{Contents}[\texttt{id}] \leftarrow (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
    $\text{ToHandle}[\texttt{sid}] \leftarrow \text{ToHandle}[\texttt{sid}] \cup \{\texttt{id}\}$
  for $\texttt{sid} \in \text{ToHandle}$ :
    $(\text{SessionGraphs}[\texttt{sid}], \cdot) \leftarrow \mathsf{UpdatedGraph}(\text{SessionGraphs}[\texttt{sid}], \text{ToHandle}[\texttt{sid}])$
```

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

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

Algorithm 8 The **FakeChatSessions** system to which fake messages (i.e. invisible to honest parties) are written. Below, **FAKE-REP** $:= [\langle[\text{Forge}]P \rightarrow \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+}$.

```
▷ $(P \in \mathcal{M})$-WRITE$(S, \texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  $\text{OUTPUT}\big(\mathbf{FAKE\text{-}REP}\text{-WRITE}(\langle[\text{Forge}]S \rightarrow \vec{V}\rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))\big)$

▷ $(P \in \overline{\mathcal{P}^H})$-READ
  $\text{OUTPUT}\big(\mathbf{FAKE\text{-}REP}\text{-READ}\big)$
```

Off-The-Record. We extend **AuthChatSessions**$[\mathfrak{P}]$ via parallel composition with **FakeChatSessions**—defined in Algorithm 8—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 5) 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

$$\textbf{OTR-ChatSessions}[\mathfrak{P}] \coloneqq \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \textbf{AuthChatSessions}[\mathfrak{P}] \\ \textbf{FakeChatSessions} \end{bmatrix}. \tag{4.5}$$

The assumed resources for this construction are similar to the ones for authenticity

Algorithm 9 Converter $\mathsf{ChatSessionsForgeProt}$.

```
▷ $(P \in \mathcal{M})$-WRITE$(S, \texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  OUTPUT$\Big(([\langle[\text{Forge}]P \to \vec{R}\rangle]_{P\in\mathcal{M},\vec{R}\in\mathcal{M}^+})$-WRITE$(\langle[\text{Forge}]S \to \vec{V}\rangle, m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks}))\Big)$
```

(Equation 4.4), but now also include repositories $\left[\langle[\text{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{P\in\mathcal{M},\vec{V}\in\mathcal{M}^+}$ 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 9) which allows writing fake messages. The real world resource is then

$$\mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}] \coloneqq \begin{pmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \\ \cdot\, \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \end{pmatrix} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \textbf{REP} \\ \left[\langle[\text{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\overline{\mathcal{P}^H}}\right]_{\substack{P\in\mathcal{M}\\ \vec{V}\in\mathcal{M}^+}} \end{bmatrix}. \tag{4.6}$$

Corollary 2 follows from Corollary 1 (see Appendix Section C.3 for a proof).

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

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

Confidentiality and Anonymity. We capture these guarantees via converter $\mathsf{ConfAnon}$ (Algorithm 4); consider any two resources $\mathbf{AR}[\mathfrak{P}]$ and $\mathbf{V}[\mathfrak{P}]$ such that

$$\mathbf{V}[\mathfrak{P}] \equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AR}[\mathfrak{P}] \tag{4.7}$$

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

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

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 give a full proof in Appendix Section C.4.

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 4.7 that have* $\overline{\mathcal{P}^H}$-Read *interfaces suitable for converter* $\mathsf{ConfAnon}$ *(Algorithm 4),*

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

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

5 UatChat: A Decentralized Messenger

In UatChat (Algorithms 12 and 13) there are two main types of operations: Read operations and command writing operations. Read operations behave similarly to chat sessions: they output the graphs of all chats a party is in. There are four interfaces for command writing: CreateChat, ProposeChange, Vote and Write. All these interfaces take as input a chat session identifier $\texttt{cid}$[18], and, in general, upon a query they issue a chat sessions Write operation and output whatever Write identifier is output. Slightly more specifically:

CreateChat: on input $(\texttt{cid}, \vec{G})$, where $\vec{G}$ defines the group member vector, issues a Write with command $(\mathsf{Create}, \vec{G})$ and acknowledgements $\text{Acks} = \emptyset$;

ProposeChange: on input $(\texttt{cid}, \texttt{vid}, \mathsf{change}, P)$, where $\texttt{vid}$ specifies the *version* of the chat to which the change is to be made, and change and P' specify the actual change—if $\mathsf{change} = \mathsf{Add}$ then P' is being added and otherwise, if $\mathsf{change} = \mathsf{Rm}$ then 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 $\texttt{vid}$;[19]

Vote: on input $(\texttt{cid}, \texttt{vid})$, where $\texttt{vid}$ is a proposed chat version, issues a Write with command $(\texttt{vid}, \mathsf{Vote})$ and acknowledgements $\text{Acks} = \{\texttt{vid}\}$; and

Write: on input $(\texttt{cid}, \texttt{vid}, m, \texttt{ReplyTo})$—where $\texttt{vid}$ identifies the chat version to which the message m is intended and $\texttt{ReplyTo}$ is the set of prior commands the party wants to explicitly acknowledge—issues a Write with command $(\texttt{vid}, \mathsf{Msg}, m, \texttt{ReplyTo})$ and a set of acknowledgements that includes each command in $\texttt{ReplyTo}$ (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 10—is denoted $\mathfrak{U}$.

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

[18] These identifiers are just chat sessions' identifiers and only serve to identify a particular chat session.

[19] Adding $\vec{G}$ to the command allows the joining party to learn the current group roster and each group member to confirm this roster.

Definition 2. *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 $\texttt{cid}$, 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 $(\texttt{vid}, \mathsf{Vote})$ is valid if $\texttt{vid}$ 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. $\mathrm{Acks} = \{\texttt{vid}\}$. Finally, a Write command $(\texttt{vid}, (\mathsf{Msg}, \cdot, \texttt{ReplyTo}))$ is valid if: 1. $\texttt{vid}$ is the identifier of a root; 2. every node in $\texttt{ReplyTo}$ is being acknowledged (i.e. $\texttt{ReplyTo} \subseteq \mathrm{Acks}$); 3. every node in Acks is in the subgraph sourced by $\texttt{vid}$; and 4. if node corresponding to $\texttt{vid}$ 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 10).

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 $(\texttt{cid}, S, \vec{V}, \texttt{cmd})$ is a root, then $\mathfrak{U}[\text{ISVALID}](\texttt{cid}, \mathcal{G}^+, S, \vec{V}, \texttt{cmd}, \mathrm{Acks}) = 1$. To exemplify, we add such extra requirements to our messenger (see Algorithm 11). 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.

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

- if $\texttt{cmd} = (\mathsf{Create}, \vec{G})$, then Acks must be the empty set;

– if $\mathtt{cmd} = (\mathtt{vid}, \mathsf{change} \in \{\mathsf{Add}, \mathsf{Rm}\}, \vec{G}, P)$, 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 $\vec{G}_{\mathtt{vid}}$ corresponding to vid must be consistent with $\vec{G}$ (see Algorithm 11).

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 $(\mathtt{cid}, \mathsf{Add}, \vec{G}, P)$: 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 an unanimous agreement (to add P to chat cid). 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. By analyzing Algorithms 6 and 12 it follows that dishonest parties' capabilities are exactly the same in **ChatSessions**$[\mathfrak{U}]$ and **UatChat**, and the same holds for interface Deliver. This means that one can equivalently define the ideal **UatChat** system by defining a converter UatChatProt run by each honest party and attaching it to **ChatSessions**$[\mathfrak{U}]$:[20] $\mathsf{UatChatProt}^{\mathcal{M}^H} \cdot \mathbf{ChatSessions}[\mathfrak{U}] \equiv \mathbf{UatChat}$.

5.2.2 Capturing additional guarantees. One can capture authenticity, confidentiality, anonymity and Off-The-Record analogously to how we captured these guarantees for **ChatSessions**$[\mathfrak{P}]$ (see Section 4.3). Corollaries analogous to Corollaries 1, 2 and 3 also hold for **UatChat**. Regarding Off-The-Record, in Algorithm 14 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 OtrCS attached.

6 MDRS-PKE: New Game-Based Notions and Their Application Semantics

In this section we 1. introduce new game-based security notions for MDRS-PKE schemes; 2. prove that our new notions—together with the ones from [19]—do

[20] For completeness, we define converter UatChatProt in the Appendix, Algorithm 29.

imply the composable notions from Section 3; and 3. prove the security of Maurer et al.'s MDRS-PKE construction [35] with respect to the new notions.

An MDRS-PKE scheme is a 6-tuple of PPTs $\Pi = (S, G_S, G_R, E, D, \mathit{Forge})$, where, for security parameter k: 1. $S(1^k)$: generates public parameters $\mathtt{pp}$; 2. $G_S(\mathtt{pp})$: generates a sender key-pair $(\mathtt{spk}, \mathtt{ssk})$; 3. $G_R(\mathtt{pp})$: generates a receiver key-pair $(\mathtt{rpk}, \mathtt{rsk})$; 4. $E(\mathtt{pp}, \mathtt{ssk}, \vec{v}, m)$: generates a ciphertext c—$\vec{v}$ being the vector of public receiver keys of the intended receivers; 5. $D(\mathtt{rsk}_j, c)$: outputs a triple $(\mathtt{spk}, \vec{v}, m)$—with $\vec{v}$ again being a vector of receiver public keys—or $\perp$ if decryption fails; 6. $\mathit{Forge}(\mathtt{pp}, \mathtt{ssk}, \vec{v}, m, \vec{s})$: generates a ciphertext c—$\vec{v}$ being the vector of public receiver keys of the intended receivers, and $\vec{s}$ a vector of receiver secret keys such that $|\vec{v}| = |\vec{s}|$ and where, for each $i \in \{1, \ldots, |\vec{v}|\}$, either s_i is the secret key corresponding to v_i, or it is $\perp$.

6.1 Assumed Resources and Protocol

Assumed Resources. Parties have access to an asynchronous and anonymous insecure repository $\mathsf{Net}\cdot\mathbf{INS}$—to which everyone can write to and read from—and to a *Key Generation Authority* (**KGA**) resource [34], which generates and stores parties' key pairs (see Algorithm 15). [21] We consider the setting from [19]—where judge Judy has access to the secret keys of honest senders—meaning the **KGA** gives Judy access to the secret keys of honest senders.

The **KGA** resource—which is implicitly parameterized by a security parameter k—first runs setup algorithm S and then samples an MDRS-PKE receiver public key—$\mathtt{rpk}_{\mathtt{pp}}$—which it attaches to the public parameters.[22] Next, it generates key-pairs for all senders and receivers—using G_S and G_R, respectively. Every honest party can query their own key-pair, the public parameters and everyone's public keys at their own interface. Dishonest parties can additionally obtain the key-pairs of any dishonest senders or receivers; finally, the J can additionally obtain the secret keys of honest senders. [23]

Protocol. Each honest sender and each honest receiver locally runs a converter Snd and Rcv, respectively (see Algorithm 16); these converters connect to both the **KGA** and to $\mathsf{Net}\cdot\mathbf{INS}$ and provide an outer interface that is identical to the interface of a repository for a party who is a writer and a reader, respectively. A party A_i's Snd converter provides a procedure WRITE which takes as input a label $\langle A_i \to \vec{V}\rangle$ (defining the vector of receivers $\vec{V} = (V_1, \ldots, V_{|\vec{V}|})$) and a message m;

[21] The **KGA** guarantees dishonest receivers can actually access their own secret keys, which allows them to come up with forged ciphertexts; in turn, being able to come up with forged ciphertexts (that look like real ones) is necessary for the Off-The-Record guarantee [23, 24, 34].

[22] The additional public key allows for simpler reductions (one can rely on the corresponding secret key for decryption) and eliminates the need for the MDRS-PKE scheme to satisfy a robustness type of notion.

[23] Resource **KGA** supports an additional helper operation GETLABEL which, given an sender's public key and a vector of receiver public keys, outputs the unique corresponding label $\langle A_i \to \vec{V}\rangle$, or $\perp$ if the label does not exist or is not unique.

upon such input, Snd encrypts the input message, writes the resulting ciphertext to $\mathsf{Net}\cdot\mathbf{INS}$, and outputs the `id` output by writing to $\mathsf{Net}\cdot\mathbf{INS}$. A party B_j's Rcv converter reads all (received) ciphertexts from $\mathsf{Net}\cdot\mathbf{INS}$—filtering out duplicated ones—and tries decrypting each of them. For each ciphertexts that decrypt correctly, Rcv uses the $\mathbf{KGA}$'s GetLabel operation to obtain a label—i.e. looks up the sender/vector of receivers with public keys matching the ones obtained from decryption—and then outputs a set of triples, each triple corresponding to a ciphertext that decrypted correctly and for which the GetLabel operation returned a valid label (i.e. not $\perp$).

In addition to converters Snd and Rcv, every party in $\mathcal{F}$—*crucially including dishonest ones*—runs a converter Forge (see Algorithm 16) that allows to forge messages, mimicking senders' Write operations towards dishonest parties. Each such Forge converter connects to the $\mathbf{KGA}$ and $\mathbf{INS}$ resources, but is not given access to the secret keys of any sender $A_i \in \mathcal{S}$. The goal of having dishonest parties run converter Forge is to capture their ability of forging real-looking ciphertexts; however, since these parties are dishonest, we still want them to have the same access to the $\mathbf{KGA}$ and $\mathbf{INS}$ resources as if they were not running Forge, i.e. running Forge cannot restrict their capabilities. This can be formally modeled by extending the $\mathbf{KGA}$ resource with additional interfaces that allows a party's Forge converter to obtain the necessary public and secret keys (see Algorithm 17), and extending the $\mathbf{INS}$ repository to provide these converters write access.[24]

Real World System. For dishonest J, the real world system $\mathbf{R}$ is given by $\mathsf{Snd}^{\mathcal{S}^H}\,\mathsf{Rcv}^{\mathcal{R}^H}\,\mathsf{Forge}^{(\mathcal{F}\times\{\mathsf{Forge}\})}\,[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}]$, and the one for the case where J is honest is the same but has an additional converter $\perp$ that covers all of J's interfaces and provides no outside interface.

6.2 New Security Notions

We now introduce Forgery Invalidity and MDRS-PKE Public-Key Collision Resistance: two new notions that we use for our composable treatment of MDRS-PKE schemes. Regarding Forgery Invalidity, note that the existing security notions for MDRS-PKE schemes do not give any guarantee on whether a ciphertext forgery on messages picked by an adversary who can access the secret key of the sender may not decrypt successfully by honest receivers. In particular, this is not captured by Unforgeability because the adversary could choose messages to be forged depending on the sender's secret key—which it does not have access to in the Unforgeability game. Furthermore, we do not know how to prove the game-based notions capture the MDRS-PKE application semantics (in the setting where secret keys of honest senders leak [19]) without requiring Forgery Invalidity from the underlying MDRS-PKE.

Let $\Pi = (S, G_S, G_R, E, D, \mathit{Forge})$ be an MDRS-PKE scheme with message space $\mathcal{M}$. The game-based notion ahead has an implicitly defined security parameter k and provide adversaries with access to the following oracles:

[24] This is achieved by defining $\mathbf{INS}$ as $\mathbf{INS}_{\mathcal{P}}^{\mathcal{P}\cup(\mathcal{F}\times\{\mathsf{Forge}\})}$.

$\mathcal{O}_{PP}$: On the first query, compute $\mathtt{pp} \leftarrow S(1^k)$; output $\mathtt{pp}$;

$\mathcal{O}_{SK}(A_i)$: On the first query on input A_i, compute and store $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow G_S(\mathtt{pp})$; output $(\mathtt{spk}_i, \mathtt{ssk}_i)$;

$\mathcal{O}_{RK}(B_j)$: Analogous to the Sender Key-Pair Oracle;

$\mathcal{O}_{SPK}(A_i)$: $(\mathtt{spk}_i, \cdot) \leftarrow \mathcal{O}_{SK}(A_i)$; output $\mathtt{spk}_i$;

$\mathcal{O}_{RPK}(B_j)$: Analogous to the Sender Public-Key Oracle;

$\mathcal{O}_E(A_i, \vec{V}, m)$: 1. $(\cdot, \mathtt{ssk}_i) \leftarrow \mathcal{O}_{SK}(A_i)$; 2. $\vec{v} \leftarrow (\mathcal{O}_{RPK}(V_1), \ldots, \mathcal{O}_{RPK}(V_{|\vec{V}|}))$; 3. output $c \leftarrow E_{\mathtt{pp}}(\mathtt{ssk}_i, \vec{v}, m)$;

$\mathcal{O}_D(B_j, c)$: 1. $(\cdot, \mathtt{rsk}_j) \leftarrow \mathcal{O}_{RK}(B_j)$; 2. $(\mathtt{spk}_i, \vec{v}, m) \leftarrow D_{\mathtt{pp}}(\mathtt{rsk}_j, c)$; 3. if, for each party A_i previously input to either $\mathcal{O}_{SK}$, $\mathcal{O}_{SPK}$ or $\mathcal{O}_E$, $\mathtt{spk}_i \neq \mathcal{O}_{SPK}(A_i)$, then output $\perp$; 4. if, for some $l \in \{1, \ldots, |\vec{V}|\}$, there is no party B_j that was previously input to either $\mathcal{O}_{RK}$, $\mathcal{O}_{RPK}$, $\mathcal{O}_E$ or $\mathcal{O}_D$ such that $v_l = \mathcal{O}_{RPK}(V_l)$, then output $\perp$; 5. output $(\mathtt{spk}, \vec{v}, m)$.

Game $\mathbf{G}^{\textsf{Forge-Invalid}}$ defined by the Forgery Invalidity notion provides adversaries with access to the oracles from above plus oracle $\mathcal{O}_{Forge}$ below:

$\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C} \subseteq \mathbf{Set}(\vec{V}))$: 1. $\mathtt{spk}_i \leftarrow \mathcal{O}_{SPK}(A_i)$; 2. for $i = 1, \ldots, |\vec{V}|$, let $(v_i, s_i) = \begin{cases} \mathcal{O}_{RK}(V_i) & \text{if } V_i \in \mathcal{C} \\ (\mathcal{O}_{RPK}(V_i), \perp) & \text{otherwise,} \end{cases}$; 3. output $\Pi.Forge_{\mathtt{pp}}(\mathtt{spk}_i, \vec{v}, m, \vec{s})$, where $\vec{v} = (v_1, \ldots, v_{|\vec{V}|})$ and $\vec{s} = (s_1, \ldots, s_{|\vec{V}|})$.

Definition 3 (Forgery Invalidity). *Game $\mathbf{G}^{\textsf{Forge-Invalid}}$ gives an adversary access to oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{RK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{RPK}$, $\mathcal{O}_E$, $\mathcal{O}_D$ and $\mathcal{O}_{Forge}$. An adversary $\mathbf{A}$ wins if there is a query $\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C})$ and a later query $\mathcal{O}_D(B_j, c)$ such that: 1. $B_j \in \vec{V}$; 2. $B_j \notin \mathcal{C}$; 3. the input c to $\mathcal{O}_D$ is the output of $\mathcal{O}_{Forge}$; and 4. the output of $\mathcal{O}_D$ is not $\perp$. The advantage of $\mathbf{A}$ is denoted $Adv^{\textsf{Forge-Invalid}}(\mathbf{A})$.*

Definition 4 (Public-Key Collision Resistance). MDRS-PKE $\Pi = (S, G_S, G_R, E, D, Forge)$ *is (n, ℓ)-Party ε-Public-Key Collision Resistant if*

$$\Pr\left[\begin{array}{c|c} \big|\{\mathtt{spk}_1, \ldots, \mathtt{spk}_n, & \mathtt{pp} \leftarrow S(1^k) \\ \mathtt{rpk}_1, \ldots, \mathtt{rpk}_\ell\}\big| & (\mathtt{spk}_1, \mathtt{ssk}_1) \leftarrow \Pi.G_{S_{\mathtt{pp}}},\ (\mathtt{rpk}_1, \mathtt{rsk}_1) \leftarrow G_{R_{\mathtt{pp}}} \\ < n + \ell & \ldots \qquad\qquad \ldots \\ & (\mathtt{spk}_n, \mathtt{ssk}_n) \leftarrow G_{S_{\mathtt{pp}}},\ (\mathtt{rpk}_\ell, \mathtt{rsk}_\ell) \leftarrow G_{R_{\mathtt{pp}}} \end{array}\right] \leq \varepsilon.$$

6.3 Application Semantics of Game-Based Notions

The informal theorem below summarizes our claims regarding the application semantics of the MDRS-PKE security notions we introduced in this section. For the formal theorem statements and respective full proofs, see Section H.

Theorem 3 (Informal). *Suppose an* MDRS-PKE *scheme Π is correct, consistent, replay-unforgeable,* $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ *secure, Off-The-Record, satisfies forgery invalidity and is public-key collision resistant. If Π is used as the* MDRS-PKE *scheme underlying the real world systems defined in Section 6.1 then there are poly-time simulators $\mathsf{sim}_{\mathbf{S}}$ and $\mathsf{sim}_{\mathbf{T}}$ and there are negligible functions $\varepsilon_{\mathbf{S}}$ and $\varepsilon_{\mathbf{T}}$ such that, for any (suitable) poly-time distinguishers $\mathbf{D_S}, \mathbf{D_T}$, the two statements below hold:*
$\Delta^{\mathbf{D_S}}\left(\mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}],\ \mathsf{sim}_{\mathbf{S}}\cdot\mathbf{S}\right) \leq \varepsilon_{\mathbf{S}}$ *(Theorem 12);*
$\Delta^{\mathbf{D_T}}\left(\mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}\bot^{J}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}],\ \mathsf{sim}_{\mathbf{T}}\cdot\mathbf{T}\right) \leq \varepsilon_{\mathbf{T}}$ *(Theorem 11).*

6.4 Composable Security of Maurer et al.'s MDRS-PKE

Finally, we prove the security Maurer et al.'s MDRS-PKE construction with respect to the new notions introduced above. Its building blocks are an MDVS scheme Π_{MDVS}, a PKEBC scheme Π_{PKEBC} and a DSS Π_{DSS} [35,36]; the informal theorem below gives an overview of our results regarding the additional guarantees given by Maurer et al.'s MDRS-PKE construction $\Pi_{\text{MDRS-PKE}}$ [35,36].[25]

Theorem 4 (Informal). *If Π_{PKEBC} is tightly correct, robust, consistent and* $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ *secure under adaptive corruptions, Π_{MDVS} is tightly consistent, unforgeable, message-bound validity and forgery invalidity secure and is public-key collision resistant (all under adaptive corruptions), and Π_{DSS} is tightly* 1-sEUF-CMA *secure then $\Pi_{\text{MDRS-PKE}}$ is tightly: 1. consistent under adaptive corruptions ([35, Theorem 7], Theorem 7); 2. (replay attack) unforgeable under adaptive corruptions (Theorem 8); 3.* $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ *secure under adaptive corruptions ([19, Theorem 13], Theorem 9[26]); 4. forgery invalidity secure (Theorem 10); and 5. public-key collision resistant (Corollary 6).*

Finally, note that, as explained in [19] there are suitable tightly secure structure preserving instantiations of each of our construction's building blocks: on one hand it follows from Theorem 5 that we can take Chakraborty et al.'s MDVS construction [19],[27] on the other hand we can take Chakraborty et al.'s PKEBC construction [19] and any (One-Time) sEUF-CMA secure DSS.

[25] As one may note, in the theorem statement we require Π_{MDVS} to provide Forgery Invalidity and public-key collision resistance, which we did not define for MDVS schemes. In Appendix, Section F, we define these MDVS notions, which are analogous to the MDRS-PKE notions defined above, and prove that Chakraborty et al.'s construction [19] does provide them.
[26] Our proof is essentially the same as [19, Proof of Theorem 13].
[27] We note that all our reductions are tight.

Algorithm 10 Unanimous policy $\mathfrak{U}$; Below, $\mathsf{Sourced}$ is as in Definition 2.

```
◇ IsRoot($\texttt{cid}, S, \vec{V}, \texttt{cmd}$)
  $(\text{Voters}, \cdot, \vec{G}, \cdot) \leftarrow \mathsf{RootCmdInfo}(\texttt{cmd})$
  if $(\text{Voters}, \cdot, \vec{G}, \cdot) = \bot$ :
    return 0
  return $\big(|\vec{G}| = |\text{Set}(\vec{G})|\big) \wedge \big(S \in \text{Voters}\big) \wedge \big(\vec{V} = \vec{G}\big)$

◇ IsValid($\texttt{cid}, \mathcal{G}^+ = (V^+, E^+), S, \vec{V}, \texttt{cmd}, \text{Acks}$)
  if IsRoot($\texttt{cid}, S, \vec{V}, \texttt{cmd}$) $= 1$ :  // 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
```

```
◇ 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

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

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

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

◇ RootCmdInfo($\texttt{cmd}$)
  if $\texttt{cmd} = (\mathsf{Create}, \vec{G})$ :
    return $(\text{Set}(\vec{G}), 0, \vec{G}, \vec{G})$
  if $\texttt{cmd} = (\cdot, \mathsf{Add}, \vec{G}, P)$ :
    $\vec{G}' \leftarrow \vec{G} \mid\mid P$
    return $(\text{Set}(\vec{G}), 1, \vec{G}', \vec{G}')$
  if $\texttt{cmd} = (\cdot, \mathsf{Rm}, \vec{G}, P)$ :
    $\vec{G}' \leftarrow \vec{G} \mid\mid P$
    return $(\text{Set}(\vec{G}), 1, \vec{G}', \vec{G})$
  return $\bot$
```

Algorithm 11 Additional root requirements. Below, Sourced is as in Definition 2.

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

Algorithm 12 The ideal **UatChat** application. The description below relies on a system **ChatSessions**$[\mathfrak{U}]$ (see Algorithm 6).

```
    ▷ $(P \in \mathcal{M}^H)$-CreateChat($\texttt{cid}, \vec{G} \in \mathcal{M}^+$)
Require: $\texttt{cid} \notin$ UatChat-Read
    $(\vec{V}, \texttt{cmd}, \text{Acks}) \leftarrow (\vec{G}, (\mathsf{Create}, \vec{G}), \emptyset)$
Require: IsRoot-Ext($\texttt{cid}, (\emptyset, \emptyset), P, \vec{V}, \texttt{cmd}, \text{Acks}) = 1$
    Output(ChatSessions$[\mathfrak{U}]$-Write($\texttt{cid}, \texttt{cmd}, \vec{V}, \text{Acks}$))

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

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

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

    ▷ $(P \in \mathcal{M}^H)$-Read
    ChatGraphs $\leftarrow \emptyset$
    for $(\texttt{cid}, \mathcal{G}^+) \in$ ChatSessions$[\mathfrak{U}]$-Read with $\mathsf{VisibleGraph}(\texttt{cid}, \mathcal{G}^+, P) \neq (\emptyset, \emptyset)$ :
        ChatGraphs $\leftarrow$ ChatGraphs $\cup \{(\texttt{cid}, \mathsf{VisibleGraph}(\texttt{cid}, \mathcal{G}^+, P))\}$
    Output(ChatGraphs)

    ▷ $(P \in \overline{\mathcal{P}^H})$-Write($\langle S \rightarrow \vec{V} \rangle, m := (\texttt{cid}, \texttt{cmd}, \text{Acks})$) // $S \in \overline{\mathcal{M}^H}$
    Output(ChatSessions$[\mathfrak{U}]$-Write($\langle S \rightarrow \vec{V} \rangle, m$))

    ▷ $(P \in \overline{\mathcal{P}^H})$-Read
    Output(ChatSessions$[\mathfrak{U}]$-Read)

    ▷ Deliver($P, \texttt{id}$)
    ChatSessions$[\mathfrak{U}]$-Deliver($P, \texttt{id}$)

    ◇ BasicRequirements($\texttt{cid}, \texttt{vid}, P$)
Require: $\texttt{cid} \in$ ChatSessions$[\mathfrak{U}]$-Read
    $\mathcal{G}^+_{\text{vis}} := (V^+_{\text{vis}}, E^+_{\text{vis}}) \leftarrow \mathsf{VisibleGraph}(\texttt{cid}, \text{ChatSessions}[\mathfrak{U}]\text{-Read}[\texttt{cid}], P)$
Require: $(\texttt{vid} \in V^+_{\text{vis}}) \wedge (\mathsf{NodeIsRoot}(\texttt{vid}, V^+_{\text{vis}}) = 1)$
```

Algorithm 13 Helper functions from **UatChat**'s description. Below, Sourced is as in Definition 2.

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

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

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

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

Algorithm 14 System **FakeUatChat**.

```
▷ $(P \in \mathcal{M})$-FakeCreateChat$(S, \texttt{cid}, \vec{G} \in \mathcal{M}^{+})$
$\big(\vec{V}, \texttt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}, (\textsf{Create}, \vec{G}), \emptyset\big)$
Output(**FakeChatSessions**-Write$(S, \texttt{cid}, \texttt{cmd}, \vec{V}, \text{Acks})$)

▷ $(P \in \mathcal{M})$-FakeProposeChange$(S, \texttt{cid}, \texttt{vid}, \textsf{change} \in \{\textsf{Add}, \textsf{Rm}\}, P' \in \mathcal{M})$
$(\cdot, \vec{G}, \mathcal{G}^{+}_{\text{src-vis}} := (V^{+}_{\text{src-vis}}, E^{+}_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \textsf{HelperFunction}(P, \texttt{cid}, \texttt{vid})$
$\vec{G}' \leftarrow (\vec{G} \mid\mid P')$
$\text{LeafAcks} \leftarrow \{\texttt{id} \mid (\exists(\texttt{id}, (\cdot, (\cdot, (\texttt{vid}, \cdot), \cdot))) \in V^{+}_{\text{src-vis}}) \wedge (\nexists(\texttt{id}, \cdot) \in E^{+}_{\text{src-vis}})\}$
$\big(\vec{V}, \texttt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}', (\texttt{vid}, \textsf{change}, \vec{G}, P'), \text{VoteAcks} \cup \text{LeafAcks}\big)$
Output(**FakeChatSessions**-Write$(S, \texttt{cid}, \texttt{cmd}, \vec{V}, \text{Acks})$)

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

▷ $(P \in \mathcal{M})$-FakeWrite$(S, \texttt{cid}, \texttt{vid}, m, \texttt{ReplyTo})$
$(\cdot, \vec{G}, \mathcal{G}^{+}_{\text{src-vis}} := (V^{+}_{\text{src-vis}}, E^{+}_{\text{src-vis}}), \cdot, \text{VoteAcks}) \leftarrow \textsf{HelperFunction}(P, \texttt{cid}, \texttt{vid})$
$\big(\vec{V}, \texttt{cmd}, \text{Acks}\big) \leftarrow \big(\vec{G}, (\texttt{vid}, \textsf{Msg}, m, \texttt{ReplyTo}), \text{VoteAcks} \cup \texttt{ReplyTo}\big)$
Output(**FakeChatSessions**-Write$(S, \texttt{cid}, \texttt{cmd}, \vec{V}, \text{Acks})$)
```

Algorithm 15 The **KGA** resource for MDRS-PKE $\Pi = (S, G_S, G_R, E, D, \mathit{Forge})$.

```
◇ INITIALIZATION
  $\mathtt{pp} \leftarrow \Pi.S(1^k)$
  $(\mathtt{rpk}_{\mathtt{pp}}, \cdot) \leftarrow \Pi.G_R(\mathtt{pp})$
  for $A_i \in \mathcal{S}$: $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow \Pi.G_S(\mathtt{pp})$
  for $B_j \in \mathcal{R}$: $(\mathtt{rpk}_j, \mathtt{rsk}_j) \leftarrow \Pi.G_R(\mathtt{pp})$

▷ $(P \in \mathcal{P})$-PUBLICPARAMETERS
  OUTPUT$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}})$

▷ $(A_i \in \mathcal{S}^H)$-SENDERKEYPAIR
  OUTPUT$(\mathtt{spk}_i, \mathtt{ssk}_i)$

▷ $(J)$-SENDERKEYPAIR$(A_i \in \mathcal{S}^H)$
  OUTPUT$(\mathtt{spk}_i, \mathtt{ssk}_i)$

▷ $(P \in \overline{\mathcal{P}^H})$-SENDERKEYPAIR$(A_i \in \overline{\mathcal{S}^H})$
  OUTPUT$(\mathtt{spk}_i, \mathtt{ssk}_i)$

▷ $(P \in \mathcal{P})$-SENDERPUBLICKEY$(A_i \in \mathcal{S})$
  OUTPUT$(\mathtt{spk}_i)$

▷ $(B_j \in \mathcal{R}^H)$-RECEIVERKEYPAIR
  OUTPUT$(\mathtt{rpk}_j, \mathtt{rsk}_j)$

▷ $(P \in \overline{\mathcal{P}^H})$-RECEIVERKEYPAIR$(B_j \in \overline{\mathcal{R}^H})$
  OUTPUT$(\mathtt{rpk}_j, \mathtt{rsk}_j)$

▷ $(P \in \mathcal{P})$-RECEIVERPUBLICKEY$(B_j \in \mathcal{R})$
  OUTPUT$(\mathtt{rpk}_j)$

▷ $(B_j \in \mathcal{R}^H)$-GETLABEL$(\mathtt{spk}, \vec{v}')$
  $\mathcal{S}_{\mathsf{spk}} := \{A_i \mid \mathtt{spk} = \mathtt{spk}_i\}$
  if $|\mathcal{S}_{\mathsf{spk}}| \neq 1 \lor {v_1}' \neq \mathtt{rpk}_{\mathtt{pp}}$ :
    OUTPUT$(\bot)$
  for $l \in \{2, \ldots, |\vec{v}'|\}$ :
    $\mathcal{R}_{{v_l}'} := \{B_k \mid {v_l}' = \mathtt{rpk}_k\}$
    if $|\mathcal{R}_{{v_l}'}| \neq 1$ :
      OUTPUT$(\bot)$
    else
      Let $B_k$ be the element of $\mathcal{R}_{{v_l}'}$
      $V_{l-1} = B_k$
  Let $A_i$ be the element of $\mathcal{S}_{\mathsf{spk}}$
  Let $\vec{V} := (V_1, \ldots, V_{|\vec{v}'|-1})$
  OUTPUT$(\langle A_i \to \vec{V} \rangle)$
```

Algorithm 16 Converters Snd, Rcv and Forge.

```
▷ $(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$  // Conv. Snd
$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}}) \leftarrow$PublicParameters
$(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow$SenderKeyPair
for $l \in \{1, \ldots, |\vec{V}|\}$ :
    $\mathtt{rpk}_l \leftarrow$ReceiverPublicKey$(V_l)$
$\vec{v}' := (\mathtt{rpk}_{\mathtt{pp}}, \mathtt{rpk}_1, \ldots, \mathtt{rpk}_{|\vec{V}|})$
$c \leftarrow \Pi.E_{\mathtt{pp}}(\mathtt{ssk}_i, \vec{v}', m)$
Output(Write$(c)$)

▷ $(B_j \in \mathcal{R}^H)$-Read  // Conv. Rcv
$(\mathtt{rpk}_j, \mathtt{rsk}_j) \leftarrow$ReceiverKeyPair
$(\mathtt{pp}, \cdot) \leftarrow$PublicParameters
list,  ctxtSet $\leftarrow \emptyset$
for $(\mathtt{id}, c) \in$ Read with $c \notin$ ctxtSet :
    ctxtSet $\leftarrow$ ctxtSet $\cup \{c\}$
    $(\mathtt{spk}_i, \vec{v}', m) \leftarrow \Pi.D_{\mathtt{pp}}(\mathtt{rsk}_j, c)$
    if $(\mathtt{spk}_i, \vec{v}', m) \neq \bot$ :
        $\langle A_i \to \vec{V}\rangle \leftarrow$GetLabel$(\mathtt{spk}_i, \vec{v}')$
        if $\langle A_i \to \vec{V}\rangle \neq \bot$ :
            list $\leftarrow$ list $\cup \{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
Output(list)

▷ $(P \in \mathcal{F})$-Write$(\langle[\text{Forge}]A_i \to \vec{V}\rangle, m)$  // Conv. Forge
$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}}) \leftarrow$PublicParameters
if $\vec{V} \in \mathcal{R}^{H+}$ :
    $\mathtt{spk}_1 \leftarrow$SenderPublicKey$(A_1)$
    $c \leftarrow \Pi.Forge_{\mathtt{pp}}(\mathtt{spk}_1, {\mathtt{rpk}_{\mathtt{pp}}}^{|\vec{V}|+1}, 0^{|m|}, \bot^{|\vec{V}|+1})$
else
    $\mathtt{spk}_i \leftarrow$SenderPublicKey$(A_i)$
    for $l \in \{1, \ldots, |\vec{V}|\}$ :
        if $V_l \in \mathcal{R}^H$ :
            $(\mathtt{rpk}_l, \mathtt{rsk}_l) \leftarrow$ (ReceiverPublicKey$(V_l)$, $\bot$)
        else // $V_l \in \overline{\mathcal{R}^H}$
            $(\mathtt{rpk}_l, \mathtt{rsk}_l) \leftarrow$ReceiverKeyPair$(V_l)$
    $(\vec{v}', \vec{s}) := \big((\mathtt{rpk}_{\mathtt{pp}}, \mathtt{rpk}_1, \ldots, \mathtt{rpk}_{|\vec{V}|}),\quad (\bot, \mathtt{rsk}_1, \ldots, \mathtt{rsk}_{|\vec{V}|})\big)$
Output(Write$(c \leftarrow \Pi.Forge_{\mathtt{pp}}(\mathtt{spk}_i, \vec{v}', m, \vec{s}))$)
```

Algorithm 17 Additional **KGA** interfaces for the Forge converters.

```
▷ $(P \in \mathcal{F}$, Forge)-PublicParameters
Output$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}})$

▷ $(P \in \mathcal{F}$, Forge)-SenderPublicKey$(A_i \in \mathcal{S})$
Output$(\mathtt{spk}_i)$

▷ $(P \in \mathcal{F}$, Forge)-ReceiverPublicKey$(B_j \in \mathcal{R}^H)$
Output$(\mathtt{rpk}_j)$

▷ $(P \in \mathcal{F}$, Forge)-ReceiverKeyPair$(B_j \in \overline{\mathcal{R}^H})$
Output$(\mathtt{rpk}_j, \mathtt{rsk}_j)$
```

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.: DeCAF: Decentralizable continuous group key agreement with fast healing. In: Galdi, C., Phan, D.H. (eds.) SCN 24. LNCS, vol. 14974, pp. 294–313. Springer, Cham (Sep 2024). https://doi.org/10.1007/978-3-031-71073-5_14
4. 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
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.) 32nd

USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 1253–1270. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/balbas
14. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 409–426. Springer, Berlin, Heidelberg (May / Jun 2006). https://doi.org/10.1007/11761679_25
15. 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
16. 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
17. 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
18. 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
19. 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
20. Chakraborty, S., Hofheinz, D., Maurer, U., Rito, G.: Deniable authentication when signing keys leak. Cryptology ePrint Archive, Report 2023/213 (2023), https://eprint.iacr.org/2023/213
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. Dodis, Y., Katz, J., Smith, A., Walfish, S.: Composability and on-line deniability of authentication. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 146–162. Springer, Berlin, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_10
25. Herlihy, M., Shavit, N.: The art of multiprocessor programming. Morgan Kaufmann (2008)
26. 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

27. 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
28. 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
29. 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
30. 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
31. 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
32. 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
33. 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
34. 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
35. 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
36. Maurer, U., Portmann, C., Rito, G.: Multi-designated receiver signed public key encryption. Cryptology ePrint Archive, Report 2022/256 (2022), https://eprint.iacr.org/2022/256
37. Maurer, U., Renner, R.: Abstract cryptography. In: Chazelle, B. (ed.) ICS 2011. pp. 1–21. Tsinghua University Press (Jan 2011)
38. 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
39. 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
40. 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
41. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), https://eprint.iacr.org/2004/332

42. Wallez, T., Protzenko, J., Beurdouche, B., Bhargavan, K.: Treesync: Authenticated group management for messaging layer security. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 1217–1233. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/wallez
43. Weidner, M.: Group messaging for secure asynchronous collaboration. Master's thesis (2019)
44. 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 Illustration of Contributions

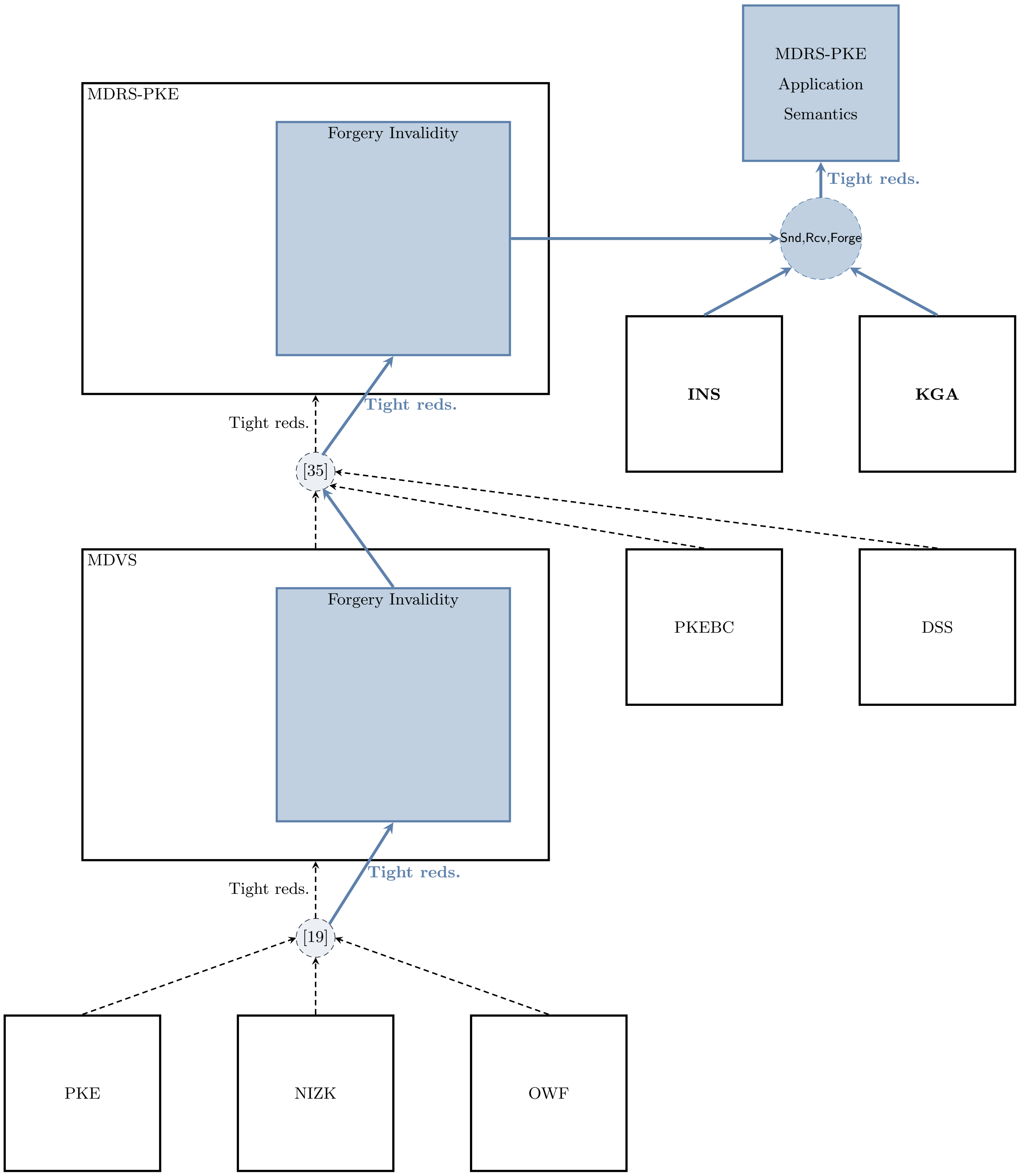

Fig. 1: Illustration of scheme related contributions. In blue are the new security notions—Forgery Invalidity for MDVS and MDRS-PKE schemes, and the application semantics for MDRS-PKE schemes—the new constructions—the construction of the ideal MDRS-PKE application from an MDRS-PKE providing the Forgery Invalidity guarantee (and the guarantees already defined in earlier work [19, 35])—and the new results—the proof Chakraborty et al.'s MDVS [19] satisfies Forgery Invalidity, the proof that Maurer et al.'s MDRS-PKE [35] satisfies Forgery Invalidity provided the underlying MDVS does too, and the composable construction proof—given in this paper, related to MDRS-PKE schemes. In the illustration, "Tight reds." means all the security reductions are tight.

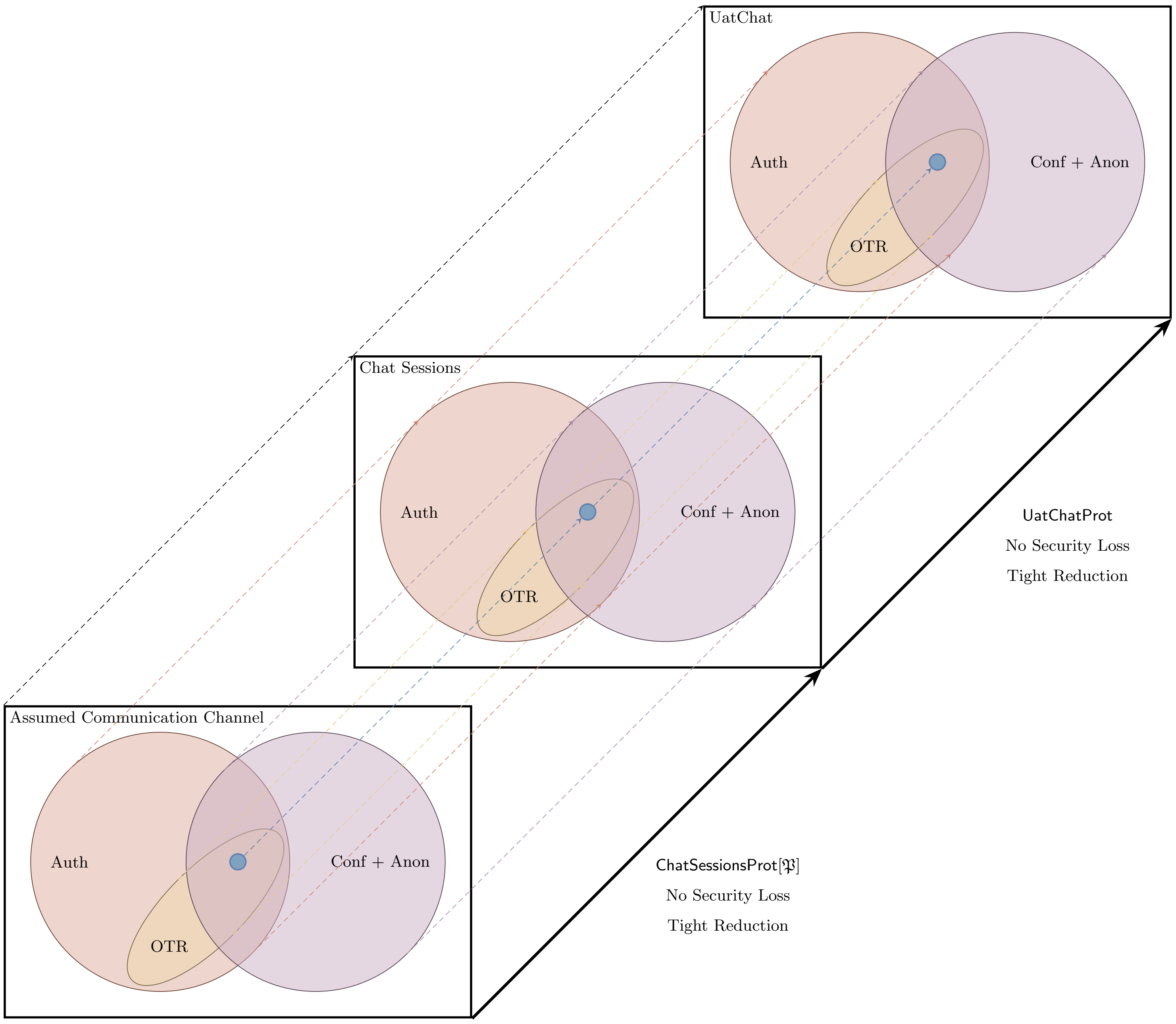

Fig. 2: Illustration of contributions related to composable constructions. In the figure, "Auth" denotes Authenticity, "OTR" Off-The-Record and "Conf + Anon" Confidentiality and Anonymity. The Venn diagram in each box illustrates additional security guarantees (i.e. beyond the ones already provided elements in the box). The $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ and $\mathsf{UatChatProt}$ constructions are Authenticity, Off-The-Record and Confidentiality plus Anonymity preserving, meaning that if the underlying assumed resources—i.e. the Assumed Communication Channel for the case of $\mathsf{ChatSessionsProt}[\mathfrak{P}]$ and the Chat Sessions assumed resource for the case of $\mathsf{UatChatProt}$—provides any of these guarantees, then the constructed resource (**ChatSessions**[$\mathfrak{P}$]and UatChat, respectively) provides them too. The blue circle in the center of each box (which corresponds to the intersection of all the additional properties) denotes the guarantees provided by the MDRS-PKE application semantics; as already explained (and as illustrated in the figure) our results imply that, when instantiated with the MDRS-PKE type of communication channel, the resulting messaging application provides authenticity, off-the-record, confidentiality and anonymity.

B (Simplified) Constructive Cryptography

In this section we introduce a (rather) simplified version of the Constructive Cryptography (CC) framework [33, 37] that suffices for our paper's statements. This simplified framework allows for (fine-grained) simulator-based type of security notions and requires little to no familiarity with CC. We note that all construction statements trivially carry to CC.

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. Akin to functionalities in UC [17], resources are interactive systems. Just like a mathematical 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 $\mathcal{Y}$ (both being non-empty), we say $\mathbf{R}$ is an $(\mathcal{X}, \mathcal{Y})$ resource. Similarly to functions, one can provide inputs $x \in \mathcal{X}$ to an $(\mathcal{X}, \mathcal{Y})$-resource, which then provides some output $y \in \mathcal{Y}$. Formally, resources are random systems [38, 39]; in turn, a random system is defined as a sequence of conditional probability distributions [39, 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 $\mathbf{R} \equiv \mathbf{S}$ [39, Definition 3]. For simplicity, we usually 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 $\mathbf{S}$.[28]. 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—corresponding to attaching $\mathbf{R}_1, \ldots, \mathbf{R}_n$ together—by $\mathbf{R} \coloneqq [\mathbf{R}_1, \ldots, \mathbf{R}_n]$, and call $\mathbf{R}$ the parallel composition of $\{\mathbf{R}_i\}_{i=1}^n$.

Interfaces. 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 $I_\mathcal{X}$ (respectively, $I_\mathcal{Y}$) an input (respectively, output) interface of $\mathbf{R}$. 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 $I_{1,\mathcal{Y}} \subseteq I_{2,\mathcal{Y}}$ (i.e. the input and output interfaces of I_1 are, respectively, a subset of the input and output interfaces of I_2). 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 $I_{1,\mathcal{Y}} \cap I_{2,\mathcal{Y}} = \emptyset$ (i.e. the input and output interfaces of I_1 are, respectively, disjoint with the input and output interfaces of I_2). We define the union of interfaces I_1 and I_2, denoted $I_1 \cup I_2$, 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. $(\mathcal{X}, \mathcal{Y}) = \bigcup_{I \in \mathcal{I}} I$.

[28] 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. [31, Definition 7]); this is not the case for all pairs of resources.

When considering (simulator-based) security notions it is often helpful to have the notion of a party. For a set of n parties $\mathcal{P} \coloneqq (P_1, \ldots, P_n)$, 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.

Converters. 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 (the entirety of) Alice's interface A of a resource $\mathbf{R}$ results in a new resource, which we denote by $\alpha^A\mathbf{R}$; we denote the inside interface of α by $\alpha.in$. The outside interface of the converter α is now the new A-interface of $\alpha^A\mathbf{R}$; we denote it by $\alpha.out$. Thus, a converter may be seen as a map between resources. Converters applied at different interfaces commute [27, 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 any set $\mathcal{S}$ we denote $(\pi_{P_i})_{P_i \in \mathcal{S}}\mathbf{R}$ by $\pi^{\mathcal{S}}\mathbf{R}$. We also often drop the interface superscript and write just $\pi\mathbf{R}$ when it is clear from the context to which interfaces π connects.

Distinguishers. To measure the distance between two resources we use the standard notion of a distinguisher, an interactive system $\mathbf{D}$ which interacts with a resource at all its interfaces, and outputs a bit 0 or 1. The distinguishing advantage for distinguisher $\mathbf{D}$ is defined as

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) \coloneqq |\Pr[\mathbf{DS} = 1] - \Pr[\mathbf{DR} = 1]|$$

where $\mathbf{DR}$ and $\mathbf{DS}$ are the random variables over the output of $\mathbf{D}$ when it interacts with $\mathbf{R}$ and $\mathbf{S}$, respectively.

Reductions. Typically one proves that the ability to distinguish between two resources is bounded by some function of the distinguisher, e.g. for any $\mathbf{D}$,

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) \leq |\varepsilon(\mathbf{D})|$$

where $\varepsilon(\mathbf{D})$ might be the probability that $\mathbf{D}$ can win a game or solves some problem believed to be hard.[29]

Security Statements. We now have all the elements needed to define a cryptographic construction.

Definition 5 (Simulation-based 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 ε*-constructs* $\mathbf{S}$ *from* $\mathbf{R}$ *if there is a simulator* sim *such that for any distinguisher* $\mathbf{D}$*,* $\Delta^{\mathbf{D}}(\pi\mathbf{R}, \mathsf{sim}\mathbf{S}) \leq \varepsilon(\mathbf{D})$ *and the interfaces of* sim*, of* π *and* I_F *are all pairwise disjoint.*

[29] Formally, one first finds an (efficient) reduction χ which constructs a solver $\mathbf{S} = \chi(\mathbf{D})$ from any distinguisher $\mathbf{D}$, and then one bounds $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S})$ by (a function of) the probability that $\chi(\mathbf{D})$ succeeds in solving some problem, e.g. $\chi(\mathbf{D})$ outputs the discrete-logarithm $x \in \mathbb{Z}_q$ of some group element $X = g^x \in \mathbb{G}$ for some group $\mathbb{G}$ of prime order q.

C ChatSessions: Full Proofs

C.1 Proof of Theorem 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 $\mathbf{R}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP})$:

$$\mathbf{R}[\mathfrak{P}] \rightsquigarrow \mathbf{H_1^{RW}} \rightsquigarrow \mathbf{H_2^{RW}} \rightsquigarrow \mathbf{H_3^{RW}} \rightsquigarrow \mathbf{H_4^{RW}} \rightsquigarrow \mathbf{H_5^{RW}} \rightsquigarrow \mathbf{H_6^{RW}} \rightsquigarrow \mathbf{H_{Mid}^{RW}},$$

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

$$\mathbf{ChatSessions}[\mathfrak{P}] \rightsquigarrow \mathbf{H_1^{IW}} \rightsquigarrow \mathbf{H_2^{IW}} \rightsquigarrow \mathbf{H_3^{IW}} \rightsquigarrow \mathbf{H_4^{IW}} \rightsquigarrow \mathbf{H_{Mid}^{IW}}.$$

The last hop of the proof is then

$$\mathbf{H_{Mid}^{RW}} \rightsquigarrow \mathbf{H_{Mid}^{IW}}.$$

More concretely, our proof will establish that all of these are *statistically* the same, i.e.

$$\mathbf{R}[\mathfrak{P}] \equiv \mathbf{H_1^{RW}} \equiv \mathbf{H_2^{RW}} \equiv \mathbf{H_3^{RW}} \equiv \mathbf{H_4^{RW}} \equiv \mathbf{H_5^{RW}} \equiv \mathbf{H_6^{RW}} \equiv \mathbf{H_{Mid}^{RW}}$$
$$\equiv \mathbf{H_{Mid}^{IW}} \equiv \mathbf{H_4^{IW}} \equiv \mathbf{H_3^{IW}} \equiv \mathbf{H_2^{IW}} \equiv \mathbf{H_1^{IW}} \equiv \mathbf{ChatSessions}[\mathfrak{P}].$$

Helper Propositions. Before moving to the hybrids, we first establish some useful facts.

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

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

See Section C.1.1 for the proof of Proposition 1.

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

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

See Section C.1.2 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 C.1.3 for the proof of Proposition 3.

Proposition 4. *In* **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 C.1.4 for the proof of Proposition 4.

The following is a direct consequence of Proposition 4.

Corollary 4. *In* **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* **ChatSessions**$[\mathfrak{P}]$*, the extended versions of the graphs in* SessionGraphs *are proper.*

See Section C.1.5 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 C.1.6 for the proof of Proposition 6.

Proposition 7. *In* $\mathbf{H}_4^{\mathbf{RW}}$*, for each* $\mathtt{sid}$ *and* $P \in \mathcal{M}^H$*,* $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ *is proper.*

See Section C.1.7 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 C.1.8 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 C.1.9 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 $i = 1, \dots, n$, 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

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

Then,

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

See Section C.1.10 for the proof of Proposition 10.

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

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

we have

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

See Section C.1.11 for the proof of Proposition 11.

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

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

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

$$(\mathcal{G}_1, S''_1) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}', S')$$
$$(\mathcal{G}_2, S''_2) \coloneqq \mathsf{UpdatedGraph}(\mathcal{G}, S' \cup \{\mathtt{id}\}),$$

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

See Section C.1.12 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 C.1.13 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 C.1.14 for the proof of Proposition 14.

Hybrid Sequence. In the hybrids' descriptions (Algorithms 19, 20, 21, 22, 23, 24, 25, 26, 27 and 28) we only show the differences relative to the previous hybrid (or relative to the ideal world system **ChatSessions**[$\mathfrak{P}$] or real world system **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).

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 $\mathbf{R}[\mathfrak{P}] \coloneqq \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \mathbf{REP})$—defined in Equation 4.2—and $\mathbf{H}^{\mathbf{RW}}_{\mathbf{1}}$—defined in Algorithm 19—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 **REP** (Algorithms 1 and 2) each `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 20) 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 `sid`, in hybrid $\mathbf{H}^{\mathbf{RW}}_{\mathbf{2}}$ it holds that $\text{ToHandle}[\texttt{sid}] = \text{ToHandle}_P[\texttt{sid}]$; we now establish this.

Fix some `sid`.

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

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

Algorithm 18 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
    $\text{Contents},\ \text{SessionGraphs}_{\text{Global}},\ \text{ToHandle}_{\text{Global}} \leftarrow \emptyset$
    for $P \in \mathcal{M}^H$ :
      $\text{Sent}[P],\ \text{SessionGraphs}_P,\ \text{ToHandle}_P,\ \text{Undelivered}_P,\ \text{Delivered}_P \leftarrow \emptyset$

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

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

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

  $(P \in \overline{\mathcal{P}^H})$-READ
    OUTPUT$((\mathsf{Net}\cdot\mathbf{REP})$-READ$)$

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

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

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

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

Algorithm 19 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
  $(\mathsf{Net} \cdot \mathbf{REP})$-INITIALIZATION
  $\text{Contents} \leftarrow \emptyset$
  for $P \in \mathcal{M}^H$ :
    $\text{SessionGraphs}_P \leftarrow \emptyset$

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

$(P \in \mathcal{M}^H)$-READ
  $\mathsf{ProcessReceived}(P)$
  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$(\langle S \rightarrow \vec{V} \rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\texttt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})\text{-WRITE}(\langle S \rightarrow \vec{V} \rangle, m := (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Contents}[\texttt{id}] \leftarrow (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  OUTPUT$(\texttt{id})$
```

```
$\mathsf{ProcessReceived}(P)$   // Not part of interface.
  $\text{ToHandle} \leftarrow \emptyset$
  for $(\texttt{id}, (\langle S \rightarrow \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))) \in (\mathsf{Net} \cdot \mathbf{REP})\text{-READ}$ 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 20 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$(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$
  $\mathsf{ProcessReceived}(P)$
  $\mathcal{G}_P := (V_P, E_P) \leftarrow \text{SessionGraphs}_P[\texttt{sid}]$
Require: $\mathfrak{P}[\text{IsValid}](\texttt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \texttt{cmd}, \text{Acks})$
  $\texttt{id} \leftarrow (\mathsf{Net}\cdot\mathbf{REP})\text{-Write}(\langle P \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{Contents}[\texttt{id}] \leftarrow (\langle P \to \vec{V}\rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks}))$
  $\text{SessionGraphs}_P[\texttt{sid}] \leftarrow (V_P \cup \{\texttt{id}\}, E_P \cup (\text{Acks} \times \{\texttt{id}\}))$
  $\forall P' \in (\text{Set}(\vec{V})^H \setminus \{P\}) : \text{Undelivered}_{P'}[\texttt{sid}] \leftarrow \text{Undelivered}_{P'}[\texttt{sid}] \cup \{\texttt{id}\}$
  Output$(\texttt{id})$

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

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

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

Algorithm 21 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: $\mathfrak{P}[\mathrm{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \mathtt{cmd}, \mathrm{Acks})$
    $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})$-WRITE($\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})$)
    $\mathrm{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
    $\mathrm{SessionGraphs}_P[\mathtt{sid}] \leftarrow (V_P \cup \{\mathtt{id}\}, E_P \cup (\mathrm{Acks} \times \{\mathtt{id}\}))$
    $\mathsf{ProcessReceived}(P)$
    $\forall P' \in (\mathrm{Set}(\vec{V})^H \setminus \{P\}) : \mathrm{Undelivered}_{P'}[\mathtt{sid}] \leftarrow \mathrm{Undelivered}_{P'}[\mathtt{sid}] \cup \{\mathtt{id}\}$
    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($P, \mathtt{id}$)
    $(\mathsf{Net} \cdot \mathbf{REP})$-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 22 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: $\mathfrak{P}[\mathrm{IsValid}](\mathtt{sid}, \mathsf{Extended}(\mathcal{G}_P), P, \vec{V}, \mathtt{cmd}, \mathrm{Acks})$
    $\mathtt{id} \leftarrow (\mathsf{Net} \cdot \mathbf{REP})$-WRITE($\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks})$)
    $\mathrm{Contents}[\mathtt{id}] \leftarrow (\langle P \to \vec{V} \rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))$
    [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 23 Hybrid $\mathbf{H}_{\mathbf{5}}^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_{\mathbf{4}}^{\mathbf{RW}}$.

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

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

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

Algorithm 24 Hybrid $\mathbf{H}_{\mathbf{6}}^{\mathbf{RW}}$. We only show the differences relative to $\mathbf{H}_{\mathbf{5}}^{\mathbf{RW}}$.

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

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

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

$\mathbf{H}_\mathbf{2}^\mathbf{RW} \rightsquigarrow \mathbf{H}_\mathbf{3}^\mathbf{RW}$**:** The only difference between $\mathbf{H}_\mathbf{2}^\mathbf{RW}$ and $\mathbf{H}_\mathbf{3}^\mathbf{RW}$ (Algorithm 21) is that, for each party $P \in \mathcal{M}^H$: in the latter $\mathsf{ProcessReceived}(P)$ is called 1. upon each $\text{Delivery}(P, \cdot)$ query, and 2. on each Write query at P's interface, after adding the resulting node to $\text{SessionGraphs}_P[\texttt{sid}]$; 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 $\mathbf{R}[\mathfrak{P}] \rightsquigarrow \mathbf{H}_\mathbf{1}^\mathbf{RW} \rightsquigarrow \mathbf{H}_\mathbf{2}^\mathbf{RW}$: for each $P \in \mathcal{M}^H$ and each $\texttt{sid}$, $\text{SessionGraphs}_P[\texttt{sid}]$ in $\mathbf{H}_\mathbf{2}^\mathbf{RW}$ 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 $\text{SessionGraphs}[\texttt{sid}]$ 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 $\text{SessionGraphs}[\texttt{sid}]$ 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 $\text{ToHandle}_P[\texttt{sid}] = \emptyset$;[30]

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

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

[30] 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.

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 $\mathbf{H}_2^{\mathbf{RW}}$, $\text{SessionGraphs}_P[\texttt{sid}]$ 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 $\mathsf{ProcessReceived}$ 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 $n' \coloneqq n + 1$, let S_1 be the set of nodes in $\text{ToHandle}_P[\texttt{sid}]$ right after $\mathsf{ProcessReceived}(P)$ is called in the beginning of query q_{Write}—i.e. $S_1 \coloneqq \text{ToHandle}_P[\texttt{sid}]$—and for $i = 2, \ldots, n'$, let $S_i \coloneqq D_{i-1}$;
- if the last query to P's interface was a Read query, say q_{Read}, then let $n' \coloneqq n$, let S_1 be the set of nodes in $\text{ToHandle}_P[\texttt{sid}]$ right after the call to $\mathsf{ProcessReceived}(P)$ in the beginning of query q_{Read} together with D_1—i.e. $S_1 \coloneqq \text{ToHandle}_P[\texttt{sid}] \cup D_1$—and for $i = 2, \ldots, n'$, let $S_i \coloneqq D_i$;
- if there was no prior query to P's Read or Write interfaces, then let $n' \coloneqq n$, and for $i = 1, \ldots, n'$, let $S_i \coloneqq D_i$.

Note that in all cases
$$S = \bigcup_{i=1,\ldots,n'} S_i$$
and so by Proposition 10 it then follows $\mathbf{H}_2^{\mathbf{RW}} \equiv \mathbf{H}_3^{\mathbf{RW}}$.

$\mathbf{H}_3^{\mathbf{RW}} \rightsquigarrow \mathbf{H}_4^{\mathbf{RW}}$: The only difference between $\mathbf{H}_3^{\mathbf{RW}}$ and $\mathbf{H}_4^{\mathbf{RW}}$ is that in $\mathbf{H}_4^{\mathbf{RW}}$ (Algorithm 22), 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 $\text{SessionGraphs}_P[\texttt{sid}]$, the node is instead added to set $\text{ToHandle}_P[\texttt{sid}]$. However, it follows from Proposition 12 that in the two cases both $\text{SessionGraphs}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$ are still the same at the end of the $(P \in \mathcal{M}^H)$-Write query. Therefore, $\mathbf{H}_3^{\mathbf{RW}} \equiv \mathbf{H}_4^{\mathbf{RW}}$.

$\mathbf{H}_4^{\mathbf{RW}} \rightsquigarrow \mathbf{H}_5^{\mathbf{RW}}$: The only difference between $\mathbf{H}_4^{\mathbf{RW}}$ and $\mathbf{H}_5^{\mathbf{RW}}$ (Algorithm 23) is that for a party P and some $\texttt{sid}$, $\mathbf{H}_5^{\mathbf{RW}}$ now computes $\mathsf{InducedPartyGraph}^+$ on $\text{SessionGraphs}_P[\texttt{sid}]$ instead of simply using this graph. To prove $\mathbf{H}_4^{\mathbf{RW}} \equiv \mathbf{H}_5^{\mathbf{RW}}$ it suffices to show that when, in $\mathsf{InducedPartyGraph}^+$, graph $\mathcal{G}_P \coloneqq (V_P, E_P)$ is set to $\text{SessionGraphs}_P[\texttt{sid}]$, the output of $\mathsf{InducedPartyGraph}^+(\texttt{sid}, P)$ is $\mathsf{Extended}(\text{SessionGraphs}_P[\texttt{sid}])$. To begin, note that from Proposition 7 each graph $\text{SessionGraphs}_P[\texttt{sid}]$ in $\mathbf{H}_4^{\mathbf{RW}}$ is proper. Furthermore, it is easy to see that the set of edges E of the graph $\mathcal{G} \coloneqq (V, E)$ output by $\mathsf{InducedPartyGraph}^+$ is such that, for function f defined in Proposition 2—i.e. $f(u) \coloneqq \text{Acks} \times \{\texttt{id}\}$—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 $\text{SessionGraphs}_P[\texttt{sid}]$. Below we prove V includes all nodes in $\text{SessionGraphs}_P[\texttt{sid}]$ (the other direction follows trivially from inspection of $\mathsf{InducedPartyGraph}^+$).

Letting $\text{SessionGraphs}_P[\texttt{sid}] \coloneqq \mathcal{G}_P \coloneqq (V_P, E_P)$, by Definition 1, for $n = |V_P|$, 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 $i = 0, \ldots, n-1$,

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

it holds that

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

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

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

Since u_j is not a root node, it follows from Requirement 2 that for each $\texttt{id} \in u_j.\text{Acks}$ there is a node $(\texttt{id}, \cdot) \in \mathcal{G}_{j-1}^+.V^+$. 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 $u_j.\text{Acks} \subseteq V_{j-1}$,

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

and so

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

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 24) are the three new variables $\text{SessionGraphs}_{\text{Global}}$, $\text{ToHandle}_{\text{Global}}$ and $\text{Sent}[P]$ (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 `id` is added and removed from set $\text{Undelivered}_P[\texttt{sid}]$, and it is also added to set $\text{Delivered}_P[\texttt{sid}]$. First, note that the behavior of $\mathbf{H_6^{RW}}$ is independent of the two new variables and of $\text{Delivered}_P[\texttt{sid}]$, implying that adding `id` to $\text{Delivered}_P[\texttt{sid}]$ does not affect $\mathbf{H_6^{RW}}$'s behavior. Regarding adding and then removing `id` from set $\text{Undelivered}_P[\texttt{sid}]$:

- if `id` was not in set $\text{Undelivered}_P[\texttt{sid}]$ prior to the query, then adding and removing it from the set has no side-effects;
- if `id` was already in $\text{Undelivered}_P[\texttt{sid}]$ (prior to the query) then it is removed from the set. However, in this case the only difference is that, because `id` is removed, upon a query DELIVER$(P, \texttt{id})$, it is not added to sets $\text{Delivered}_P[\texttt{sid}]$ and $\text{ToHandle}_P[\texttt{sid}]$. However, on one hand, as we already explained $\mathbf{H_6^{RW}}$ is independent of $\text{Delivered}_P[\texttt{sid}]$, and on the other hand, `id` is added to $\text{ToHandle}_P[\texttt{sid}]$ 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 $\mathbf{H_5^{RW}} \equiv \mathbf{H_6^{RW}}$.

$\mathbf{H_6^{RW}} \rightsquigarrow \mathbf{H_{Mid}^{RW}}$: Note that $\mathbf{H_6^{RW}}$ and $\mathbf{H_{Mid}^{RW}}$ (Algorithm 18) have the same description, so $\mathbf{H_5^{RW}} \equiv \mathbf{H_{Mid}^{RW}}$.

Algorithm 25 Hybrid $\mathbf{H_1^{IW}}$ for the proof (Section C.1) of Theorem 1. In the description below we only show what is different relative to **ChatSessions**$[\mathfrak{P}]$.

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

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

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

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

Algorithm 26 Hybrid $\mathbf{H_2^{IW}}$. We only show the differences relative to $\mathbf{H_1^{IW}}$.

```
Initialization
  (Net · REP)-Initialization
  Contents,  SessionGraphs_Global,  ToHandle_Global ← ∅
  for $P \in \mathcal{M}^H$ :
    Sent[$P$],  ToHandle$_P$,  Undelivered$_P$,  Delivered$_P$ ← ∅

($P \in \mathcal{M}^H$)-Write(sid, cmd, $\vec{V}$, Acks)
  $\mathcal{G}^+$ ← InducedPartyGraph$^+$(sid, $P$)
Require: $\mathfrak{P}$[IsValid](sid, $\mathcal{G}^+$, $P$, $\vec{V}$, cmd, Acks)
  id ← (Net · REP)-Write($\langle P \to \vec{V} \rangle$, $m$ := (sid, cmd, Acks))
  Contents[id] ← ($\langle P \to \vec{V} \rangle$, (sid, cmd, Acks))
  Sent[$P$] ← Sent[$P$] ∪ {id}
  AddToGraph(sid, id)
  $\forall P' \in (\mathrm{Set}(\vec{V})^H \setminus \{P\})$ : Undelivered$_{P'}$[sid] ← Undelivered$_{P'}$[sid] ∪ {id}
  Output(id)

($P \in \overline{\mathcal{P}^H}$)-Write($\langle S \to \vec{V} \rangle$, $m$ := (sid, cmd, Acks))
  id ← (Net · REP)-Write($\langle S \to \vec{V} \rangle$, $m$ := (sid, cmd, Acks))
  Contents[id] ← ($\langle S \to \vec{V} \rangle$, (sid, cmd, Acks))
  AddToGraph(sid, id)
  $\forall P' \in \mathrm{Set}(\vec{V})^H$ : Undelivered$_{P'}$[sid] ← Undelivered$_{P'}$[sid] ∪ {id}
  Output(id)

Deliver($P$, id)
  (Net · REP)-Deliver($P$, id)
  if ∃sid  such that  id ∈ Undelivered$_P$[sid] ∧ $P \in \mathcal{M}^H$ :
    Undelivered$_P$[sid] ← Undelivered$_P$[sid] \ {id}
    Delivered$_P$[sid] ← Delivered$_P$[sid] ∪ {id}
    ToHandle$_P$[sid] ← ToHandle$_P$[sid] ∪ {id}
```

Algorithm 27 Hybrid $\mathbf{H_3^{IW}}$. We only show the differences relative to $\mathbf{H_2^{IW}}$.

```
InducedPartyGraph$^+$(sid, $P$)  // Not part of interface.
  $\mathcal{G}$ := ($V$, $E$) ← SessionGraphs_Global[sid]
  $V_P \leftarrow V \cap$ {id | id ∈ Delivered$_P$[sid] ∪ Sent[$P$]}
  $V_0 \leftarrow V_P \cap$ {id | Contents[id] = ($\langle S \to \vec{V} \rangle$, (sid, cmd, ·)) ∧ $\mathfrak{P}$[IsRoot](sid, $S$, $\vec{V}$, cmd)}
  $i \leftarrow 0$
  repeat
    $V_{i+1} \leftarrow V_i$
    for id ∈ $V_P$ :
      (·, (·, ·, Acks)) ← Contents[id]
      if Acks ⊆ $V_i$ :
        $V_{i+1} \leftarrow V_{i+1} \cup$ {id}
    $i \leftarrow i + 1$
  until $V_i = V_{i-1}$
  $V_E$ := {id | (id, id′) ∈ $E$}
  return Extended($\mathcal{G}_i$ := ($V_i$, $E \cap (V_E \times V_i)$))
```

Algorithm 28 Hybrid $\mathbf{H}_4^{\mathbf{IW}}$. We only show the differences relative to $\mathbf{H}_3^{\mathbf{IW}}$.

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

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

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

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

ChatSessions$[\mathfrak{P}] \rightsquigarrow \mathbf{H_1^{IW}}$**:** **ChatSessions**$[\mathfrak{P}]$ (Algorithm 6) and $\mathbf{H_1^{IW}}$ (defined in Algorithm 25) only differ in the names of variables SessionGraphs and ToHandle, and so **ChatSessions**$[\mathfrak{P}] \equiv \mathbf{H_1^{IW}}$.

$\mathbf{H_1^{IW}} \rightsquigarrow \mathbf{H_2^{IW}}$**:** The only difference between $\mathbf{H_1^{IW}}$ and $\mathbf{H_2^{IW}}$ (Algorithm 26) is that in $\mathbf{H_2^{IW}}$ 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 $\mathbf{H_2^{IW}}$, so $\mathbf{H_1^{IW}} \equiv \mathbf{H_2^{IW}}$.

$\mathbf{H_2^{IW}} \rightsquigarrow \mathbf{H_3^{IW}}$**:** Hybrid $\mathbf{H_3^{IW}}$ (Algorithm 27) only differs from $\mathbf{H_2^{IW}}$ in what the variable V_P in the $\mathsf{InducedPartyGraph}^+$ procedure is set to: for a party P, in $\mathbf{H_2^{IW}}$, 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 $\mathbf{H_3^{IW}}$ it is set to the union of $\text{Delivered}_P[\mathtt{sid}]$ and $\text{Sent}[P]$. However, from inspection of $\mathbf{H_3^{IW}}$ 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 $\mathtt{id} \in \text{Delivered}_P[\mathtt{sid}]$ *if and only if* there is a node $u := (\mathtt{id}, (\cdot, (\mathtt{sid}, \cdot, \cdot)))$ 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 28) is the additional variable SessionGraphs_P (see Algorithm 28), that upon a $(P \in \mathcal{M}^H)$-WRITE query the resulting $\mathtt{id}$ 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 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 $\mathtt{id}$ may only have side effects if $\mathtt{id}$ is already in $\text{Undelivered}_P[\mathtt{sid}]$ as this may prevent a later DELIVER$(P, \mathtt{id})$ query from adding $\mathtt{id}$ to variable $\text{Delivered}_P[\mathtt{sid}]$—if $\mathtt{id}$ is not in $\text{Undelivered}_P[\mathtt{sid}]$, then adding and removing it has no side effects. However, even in case $\mathtt{id}$ is in $\text{Undelivered}_P[\mathtt{sid}]$, noting that $\mathtt{id}$ 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 18) 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 18), 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 $\mathtt{sid}$, $\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}]$ is proper; and
– in $\mathbf{H_{Mid}^{RW}}$, for each $\mathtt{sid}$ 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 $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ or as in $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. For both $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$, the graph $\mathcal{G}^+ := (V^+, E^+)$ 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.

$$f(u := (\mathtt{id}, (\cdot, (\cdot, \cdot, \mathrm{Acks})))) := \mathrm{Acks} \times \{\mathtt{id}\};$$

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

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

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* $\mathrm{ToHandle}_P[\mathtt{sid}]$ *is in* $\mathrm{Delivered}_P[\mathtt{sid}]$.

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 18). □

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

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 18). □

Fact 3. *Any root that was added to* $\mathrm{ToHandle}_P[\mathtt{sid}]$ *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 $\mathrm{ToHandle}_P$ there is a subsequent call—during the same query—to $\mathsf{ProcessReceived}(P)$.

Consider any root node

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

First note that $\mathrm{SessionGraphs}_P[\mathtt{sid}]$ is proper, and that $\mathsf{UpdatedGraph}$ constructs the output graph following the definition of proper graph (Definition 1); 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

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

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 $\textsc{Deliver}$ and $(P \in \mathcal{M}^H)$-$\textsc{Write}$ in hybrids $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$ (Algorithm 18). □

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 $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$ (Algorithm 18), nodes are only added to $\text{Undelivered}_P[\texttt{sid}]$ in $\textsc{Write}$ 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* $\text{ToHandle}_{\textit{Global}}[\texttt{sid}]$ *is added to* V^{Global}.

Proof (Fact 8). From inspection of both $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ and $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$, 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 18, 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* $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{RW}}$ *or* $\mathbf{H}_{\mathbf{Mid}}^{\mathbf{IW}}$*, 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*

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

where $\mathcal{G}^+_{\textit{Local}}$ *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. $\square$

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 18), 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 $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$ 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 $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$: **S.2** implies that every node in the graph output by $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ is also in the graph output by $\mathsf{InducedPartyGraph}^+$ in $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$; 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 $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. So, establishing **S.2** and **S.3** implies $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.

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

$$v_1, \dots, v_n$$

as in Definition 1. 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' ($v \rightsquigarrow \dots \rightsquigarrow u'$) 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 $j \in \{1, \dots, n\}$ 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 ($v \rightsquigarrow \dots \rightsquigarrow v_j$) where each node in the path is not a root. Say $v_j := (\texttt{id}, (\langle S \to \vec{V} \rangle, (\texttt{sid}, \texttt{cmd}, \text{Acks})))$. 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 $v_j \in V_P^{\text{Global}}$, it follows from definition of V_P^{Global} and from Fact 10 that $v_j \in V^{\text{Global}}$ and $v_j \in \text{Delivered}_P[\texttt{sid}]$. From Fact 2, we have v_j was added to $\text{ToHandle}_P[\texttt{sid}]$. Since a node is only removed from $\text{ToHandle}_P[\texttt{sid}]$ when it is added to V_P^{Local}, and $v_j \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$, it follows v_j is currently in $\text{ToHandle}_P[\texttt{sid}]$. Since $v_j \in V^{\text{Global}}$ and nodes are only added to V^{Global} via $\mathsf{UpdatedGraph}$—which constructs the output graph following Definition 1 (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

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

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

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

We have already established $\text{SessionGraphs}_P[\texttt{sid}]$ is proper; since $\mathcal{G}_i$ is also proper, it follows from Requirement 3 that $\text{Acks} \not\subseteq V_P^{\text{Local}}$. By Definition 1, since v_j is not a root and from Requirement 3, every node in $x \in \text{Acks}$ must appear before v_j in the sequence $v_1, \dots, v_n$. Taking any such $x \in \text{Acks} \setminus V_P^{\text{Local}}$ (which exists because we already concluded $\text{Acks} \not\subseteq V_P^{\text{Local}}$) we know $x = v_l$ for some $l < j$. To conclude the proof of **S.3'** (and therefore of **S.3**), consider two cases:

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

We now obtain a contradiction for both of these cases.

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

$v_l \notin \text{ToHandle}_P[\texttt{sid}]$: from Fact 1 we know $v_l \notin \text{Delivered}_P[\texttt{sid}]$, and since $v_l \in V^{\text{Global}}$, then $v_l \in V^{\text{Global}} \setminus \text{Delivered}_P[\texttt{sid}]$. However this is a contradiction with our assumption because $v_j \in V_P^{\text{Global}} \setminus V_P^{\text{Local}}$ and yet there is a node in $V^{\text{Global}} \setminus \text{Delivered}_P[\texttt{sid}]$, 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 $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}}$ and $\mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$.

Base case: Upon 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:

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

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

Queries $(P' \in \mathcal{P})$**-Read.** For both honest and dishonest parties, 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 Read query the claim still holds.

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

Queries P**-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 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 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 both 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[31] 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 $\mathbf{H}^{\mathbf{RW}}_{\mathbf{Mid}} \equiv \mathbf{H}^{\mathbf{IW}}_{\mathbf{Mid}}$. $\square$

[31] 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 1.

Proofs of Helper Propositions.

C.1.1 Proof of Proposition 1.

Proof. Consider any $u := (\mathtt{id}, (\langle S \to \vec{V}\rangle, (\mathtt{sid}, \mathtt{cmd}, \mathrm{Acks}))) \in V^+$. Definition 1 implies there is a proper subgraph of $\mathcal{G}^+$, say $\mathcal{G}'^+ = (V'^+, E'^+)$, such that

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

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

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

□

C.1.2 Proof of Proposition 2.

Proof. Follows from the definition of proper graph (Definition 1). □

C.1.3 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

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

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

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

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

$$\mathcal{G}_{i+1}^+ := (\mathcal{G}_{i+1}.V^+ \cup \{u\}, \mathcal{G}_{i+1}.E^+ \cup (\mathrm{Acks} \times \{\mathtt{id}\}))$$

is also proper. □

C.1.4 Proof of Proposition 4.

Proof. First note that from Algorithm 6 the graph output by $\mathsf{InducedPartyGraph}^+$ on input $(\mathtt{sid}, P)$ is a subgraph of (the extended version of) SessionGraphs[$\mathtt{sid}$], 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 $\mathcal{G}_i := (V_i, E_i := E \cap (V_E \times V_i))$ is proper. Noting that $E_i = \bigcup_{u \in V_i} f(u)$ for $f(u) := \mathrm{Acks} \times \{\mathtt{id}\}$ (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[$\mathtt{sid}$] by $\mathcal{G}^+_{\mathtt{sid}} = (V^+_{\mathtt{sid}}, E^+_{\mathtt{sid}})$.

To begin, note that V_P is a subset of the vertices of graph SessionGraphs[$\mathtt{sid}$], 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 1 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

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

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

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

By Proposition 1,

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

so

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

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

C.1.5 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 $(\mathtt{sid}, \mathtt{cmd}, \vec{V}, \text{Acks})$ be the input to the query. By assumption all graphs in SessionGraphs before the query are proper, and only SessionGraphs[$\mathtt{sid}$] is modified. Concretely, the new value of SessionGraphs[$\mathtt{sid}$] is the graph output by $\mathsf{UpdatedGraph}$. Since the graph input to $\mathsf{UpdatedGraph}$ is SessionGraphs[$\mathtt{sid}$], 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 $(\langle S \to \vec{V} \rangle, m \coloneqq (\texttt{sid}, \texttt{cmd}, \text{Acks}))$ be the corresponding input. By Proposition 3 and the assumption that all graphs in SessionGraphs before the query are proper, the output of $\mathsf{UpdatedGraph}$ is proper. Again, by the definition of **ChatSessions**$[\mathfrak{P}]$ (Algorithm 6), only SessionGraphs[$\texttt{sid}$] may be modified; if it is modified, it is set to the output of $\mathsf{UpdatedGraph}$, which is proper. It follows that after the query all graphs stored in SessionGraphs are still proper. □

C.1.6 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 Write$(\texttt{sid}, \texttt{cmd}, \vec{V}, \text{Acks})$, 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 $\texttt{sid}$ in the set ToHandle). □

C.1.7 Proof of Proposition 7.

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

C.1.8 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), $u \in S'$. □

C.1.9 Proof of Proposition 9.

Proof. We prove this by contradiction. Since $\mathcal{G} = (V, E)$ is proper, letting $n = |V|$, by Definition 1 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 $i = 0, \ldots, n-1$,

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

it holds $\textsc{IsValid}(u_{i+1}.\texttt{sid}, {\mathcal{G}_i}^{+}, u_{i+1}.S, u_{i+1}.\vec{V}, u_{i+1}.\texttt{cmd}, u_{i+1}.\text{Acks}) = 1$. For some set $V_S \subseteq V$, let $(\mathcal{G}_S, S') := \mathsf{UpdatedGraph}(\mathcal{G}, S)$, and furthermore let $(\mathcal{G}_{V_S}, {S_{V_S}}') := \mathsf{UpdatedGraph}(\mathcal{G}, S \cup V_S)$. By inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), graphs $\mathcal{G}_S$ and $\mathcal{G}_{V_S}$ are constructed according to Definition 1, 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 $u^{S\cup V_S}_l$ 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

$$\mathcal{G}^S_{j+1} := (V^S_j \cup \{u^S_{j+1}.\texttt{id}\}, E^S_j \cup (u^S_{j+1}.\text{Acks} \times \{u^S_{j+1}.\texttt{id}\})),$$
$$\mathcal{G}^{S\cup V_S}_{l+1} := (V^{S\cup V_S}_l \cup \{u^{S\cup V_S}_{l+1}.\texttt{id}\}, E^{S\cup V_S}_l \cup (u^{S\cup V_S}_{l+1}.\text{Acks} \times \{u^{S\cup V_S}_{l+1}.\texttt{id}\})),$$

it holds that

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

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^S_j in the sequence $u^S_{n+1}, \ldots, u^S_{(n+|S'|)}$ that is not in $V_{\mathcal{G}_{V_S}}$; u^S_j is not a root because this would contradict the assumption that $\mathfrak{P}$ satisfies Requirement 1. Given u^S_j is not a root, since

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

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

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

and so

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

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^S_j 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 $\mathcal{G}_S$). □

□

C.1.10 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

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

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 1 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 $i = 0, \ldots, n-1$, $\mathcal{G}_{i+1} \coloneqq (V_i \cup \{u_{i+1}.\texttt{id}\}, E_i \cup (u_{i+1}.\text{Acks} \times \{u_{i+1}.\texttt{id}\}))$, we have $\textsc{IsValid}(u_{i+1}.\texttt{sid}, {\mathcal{G}_i}^+, u_{i+1}.S, u_{i+1}.\vec{V}, u_{i+1}.\texttt{cmd}, u_{i+1}.\text{Acks}) = 1$. By inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6), graphs $\mathcal{G}'$, $\mathcal{G}_2$ and ${\mathcal{G}_3}'$ are constructed according to Definition 1, 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 C.1) such that, for $i = n, \ldots, (n + |S'|) - 1$, for $j = n, \ldots, (n + |{S_2}'|) - 1$, and for $l = (n + |{S_2}'|), \ldots, (n + |{S_2}'| + |S'_{\mathcal{G}_3'}|) - 1$,

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

it holds that

$$\begin{aligned}
&\textsc{IsValid}(u^{S'}_{i+1}.\mathtt{sid}, \left(\mathcal{G}^{S'}_i\right)^+, u^{S'}_{i+1}.S, u^{S'}_{i+1}.\vec{V}, u^{S'}_{i+1}.\mathtt{cmd}, u^{S'}_{i+1}.\text{Acks}) = 1,\\
&\textsc{IsValid}(u^{{S_2}'}_{j+1}.\mathtt{sid}, \left(\mathcal{G}^{{S_2}'}_j\right)^+, u^{{S_2}'}_{j+1}.S, u^{{S_2}'}_{j+1}.\vec{V}, u^{{S_2}'}_{j+1}.\mathtt{cmd}, u^{{S_2}'}_{j+1}.\text{Acks}) = 1,\\
&\textsc{IsValid}(u^{{\mathcal{G}_3}'}_{l+1}.\mathtt{sid}, \left(\mathcal{G}^{{\mathcal{G}_3}'}_l\right)^+, u^{{\mathcal{G}_3}'}_{l+1}.S, u^{{\mathcal{G}_3}'}_{l+1}.\vec{V}, u^{{\mathcal{G}_3}'}_{l+1}.\mathtt{cmd}, u^{{\mathcal{G}_3}'}_{l+1}.\text{Acks}) = 1.
\end{aligned}$$

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 $u^{{S_2}'}_j$ in the sequence $u^{{S_2}'}_{n+1}, \ldots, u^{{S_2}'}_{(n+|{S_2}'|)}$ such that $u^{{S_2}'}_j \notin V_{\mathcal{G}'}$. First, $u^{{S_2}'}_j$ cannot be a root node, as otherwise this would imply $\mathfrak{P}$ does not satisfy Requirement 1. Since $u^{{S_2}'}_j$ is not a root and noting that

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

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

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

and so

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

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

$V_{\mathcal{G}_3'} \setminus V_{\mathcal{G}'} = \emptyset$: 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. □

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

□

C.1.11 Proof of Proposition 11.

Proof. Follows from inspection of $\mathsf{UpdatedGraph}$ (Algorithm 6): consider any node

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

If it were the case that

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

u would be added in the last iteration of $\mathsf{UpdatedGraph}$. □

C.1.12 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 $\mathfrak{P}[\textsc{IsValid}](\mathtt{sid}, \mathcal{G}^+, S, \vec{V}, \mathtt{cmd}, \text{Acks}) = 1$, $\mathtt{id}$ 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 $E_{\mathtt{id}} = E \cup (\text{Acks} \times \{\mathtt{id}\})$, 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 $S_2 \coloneqq S'$. □

C.1.13 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 $\text{SessionGraphs}_{\text{Global}}[\mathtt{sid}]$ 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 $\text{SessionGraphs}_P = \emptyset$, 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 $\text{SessionGraphs}_P[\mathtt{sid}]$. 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 $\text{SessionGraphs}_P[\mathtt{sid}]$ is still proper. □

C.1.14 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.2 Proof of Corollary 1

$$\begin{aligned}
\mathbf{AuthChatSessions}[\mathfrak{P}] &= \perp^{\textsf{Auth-Intf}} \cdot \mathbf{ChatSessions}[\mathfrak{P}] && (1)\\
&\equiv \perp^{\textsf{Auth-Intf}} \cdot (\textsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\textsf{Net} \cdot \mathbf{REP})) && (2)\\
&\equiv \textsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\perp^{\textsf{Auth-Intf}} \cdot \textsf{Net} \cdot \mathbf{REP}) && (3)\\
&\equiv \textsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\textsf{Net} \cdot \perp^{\textsf{Auth-Intf}} \cdot \mathbf{REP}) && (4)\\
&= \mathbf{R_{Auth}}[\mathfrak{P}]. && (5)
\end{aligned}$$

(1): Definition of $\mathbf{AuthChatSessions}[\mathfrak{P}]$ (Equation 4.3);
(2): Theorem 1;
(3): Commutativity of converter application at disjoint interfaces ([27, Proposition 1]);
(4): By Equation C.2 (stated and proven below);
(5): Definition of $\mathbf{R}[\mathfrak{P}]$ (Equation 4.4).

It only remains to prove

$$\perp^{\textsf{Auth-Intf}} \cdot \textsf{Net} \cdot \mathbf{REP} \equiv \textsf{Net} \cdot \perp^{\textsf{Auth-Intf}} \cdot \mathbf{REP}. \tag{C.2}$$

Converter $\perp$ disables the interfaces it is attached to; in the case above these interfaces are $\textsf{Auth-Intf} \coloneqq \overline{\mathcal{P}^H}\text{-}\textsc{Write}(\langle \mathcal{S}^H \to \mathcal{R}^+\rangle, \cdot)$. So by attaching $\perp^{\textsf{Auth-Intf}}$ to $\textsf{Net} \cdot \mathbf{REP}$, we are simply disallowing parties to issue $\textsc{Write}$ operations for labels $\langle S \to \vec{V}\rangle$ where S is an honest party (i.e. $S \in \mathcal{S}^H$). However, note that the definition of converter $\textsf{Net}$ depends on the repositories to which it connects to (see Algorithm 3); in particular it only allows a party P to issue a $\textsc{Write}$ operation for a repository $\mathbf{rep_i} \coloneqq \mathbf{rep_i}{}^{\mathcal{W}_i}_{\mathcal{R}_i}$ if $P \in \mathcal{W}_i$, i.e. if P has write permissions—because the description of $\textsf{Net}$ specifies that the party's interface of $\textsf{Net}$ at which the $\textsc{Write}$ operation was issued matches the one that $\textsf{Net}$ uses to issue the corresponding $\textsc{Write}$ operation to the repository. This then implies Equation C.2. □

C.3 Proof of Corollary 2

To begin, note that by the definitions of **FakeChatSessions** (Algorithm 8), $\textsf{ChatSessionsForgeProt}$ (Algorithm 9) and $\left[\langle[\text{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\mathcal{P}^H}\right]_{P\in\mathcal{M},\vec{V}\in\mathcal{M}^+}$ (Algorithm 2), we have

$$\mathbf{FakeChatSessions} \equiv \textsf{ChatSessionsForgeProt}^{\mathcal{M}} \cdot \left[\langle[\text{Forge}]P \to \vec{V}\rangle^{\mathcal{M}}_{\mathcal{P}^H}\right]_{P\in\mathcal{M},\vec{V}\in\mathcal{M}^+}. \tag{C.3}$$

It then follows

$$\mathbf{OTR\text{-}ChatSessions}[\mathfrak{P}] = \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathbf{AuthChatSessions}[\mathfrak{P}] \\ \mathbf{FakeChatSessions} \end{bmatrix}. \tag{1}$$

$$\equiv \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathbf{R}_{\mathbf{Auth}}[\mathfrak{P}] \\ \mathbf{FakeChatSessions} \end{bmatrix} \tag{2}$$

$$\equiv \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP}) \\ \mathbf{FakeChatSessions} \end{bmatrix} \tag{3}$$

$$\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \mathbf{FakeChatSessions} \end{bmatrix} \tag{4}$$

$$\equiv \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \cdot \left[\langle [\mathrm{Forge}] P \to \vec{V} \rangle_{\overline{\mathcal{P}^H}}^{\mathcal{M}}\right]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+} \end{bmatrix} \tag{5}$$

$$\equiv \begin{pmatrix} \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \\ \cdot\, \mathsf{ChatSessionsForgeProt}^{\mathcal{M}} \end{pmatrix} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \bot^{\mathsf{Auth\text{-}Intf}} \cdot \mathbf{REP} \\ \left[\langle [\mathrm{Forge}] P \to \vec{V} \rangle_{\overline{\mathcal{P}^H}}^{\mathcal{M}}\right]_{P \in \mathcal{M}, \vec{V} \in \mathcal{M}^+} \end{bmatrix} \tag{6}$$

$$= \mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}]. \tag{7}$$

(1): Definition of **OTR-ChatSessions**$[\mathfrak{P}]$ (Equation 4.5);
(2): Corollary 1;
(3): Definition of $\mathbf{R}_{\mathbf{Auth}}[\mathfrak{P}]$ (Equation 4.4);
(4): Commutativity of converter application at disjoint interfaces ([27, Proposition 1]);
(5): By Equation C.3;
(6): Commutativity of converter application at disjoint interfaces ([27, Proposition 1]);
(7): Definition of $\mathbf{R}_{\mathsf{OTR}}[\mathfrak{P}]$ (Equation 4.6).

□

C.4 Proof of Corollary 3

$$\mathbf{R}_{\mathsf{ConfAnon}}[\mathfrak{P}] = \mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot (\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathbf{AR}[\mathfrak{P}]) \tag{1}$$

$$\equiv \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot (\mathsf{ChatSessionsProt}[\mathfrak{P}]^{\mathcal{M}^H} \cdot \mathbf{AR}[\mathfrak{P}]) \tag{2}$$

$$\equiv \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot (\mathbf{V}[\mathfrak{P}]). \tag{3}$$

(1): Definition of $\mathbf{R}_{\mathsf{ConfAnon}}[\mathfrak{P}]$;

(2): Commutativity of converter application at disjoint interfaces ([27, Proposition 1]);
(3): Assumption stated in Equation 4.7.

□

D UatChat

D.1 Definition of UatChatProt

For completeness, in Algorithm 29 we formalize the UatChatProt converter.

Algorithm 29 Description of the UatChatProt converter run by each honest party $P \in \mathcal{M}^H$ for constructing **UatChat** (see Algorithm 12). We rely on the helper functions from Algorithm 13.

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

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

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

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

    $\textsc{Read}$
    $\text{ChatGraphs} \leftarrow \emptyset$
    for $(\texttt{cid}, \mathcal{G}^+) \in \textbf{ChatSessions}[\mathfrak{U}]\text{-}\textsc{Read}$ with $\textsf{VisibleGraph}(\texttt{cid}, \mathcal{G}^+, P) \neq (\emptyset, \emptyset)$ :
        $\text{ChatGraphs} \leftarrow \text{ChatGraphs} \cup \{(\texttt{cid}, \textsf{VisibleGraph}(\texttt{cid}, \mathcal{G}^+, P))\}$
    $\textsc{Output}(\text{ChatGraphs})$
```

E Game-Based Security Definitions

In this section we introduce game-based notions that we use to prove the security of Maurer et al.'s MDRS-PKE construction [35]. We only introduce notions that are strictly necessary for such security proofs.

E.1 One Way Function

A One Way Function (OWF) is a pair $\Pi = (S, F)$, where S is a PPT and F a PT.

Consider an OWF $\Pi = (S, F)$; the game system of Definition 6 has (an implicitly defined) security parameter k and provides adversaries with access to oracles $\mathcal{O}_Y$ and $\mathcal{O}_S$ defined below:

Image Generation Oracle: $\mathcal{O}_Y(i \in \mathbb{N})$
1. On the first call on index $i \in \mathbb{N}$, compute $x \leftarrow S(1^k)$ and store $(i, x, y \coloneqq F(x))$; output y;
2. On subsequent calls, simply output y.

Submission Oracle: $\mathcal{O}_S(i \in \mathbb{N}, x)$
1. On the first call on i (to either this oracle or to $\mathcal{O}_Y$), compute $x \leftarrow S(1^k)$ and store $(i, x, y \coloneqq F(x))$; the oracle does not give any output;
2. On subsequent calls, the oracle simply does not perform any action nor give any output.

Definition 6. *Game* $\mathbf{G}^{\text{OWF}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_Y$ *and* $\mathcal{O}_S$*.* $\mathbf{A}$ *wins if it makes a query* $\mathcal{O}_S(i, x)$ *such that* $F(x) = \mathcal{O}_Y(i)$*.*

$\mathbf{A}$*'s winning advantage is defined as* $\mathbf{A}$*'s probability of winning the game.*

An adversary $\mathbf{A}$ (ε, t)-breaks the (n)-One-Wayness of OWF Π if it runs in time t, queries oracles $\mathcal{O}_Y$ and $\mathcal{O}_S$ on at most n different indices $i \in \mathbb{N}$, and satisfies $Adv^{\text{OWF}}(\mathbf{A}) \geq \varepsilon$.

E.2 Public Key Encryption

A Public Key Encryption (PKE) scheme with message space $\mathcal{M}$ is a triple of PPTs $\Pi = (G, E, D)$. Below we state the Correctness notion from [20].

Let $\Pi = (G, E, D)$ be a PKE scheme with message space $\mathcal{M}$; as before, we assume the game system of the following definition has (an implicitly defined) security parameter k. Definition 7 provides adversaries with access to the following oracles:

Secret Key Generation Oracle: $\mathcal{O}_{SK}(B_j)$
1. On the first call on B_j, compute and store $(\mathtt{pk}_j, \mathtt{sk}_j) \leftarrow G(1^k)$; output $(\mathtt{pk}_j, \mathtt{sk}_j)$;
2. On subsequent calls, simply output $(\mathtt{pk}_j, \mathtt{sk}_j)$.

Encryption Oracle: $\mathcal{O}_E(B_j, m; r)$
1. If r is given as input, encrypt m under $\mathtt{pk}_j$ (B_j's public key, as generated by $\mathcal{O}_{PK}$) using r as random tape; if r is not given as input create a fresh encryption of m under $\mathtt{pk}_j$;

2. Output the resulting ciphertext back to the adversary.

Decryption Oracle: $\mathcal{O}_D(B_j, c)$

1. Decrypt c using $\mathtt{sk}_j$ (B_j's secret key, as generated by $\mathcal{O}_{PK}$);
2. Output the resulting plaintext back to the adversary (or $\perp$ if decryption failed).

Definition 7. *Game $\mathbf{G}^{\mathsf{Corr}}$ provides an adversary $\mathbf{A}$ with access to oracles $\mathcal{O}_{SK}$, $\mathcal{O}_E$ and $\mathcal{O}_D$. $\mathbf{A}$ wins the game if there are two queries q_E and q_D to $\mathcal{O}_E$ and $\mathcal{O}_D$, respectively, where q_E has input $(B_j, m; r)$ and q_D has input $({B_j}', c)$, the input c in q_D is the output of q_E, $B_j = {B_j}'$, and the output of q_D is not m.*

The advantage of $\mathbf{A}$ in winning the Correctness game, denoted $Adv^{\mathsf{Corr}}(\mathbf{A})$, is the probability that $\mathbf{A}$ wins game $\mathbf{G}^{\mathsf{Corr}}$ as described above.

A (computationally unbounded) adversary $\mathbf{A}$ (ε)-breaks the (n)-Correctness of a PKE scheme Π if $\mathbf{A}$ queries $\mathcal{O}_{SK}$, $\mathcal{O}_E$ and $\mathcal{O}_D$ on at most n different parties and satisfies $Adv^{\mathsf{Corr}}(\mathbf{A}) \geq \varepsilon$.

E.3 Digital Signature Scheme

A Digital Signature Scheme (DSS) for a message space $\mathcal{M}$ is a triple $\Pi = (G, Sig, Vfy)$ of PPTs. Below we state the definition of (One-Time) Strong Existential Unforgeability for DSS. The notion has an implicitly defined security parameter k and makes use of oracles $\mathcal{O}_{VK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$, which, for a DSS $\Pi = (G, Sig, Vfy)$, are defined as:

Key-Pair Generation Oracle: $\mathcal{O}_{VK}(i \in \mathbb{N})$

1. On the first query on i, compute and store $(\mathtt{vk}_i, \mathtt{sk}_i) \leftarrow G(1^k)$;
2. Output $\mathtt{vk}_i$.

Signing Oracle: $\mathcal{O}_S(i, m)$

1. Compute $\sigma \leftarrow Sig_{\mathtt{sk}_i}(m)$, where $\mathtt{sk}_i$ is the signing key associated with i; output σ.

Verification Oracle: $\mathcal{O}_V(i, m, \sigma)$

1. Compute $d \leftarrow Vfy_{\mathtt{vk}_i}(m, \sigma)$, where $\mathtt{vk}_i$ is the verification key associated with i; output d.

Definition 8. *Game $\mathbf{G}^{\text{1-}\mathsf{sEUF\text{-}CMA}}$ provides an adversary with access to oracles $\mathcal{O}_{VK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$. $\mathbf{A}$ wins the game if there is a query to $\mathcal{O}_V$ on some input (i^*, m^*, σ^*) that outputs 1, there is no query to $\mathcal{O}_S$ on input (i^*, m^*) that output σ^*, and for each $i \in \mathbb{N}$ there is only at most one query to $\mathcal{O}_S$ with input i.*

$\mathbf{A}$'s winning advantage is $Adv^{\text{1-}\mathsf{sEUF\text{-}CMA}}(\mathbf{A}) \coloneqq \Pr[\mathbf{A}\mathbf{G}^{\text{1-}\mathsf{sEUF\text{-}CMA}} = \mathtt{win}]$.

An adversary $\mathbf{A}$ $(\varepsilon_{\text{1-}\mathsf{sEUF\text{-}CMA}}, t)$-breaks the (n, q_S, q_V)-1-sEUF-CMA security of Π if $\mathbf{A}$ runs in time at most t, queries $\mathcal{O}_{VK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$ on at most n different indices, makes at most q_S and q_V queries to, respectively, $\mathcal{O}_S$ and $\mathcal{O}_V$, and satisfies $Adv^{\text{1-}\mathsf{sEUF\text{-}CMA}}(\mathbf{A}) \geq \varepsilon_{\text{1-}\mathsf{sEUF\text{-}CMA}}$.

E.4 Multi-Designated Verifier Signature

In this section we present the Consistency, Unforgeability and Message-Bound Validity notions from [20].

Let $\mathit{\Pi} = (S, G_S, G_V, \mathit{Sig}, \mathit{Vfy}, \mathit{Forge})$ be an MDVS scheme. The security games below have an implicitly defined security parameter k and provide adversaries with access to the following oracles:

Public Parameter Generation Oracle: $\mathcal{O}_{PP}$

1. On the first call to $\mathcal{O}_{PP}$, compute $\mathtt{pp} \leftarrow S(1^k)$; output $\mathtt{pp}$;
2. On subsequent calls, simply output $\mathtt{pp}$.

Signer Key-Pair Generation Oracle: $\mathcal{O}_{SK}(A_i)$

1. On the first call to $\mathcal{O}_{SK}$ on input A_i, compute $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow G_S(\mathtt{pp})$, and output $(\mathtt{spk}_i, \mathtt{ssk}_i)$;
2. On subsequent calls, simply output $(\mathtt{spk}_i, \mathtt{ssk}_i)$.

Verifier Key-Pair Generation Oracle: $\mathcal{O}_{VK}(B_j)$

1. Analogous to the Signer Key-Pair Generation Oracle.

Signer Public-Key Oracle: $\mathcal{O}_{SPK}(A_i)$

1. $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow \mathcal{O}_{SK}(A_i)$; output $\mathtt{spk}_i$.

Verifier Public-Key Oracle: $\mathcal{O}_{VPK}(B_j)$

1. Analogous to the Signer Public-Key Oracle.

Signing Oracle: $\mathcal{O}_S(A_i, \vec{V}, m)$

1. $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow \mathcal{O}_{SK}(A_i)$;
2. $\vec{v} = (\mathcal{O}_{VPK}(V_1), \ldots, \mathcal{O}_{VPK}(V_{|\vec{V}|}))$;
3. Output $\sigma \leftarrow \mathit{Sig}_{\mathtt{pp}}(\mathtt{ssk}_i, \vec{v}, m)$.

Verification Oracle: $\mathcal{O}_V(A_i, B_j \in \mathrm{Set}(\vec{V}), \vec{V}, m, \sigma)$

1. $\mathtt{spk}_i \leftarrow \mathcal{O}_{SPK}(A_i)$;
2. $\vec{v} = (\mathcal{O}_{VPK}(V_1), \ldots, \mathcal{O}_{VPK}(V_{|\vec{V}|}))$;
3. $(\mathtt{vpk}_j, \mathtt{vsk}_j) \leftarrow \mathcal{O}_{VK}(B_j)$;
4. output $d \leftarrow \mathit{Vfy}_{\mathtt{pp}}(\mathtt{spk}_i, \mathtt{vsk}_j, \vec{v}, m, \sigma)$, where $d \in \{0, 1\}$.

Definition 9 (Consistency). *Game system $\mathbf{G}^{\mathsf{Cons}}$ provides an adversary $\mathbf{A}$ with access to oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{VK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{VPK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$. $\mathbf{A}$ wins if it makes two queries $\mathcal{O}_V(A_i, B_j, \vec{V}, m, \sigma)$ and $\mathcal{O}_V(A_i{}', B_j{}', \vec{V}', m', \sigma')$ such that $(A_i, \vec{V}, m, \sigma) = (A_i{}', \vec{V}', m', \sigma')$, $\{B_j, B_j{}'\} \subseteq \vec{V}$, the outputs of the two queries differ, and there is no query $\mathcal{O}_{VK}(B_j)$ prior to query $\mathcal{O}_V(A_i, B_j, \vec{V}, m, \sigma)$, and no query $\mathcal{O}_{VK}(B_j{}')$ prior to query $\mathcal{O}_V(A_i{}', B_j{}', \vec{V}', m', \sigma')$.*

The advantage of $\mathbf{A}$ in winning the Consistency game is the probability that $\mathbf{A}$ wins game $\mathbf{G}^{\mathsf{Cons}}$ as described above, and is denoted $\mathit{Adv}^{\mathsf{Cons}}(\mathbf{A})$.

Definition 10 (Unforgeability). *$\mathbf{G}^{\mathsf{Unforg}}$ provides an adversary $\mathbf{A}$ with access to oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{VK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{VPK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$. $\mathbf{A}$ wins if it makes a query $\mathcal{O}_V(A_i{}^*, B_j{}^*, \vec{V}^*, m^*, \sigma^*)$ with $B_j{}^* \in \vec{V}^*$ that outputs 1, for every query $\mathcal{O}_S(A_i{}', \vec{V}', m')$, $(A_i{}^*, \vec{V}^*, m^*) \neq (A_i{}', \vec{V}', m')$ and there is no $\mathcal{O}_{SK}$ query on $A_i{}^*$ nor $\mathcal{O}_{VK}$ query on $B_j{}^*$. $\mathbf{A}$'s advantage is the probability that $\mathbf{A}$ wins $\mathbf{G}^{\mathsf{Unforg}}$, and is denoted $\mathit{Adv}^{\mathsf{Unforg}}(\mathbf{A})$.*

Definition 11 (Message-Bound Validity). *Game* $\mathbf{G}^{\textsf{Bound-Val}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$,$\mathcal{O}_{SK}$,$\mathcal{O}_{VK}$,$\mathcal{O}_{SPK}$,$\mathcal{O}_{VPK}$,$\mathcal{O}_S$, *and* $\mathcal{O}_V$. $\mathbf{A}$ *wins the game if there are two queries* q_S *and* q_V *to* $\mathcal{O}_S$ *and* $\mathcal{O}_V$, *respectively, where* q_S *has input* $(A_i, \vec{V}, m)$ *and* q_V *has input* $({A_i}', B_j, \vec{V}', m', \sigma)$, *satisfying 1.* $(A_i, \vec{V}) = ({A_i}', \vec{V}')$; *2.* $B_j \in \vec{V}$; *3.* $m \neq m'$; *4. the input* σ *in* q_V *is* $\mathcal{O}_S$*'s output on query* q_S; *and 5. the output of* $\mathcal{O}_V$ *on query* q_V *is* 1. $\mathbf{A}$*'s advantage is the probability that* $\mathbf{A}$ *wins* $\mathbf{G}^{\textsf{Bound-Val}}$, *and is denoted* $Adv^{\textsf{Bound-Val}}(\mathbf{A})$.

We say an adversary $\mathbf{A}$ (ε, t)-breaks the $(n_S, n_V, d_S, q_S, q_V)$-Consistency (resp. -Unforgeability, -Message-Bound Validity) of $\mathit{\Pi}$ if $\mathbf{A}$ runs in time at most t, queries $\mathcal{O}_{SK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$ on at most n_S different signers, $\mathcal{O}_{VK}$, $\mathcal{O}_{VPK}$, $\mathcal{O}_S$ and $\mathcal{O}_V$ on at most n_V different verifiers, makes at most q_S and q_V queries to $\mathcal{O}_S$ and $\mathcal{O}_V$, respectively, with the sum of the verifier vectors' lengths input to $\mathcal{O}_S$ being at most d_S, and satisfies $Adv^{\textsf{Cons}}(\mathbf{A}) \geq \varepsilon$ (resp. $Adv^{\textsf{Unforg}}(\mathbf{A}) \geq \varepsilon$, $Adv^{\textsf{Bound-Val}}(\mathbf{A}) \geq \varepsilon$).

E.5 Public Key Encryption for Broadcast

A Public Key Encryption for Broadcast (PKEBC) scheme $\mathit{\Pi}$ with message space $\mathcal{M}$ is a quadruple $\mathit{\Pi} = (S, G, E, D)$ of PPTs. Below we state the Correctness, Robustness, Consistency and $\{\textsf{IND}, \textsf{IK}\}$-$\textsf{CCA-2}$ security notions from [20].

Consider a PKEBC $\mathit{\Pi} = (S, G, E, D)$ with message space $\mathcal{M}$. The game systems defined by the security notions ahead have an implicitly defined security parameter k and provide adversaries with access to the following oracles:

Public Parameters Oracle: $\mathcal{O}_{PP}$

1. On the first call, compute and store $\texttt{pp} \leftarrow S(1^k)$; output $\texttt{pp}$;
2. On subsequent calls, output the previously generated $\texttt{pp}$.

Secret Key Generation Oracle: $\mathcal{O}_{SK}(B_j)$

1. If $\mathcal{O}_{SK}$ was queried on B_j before, simply look up and return the previously generated key for B_j;
2. Otherwise, store $(\texttt{pk}_j, \texttt{sk}_j) \leftarrow G(\texttt{pp})$ as B_j's key-pair, and output $(\texttt{pk}_j, \texttt{sk}_j)$.

Public Key Generation Oracle: $\mathcal{O}_{PK}(B_j)$

1. $(\texttt{pk}_j, \texttt{sk}_j) \leftarrow \mathcal{O}_{SK}(B_j)$;
2. Output $\texttt{pk}_j$.

Encryption Oracle: $\mathcal{O}_E(\vec{V}, m)$

1. $\vec{v} \leftarrow (\mathcal{O}_{PK}(V_1), \ldots, \mathcal{O}_{PK}(V_{|\vec{V}|}))$;
2. Create and output a fresh encryption $c \leftarrow E_{\texttt{pp},\vec{v}}(m)$.

Decryption Oracle: $\mathcal{O}_D(B_j, c)$

1. Query $\mathcal{O}_{SK}(B_j)$ to obtain the corresponding secret-key $\texttt{sk}_j$;
2. Decrypt c using $\texttt{sk}_j$, $(\vec{v}, m) \leftarrow D_{\texttt{pp},\texttt{sk}_j}(c)$, and then output the resulting receivers-message pair $(\vec{v}, m)$, or $\perp$ (if $(\vec{v}, m) = \perp$, i.e. the ciphertext is not valid with respect to B_j's secret key).

Definition 12 (Correctness). *Game* $\mathbf{G}^{\mathsf{Corr}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if there are two queries* q_E *and* q_D *to* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*, respectively, where* q_E *has input* $(\vec{V}, m)$ *and* q_D *has input* (B_j, c)*, satisfying* $B_j \in \vec{V}$*, the input* c *in* q_D *is the output of* q_E*, and the output of* q_D *is either* $\perp$ *or* $(\vec{v}', m')$ *with* $(\vec{v}, m) \neq (\vec{v}', m')$*.*

The advantage of $\mathbf{A}$ *in winning the Correctness game is the probability that* $\mathbf{A}$ *wins game* $\mathbf{G}^{\mathsf{Corr}}$ *as described above, and is denoted* $Adv^{\mathsf{Corr}}(\mathbf{A})$*.*

Definition 13 (Robustness). *Game* $\mathbf{G}^{\mathsf{Rob}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_{PK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if there are two queries* q_E *and* q_D *to* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*, respectively, where* q_E *has input* $(\vec{V}, m)$ *and* q_D *has input* (B_j, c)*, satisfying* $B_j \notin \vec{V}$*, the input* c *in* q_D *is the output of* q_E*, and the output of* q_D *is* $(\vec{v}', m')$ *with* $(\vec{v}', m') \neq \perp$*. The advantage of* $\mathbf{A}$ *in winning the Robustness game is the probability that* $\mathbf{A}$ *wins game* $\mathbf{G}^{\mathsf{Rob}}$ *as described above, and is denoted* $Adv^{\mathsf{Rob}}(\mathbf{A})$*.*

Definition 14 (Consistency). $\mathbf{G}^{\mathsf{Cons}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if there are two queries* $\mathcal{O}_D(B_i, c)$ *and* $\mathcal{O}_D(B_j, c)$ *for some* B_i *and* B_j *(possibly with* $B_i = B_j$*) on the same ciphertext* c *such that query* $\mathcal{O}_D(B_i, c)$ *outputs some pair* $(\vec{v}, m) \neq \perp$ *with* $\mathtt{pk}_j \in \vec{v}$ *(where* $\mathtt{pk}_j$ *is* B_j*'s public key), and query* $\mathcal{O}_D(B_j, c)$ *does not output* $(\vec{v}, m)$*.*

$\mathbf{A}$*'s advantage in winning the Consistency game is denoted* $Adv^{\mathsf{Cons}}(\mathbf{A})$ *and corresponds to the probability that* $\mathbf{A}$ *wins game* $\mathbf{G}^{\mathsf{Cons}}$*.*

Below we present the definition of $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ security from [19]. The games defined by this definition provide adversaries with access to the oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$ and $\mathcal{O}_{PK}$ defined above, as well as to oracles $\mathcal{O}_E$ and $\mathcal{O}_D$ defined below:

Encryption Oracle: $\mathcal{O}_E\big((\vec{V}_0, m_0), (\vec{V}_1, m_1)\big)$

1. For game system $\mathbf{G}_{\mathbf{b}}^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}}$, encrypt $m_{\mathbf{b}}$ under $\vec{v}_{\mathbf{b}}$, the vector of public keys corresponding to $\vec{V}_{\mathbf{b}}$; output c.

Decryption Oracle: $\mathcal{O}_D(B_j, c)$

1. If c was the output of some query to $\mathcal{O}_E$, output `test`;
2. Otherwise, compute and output $(\vec{v}, m) \leftarrow D_{\mathtt{pp}, \mathtt{sk}_j}(c)$, where $\mathtt{sk}_j$ is B_j's secret key.

Definition 15 ($\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ Security). *For* $\mathbf{b} \in \{0, 1\}$*, game system* $\mathbf{G}_{\mathbf{b}}^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_{PK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if it outputs a guess bit* b' *satisfying* $b' = \mathbf{b}$ *and for every query* $\mathcal{O}_E\big((\vec{V}_0, m_0), (\vec{V}_1, m_1)\big)$*: 1.* $|\vec{V}_0| = |\vec{V}_1|$*; 2.* $|m_0| = |m_1|$*; and 3. there is no query to* $\mathcal{O}_{SK}$ *on any* $B_j \in Set(\vec{V}_0) \cup Set(\vec{V}_1)$ *at any point during the game. We define the advantage of* $\mathbf{A}$ *in winning the* $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA}$-$\mathsf{2}$ *game as*

$$Adv^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}}(\mathbf{A}) :=$$
$$\Big|\Pr[\mathbf{A}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}} = \mathtt{win}] + \Pr[\mathbf{A}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}} = \mathtt{win}] - 1\Big|.$$

We say an adversary $\mathbf{A}$ (ε, t)-breaks the (n, d_E, q_E, q_D)-Correctness (resp. -Robustness, -Consistency, -{IND, IK}-CCA-2 security) of a PKEBC scheme Π if $\mathbf{A}$ runs in time at most t, queries $\mathcal{O}_{SK}$, $\mathcal{O}_E$ and $\mathcal{O}_D$ on at most n different parties, makes at most q_E and q_D queries to $\mathcal{O}_E$ and $\mathcal{O}_D$, respectively, with the sum of lengths of the party vectors input to $\mathcal{O}_E$ being at most d_E, and satisfies $Adv^{\mathsf{Corr}}(\mathbf{A}) \geq \varepsilon$ (resp. $Adv^{\mathsf{Rob}}(\mathbf{A}) \geq \varepsilon$, $Adv^{\mathsf{Cons}}(\mathbf{A}) \geq \varepsilon$, $Adv^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}\mathsf{2}}(\mathbf{A}) \geq \varepsilon$).

E.6 Multi-Designated Receiver Signed Public Key Encryption

In this section we introduce the (remaining) MDRS-PKE security notions from [19].[32] Let $\Pi = (S, G_S, G_R, E, D, \mathit{Forge})$ be an MDRS-PKE scheme with message space $\mathcal{M}$. The games ahead provide adversaries with access to the oracles defined in Section 6.2.

Definition 16 (Correctness). $\mathbf{G}^{\mathsf{Corr}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{RK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{RPK}$, $\mathcal{O}_E$ *and* $\mathcal{O}_D$. $\mathbf{A}$ *wins the game if there is a query* q_E *to* $\mathcal{O}_E$ *and a later query* q_D *to* $\mathcal{O}_D$ *such that* q_E *has input* $(A_i, \vec{V}, m)$ *and* q_D *has input* (B_j, c) *with* $B_j \in \vec{V}$ *and* c *being the output of* q_E*, the output of* q_D *is* $(\mathtt{spk}_i', \vec{v}', m')$ *with* $(\mathtt{spk}_i', \vec{v}', m') \neq (\mathtt{spk}_i, \vec{v}, m)$*—where* $\mathtt{spk}_i$ *is* A_i*'s public key and* $\vec{v}$ *is the vector of public keys corresponding to* $\vec{V}$.

The advantage of $\mathbf{A}$ *in winning the Correctness game is the probability that* $\mathbf{A}$ *wins game* $\mathbf{G}^{\mathsf{Corr}}$ *as described above, and is denoted* $Adv^{\mathsf{Corr}}(\mathbf{A})$.

The following Consistency notion slightly differs from the one given in [19]; it captures the additional property that if the decryption of a ciphertext c by a party B_j outputs some valid triple $(\mathtt{spk}, \vec{v}, m) \neq \bot$, then B_j's public key $\mathtt{rpk}_j$ must be part of the vector $\vec{v}$ output by decryption (i.e. $\mathtt{rpk}_j \in \vec{v}$).[33] This property is useful because it eliminates the need for receivers' converters to perform this additional check (see Algorithm 16).

Definition 17 (Consistency). *Game system* $\mathbf{G}^{\mathsf{Cons}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{RK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{RPK}$, $\mathcal{O}_E$ *and* $\mathcal{O}_D$. $\mathbf{A}$ *wins the game if (at least) one of the following events* (ξ_1 *or* ξ_2) *occurs:*

Event ξ_1: *there is a query* $\mathcal{O}_D(B_i, c)$ *that outputs some triple* $(\mathtt{spk}, \vec{v}, m)$ *with* $(\mathtt{spk}, \vec{v}, m) \neq \bot$ *and* $\mathtt{pk}_i \notin \vec{v}$*, where* $\mathtt{pk}_i$ *is* B_i*'s public key;*

[32] Some of the notions we define are stronger than the original ones from [19]; we present them because we rely on the additional properties in our composable treatment of MDRS-PKE schemes. Note that we do prove (for completeness) that Maurer et al.'s MDRS-PKE scheme (when instantiated with Chakraborty et al.'s MDVS and PKEBC schemes yields a scheme that) is secure with respect to all these strengthened security notions.

[33] Maurer et al.'s MDRS-PKE construction [35] trivially satisfies this modified notion as the decryption algorithm explicitly checks if the receiver's public key is in the vector to be output.

Event ξ_2: *there are two $\mathcal{O}_D$ queries, say q_{D_i} and q_{D_j}, on inputs, respectively, (B_i, c) and (B_j, c') with $c = c'$ such that: 1. the outputs of q_{D_i} and q_{D_j} differ; 2. either the receiver public key $\texttt{rpk}_j$ of B_j is part of the vector of receiver public keys output by q_{D_i}, or the receiver public key $\texttt{rpk}_i$ of B_i is part of the vector of public keys output by q_{D_j}; 3. there is no query $\mathcal{O}_{RK}(B_i)$ (resp. $\mathcal{O}_{RK}(B_j)$) prior to q_{D_i} (resp. q_{D_j}); and 4. there is no sender A (resp. no receiver B) which had not been input to a query $\mathcal{O}_{SPK}$, $\mathcal{O}_{SK}$ or $\mathcal{O}_E$ (resp. $\mathcal{O}_{RPK}$, $\mathcal{O}_{RK}$ or $\mathcal{O}_E$) prior to both q_{D_i} and q_{D_j} and whose public key is output by one of these queries.*

The advantage of $\mathbf{A}$ in winning the Consistency game corresponds to the probability that $\mathbf{A}$ wins game $\mathbf{G}^{\mathsf{Cons}}$ as described above and is denoted $Adv^{\mathsf{Cons}}(\mathbf{A})$.

The following notion strengthens the one introduced by Maurer et al. [35] by capturing (the infeasibility of) replay attacks.

Definition 18 (Replay Unforgeability). *Game $\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ provides an adversary $\mathbf{A}$ with access to oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{RK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{RPK}$, $\mathcal{O}_E$ and $\mathcal{O}_D$. $\mathbf{A}$ wins if it makes a query $\mathcal{O}_D(B_j, c)$ that outputs $(\texttt{spk}_i, \vec{v}, m) \neq \perp$, there is a sender A_i and a vector of receivers $\vec{V}$ such that $\texttt{spk}_i$ is A_i's sender public key (i.e. $\mathcal{O}_{SPK}(A_i) = \texttt{spk}_i$) and $\vec{v}$ is the vector of receiver public keys corresponding to $\vec{V}$ (i.e. $|\vec{V}| = |\vec{v}|$ and for each $l \in \{1, \ldots, |\vec{v}|\}$, $\mathcal{O}_{RPK}(V_l) = v_l$), there was no query $\mathcal{O}_E(A_i{}', \vec{V}', m')$ with $(A_i, \vec{V}, m) = (A_i{}', \vec{V}', m')$ that output the same ciphertext c that was input to $\mathcal{O}_D$, and neither $\mathcal{O}_{SK}$ was queried on input A_i nor $\mathcal{O}_{RK}$ was queried on input B_j.*

The advantage of $\mathbf{A}$ in winning the Unforgeability against Replays game is the probability that $\mathbf{A}$ wins game $\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ as described above, and is denoted $Adv^{\mathsf{R\text{-}Unforg}}(\mathbf{A})$.

The $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA\text{-}2}$ security notion below is a slight modification of the original one introduced by Maurer et al. [35] in that it allows adversaries to obtain encryptions of messages by honest senders to vectors of receivers who are not all honest. (Concretely, adversaries are allowed to query the encryption oracle even if some of the receivers are dishonest, though in this case it is required that the challenge inputs $(A_{i,0}, \vec{V_0}, m_0)$ and $(A_{i,1}, \vec{V_1}, m_1)$ are the same, i.e. $(A_{i,0}, \vec{V_0}, m_0) = (A_{i,1}, \vec{V_1}, m_1)$.) The $\{\mathsf{IND}, \mathsf{IK}\}$-$\mathsf{CCA\text{-}2}$ security games provide adversaries with access to the oracles defined in Section 6.2 and to (modified) oracles $\mathcal{O}_E$ and $\mathcal{O}_D$:

Encryption Oracle: $\mathcal{O}_E\big((A_{i,0}, \vec{V_0}, m_0), (A_{i,1}, \vec{V_1}, m_1)\big)$

1. For game system $\mathbf{G}_{\mathbf{b}}^{\{\mathsf{IND}, \mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$, encrypt $m_{\mathbf{b}}$ under $\texttt{ssk}_{i,\mathbf{b}}$ ($A_{i,\mathbf{b}}$'s sender secret key) and $\vec{v_{\mathbf{b}}}$ ($\vec{V_{\mathbf{b}}}$'s corresponding vector of receiver public keys); output c.

Decryption Oracle: $\mathcal{O}_D(B_j, c)$

1. If c was the output of some query to $\mathcal{O}_E$, output $\texttt{test}$;
2. Otherwise, proceed as in the default $\mathcal{O}_D$ oracle.

Definition 19 ($\{\mathsf{IND},\mathsf{IK}\}$-CCA-2 Security). *For bit* $\mathbf{b} \in \{0,1\}$*, game* $\mathbf{G}_{\mathbf{b}}^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_{RK}$*,* $\mathcal{O}_{SPK}$*,* $\mathcal{O}_{RPK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if it outputs a guess bit* b' *with* $b' = \mathbf{b}$ *and for every query* $\mathcal{O}_E\big((A_{i,0},\vec{V}_0,m_0),(A_{i,1},\vec{V}_1,m_1)\big)$ *with* $(A_{i,0},\vec{V}_0,m_0) \neq (A_{i,1},\vec{V}_1,m_1)$*: 1.* $|m_0| = |m_1|$*; 2.* $|\vec{V}_0| = |\vec{V}_1|$*; and 3. there is no query to* $\mathcal{O}_{RK}$ *on any* $B_j \in Set(\vec{V}_0) \cup Set(\vec{V}_1)$ *at any point during the game.*

The advantage of $\mathbf{A}$ *in winning the* $\{\mathsf{IND},\mathsf{IK}\}$-$\mathsf{CCA}$-$2$ *games is*

$$Adv^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}(\mathbf{A}) := \Big|\Pr[\mathbf{A}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2} = \mathtt{win}] + \Pr[\mathbf{A}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2} = \mathtt{win}] - 1\Big|.$$

The following notion defines two game systems, $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$ and $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$, which provide adversaries with access to the oracles from before as well as to (modified) $\mathcal{O}_E$ and $\mathcal{O}_D$ oracles:

Encryption Oracle: $\mathcal{O}_E(\mathtt{type} \in \{\mathtt{sig},\mathtt{sim}\}, A_i, \vec{V}, m, \mathcal{C} \subseteq \mathrm{Set}(\vec{V}))$

For $\mathbf{b} \in \{0,1\}$, oracle $\mathcal{O}_E$ of game system $\mathbf{G}_{\mathbf{b}}^{\mathsf{OTR}}$ behaves as follows:

1. Let $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow \mathcal{O}_{SK}(A_i)$;
2. Let $\vec{v} = (v_1,\dots,v_{|\vec{V}|})$ and $\vec{s} = (s_1,\dots,s_{|\vec{V}|})$, where, for $i = 1,\dots,|\vec{V}|$:
   - $(v_i, s_i) = \begin{cases} \mathcal{O}_{RK}(V_i) & \text{if } V_i \in \mathcal{C} \\ (\mathcal{O}_{RPK}(V_i), \bot) & \text{otherwise;} \end{cases}$
3. $(c_{\mathbf{0}}, c_{\mathbf{1}}) \leftarrow (\mathit{\Pi}.E_{\mathtt{pp}}(\mathtt{ssk}_i, \vec{v}, m), \mathit{\Pi}.\mathit{Forge}_{\mathtt{pp}}(\mathtt{spk}_i, \vec{v}, m, \vec{s}))$;
4. If $\mathbf{b} = 0$ and $\mathtt{type} = \mathtt{sig}$ output $c_{\mathbf{0}}$; otherwise output $c_{\mathbf{1}}$.

Decryption Oracle: $\mathcal{O}_D(B_j, c)$

1. If c was the output of some query to $\mathcal{O}_E$, output $\mathtt{test}$;
2. Otherwise, proceed as in the default $\mathcal{O}_D$ oracle.

Definition 20 (Off-The-Record). *For* $\mathbf{b} \in \{0,1\}$*, game* $\mathbf{G}_{\mathbf{b}}^{\mathsf{OTR}}$ *provides an adversary* $\mathbf{A}$ *with access to oracles* $\mathcal{O}_{PP}$*,* $\mathcal{O}_{SK}$*,* $\mathcal{O}_{RK}$*,* $\mathcal{O}_{SPK}$*,* $\mathcal{O}_{RPK}$*,* $\mathcal{O}_E$ *and* $\mathcal{O}_D$*.* $\mathbf{A}$ *wins the game if it outputs a guess bit* b' *with* $b' = \mathbf{b}$ *and for every query* $\mathcal{O}_E(\mathtt{type}, A_i, \vec{V}, m, \mathcal{C})$ *and every query* $\mathcal{O}_{VK}(B_j)$*, we have* $B_j \notin Set(\vec{V}) \setminus \mathcal{C}$*.*

$\mathbf{A}$*'s advantage in winning the Off-The-Record game is*

$$Adv^{\mathsf{OTR}}(\mathbf{A}) := \Big|\Pr[\mathbf{A}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} = \mathtt{win}] + \Pr[\mathbf{A}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \mathtt{win}] - 1\Big|.$$

F New MDVS Security Notions and Security Analysis of Chakraborty et al.'s MDVS [19]

F.1 New MDVS Game-Based Notions

We now introduce new MDVS security notions (analogous to the MDRS-PKE ones given in Section 6.2).[34]

[34] As for MDRS-PKE, the (MDVS) Forgery Invalidity notion also seems necessary to imply (appropriately defined) composable semantics (for MDVS schemes) in the setting where the secret key of senders leak.

The security game defined by the notion below makes use of the oracles defined in Section E.4 plus the following new oracle:

Forgery Oracle: $\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C} \subseteq \mathrm{Set}(\vec{V}))$

1. $\mathtt{spk}_i \leftarrow \mathcal{O}_{SPK}(A_i)$;
2. for $i = 1, \ldots, |\vec{V}|$, let $(v_i, s_i) = \begin{cases} \mathcal{O}_{VK}(V_i) & \text{if } V_i \in \mathcal{C} \\ (\mathcal{O}_{VPK}(V_i), \perp) & \text{otherwise,} \end{cases}$;
3. output $\Pi.Forge_{\mathsf{pp}}(\mathtt{spk}_i, \vec{v}, m, \vec{s})$, where $\vec{v} = (v_1, \ldots, v_{|\vec{V}|})$ and $\vec{s} = (s_1, \ldots, s_{|\vec{V}|})$.

Definition 21 (Forgery Invalidity). *Game system $\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}$ provides an adversary $\mathbf{A}$ with access to oracles $\mathcal{O}_{PP}$, $\mathcal{O}_{SK}$, $\mathcal{O}_{VK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_{VPK}$, $\mathcal{O}_S$, $\mathcal{O}_V$ and $\mathcal{O}_{Forge}$. $\mathbf{A}$ wins the game if there are two queries q_{Forge} and q_V to $\mathcal{O}_{Forge}$ and $\mathcal{O}_V$, respectively, where q_{Forge} has input $(A_i, \vec{V}, m, \mathcal{C})$ and q_V has input $(A_i', B_j, \vec{V}', m', \sigma)$, satisfying 1. $(A_i, \vec{V}, m) = (A_i', \vec{V}', m')$; 2. $B_j \in \vec{V}$; 3. $B_j \notin \mathcal{C}$; 4. the input σ in q_V is the output of $\mathcal{O}_{Forge}$ on query q_{Forge}; and 5. the output of the oracle $\mathcal{O}_V$ on the query q_V is 1.*

$\mathbf{A}$'s advantage is the probability that $\mathbf{A}$ wins $\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}$, and is denoted $Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{A})$.

An adversary $\mathbf{A}$ (ε, t)-breaks the $(n_S, n_V, d_S, d_F, q_S, q_V, q_F)$-Forgery Invalidity of Π if $\mathbf{A}$ runs in time at most t, queries $\mathcal{O}_{SK}$, $\mathcal{O}_{SPK}$, $\mathcal{O}_S$, $\mathcal{O}_V$ and $\mathcal{O}_{Forge}$ on at most n_S different signers, $\mathcal{O}_{VK}$, $\mathcal{O}_{VPK}$, $\mathcal{O}_S$, $\mathcal{O}_V$ and $\mathcal{O}_{Forge}$ on at most n_V different verifiers, makes at most q_S, q_V and q_F queries to $\mathcal{O}_S$, $\mathcal{O}_V$ and $\mathcal{O}_{Forge}$, respectively, with the sum of the verifier vectors' lengths input to $\mathcal{O}_S$ and $\mathcal{O}_{Forge}$ being at most d_S and d_F, respectively, and satisfies $Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{A}) \geq \varepsilon$.

Definition 22 (Public-Key Collision Resistance). *An* MDVS *scheme $\Pi = (S, G_S, G_V, Sig, Vfy, Forge)$ is (n, ℓ)-Party ε-Public-Key Collision Resistant if*

$$\Pr\left[\begin{array}{c|c} \left|\{\mathtt{spk}_1, \ldots, \mathtt{spk}_n, & \mathtt{pp} \leftarrow S(1^k) \\ \mathtt{vpk}_1, \ldots, \mathtt{vpk}_\ell\}\right| & (\mathtt{spk}_1, \mathtt{ssk}_1) \leftarrow G_{S_{\mathsf{pp}}},\ (\mathtt{vpk}_1, \mathtt{vsk}_1) \leftarrow G_{V_{\mathsf{pp}}} \\ < n + \ell & \ldots \qquad\qquad \ldots \\ & (\mathtt{spk}_n, \mathtt{ssk}_n) \leftarrow G_{S_{\mathsf{pp}}},\ (\mathtt{vpk}_\ell, \mathtt{vsk}_\ell) \leftarrow G_{V_{\mathsf{pp}}} \end{array}\right] \leq \varepsilon.$$

F.2 Security of Chakraborty et al.'s MDVS Construction [19]

We prove the security of Chakraborty et al.'s MDVS construction [19]—denoted $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$ and defined in Algorithm 30—with respect to the new security notions. The building blocks are a Public Key Encryption scheme Π_{PKE}, a One Way Function Π_{OWF} and a Non Interactive Zero Knowledge Π_{NIZK}; the informal theorem below summarizes our results regarding $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$'s additional security guarantees.

Theorem 5 (Informal). *If Π_{PKE} is correct and Π_{OWF} is tightly multi-instance secure then $\Pi_{\mathrm{MDVS}}^{\mathsf{adap}}$ is tightly Forgery Invalidity secure (see Theorem 6) and is Public-Key Collision Resistant (see Corollary 5).*

As noted in [19], there are tightly secure and structure preserving instantiations of each of $\Pi_{\mathrm{MDVS}}^{\mathsf{adap}}$'s building blocks.

F.2.1 Forgery Invalidity

Theorem 6. *If no adversary* $\mathbf{A}$ $(\varepsilon_{\mathrm{PKE}})$*-breaks the* (n_{PKE})*-Correctness of* Π_{PKE} *then no (computationally unbounded) adversary* (ε)*-breaks* $\Pi_{\mathrm{MDVS}}^{\mathsf{adap}}$*'s*

$$(n_V := n_{\mathrm{PKE}})\text{-}\textit{Forgery Invalidity,}$$

with $\varepsilon > 2 \cdot \varepsilon_{\mathrm{PKE}}$.

Proof. This proof proceeds in a sequence of games [14, 41].

$\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} \rightsquigarrow \mathbf{G}^1$**:** $\mathbf{G}^1$ is just like the original game $\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}$, except that in $\mathbf{G}^1$ the Π_{PKE} key-pair $(\mathtt{pk}_0, \mathtt{sk}_0)$ sampled for each party B_j is assumed to be correct.

One can reduce distinguishing these two games to breaking Π_{PKE}'s correctness because the reduction holds all secret keys (and so it can handle any oracle queries). If an adversary $\mathbf{A}$ only queries for the verifier keys of up to $n_V \leq n_{\mathrm{PKE}}$ parties, and given the reduction only has to use Π_{PKE}-$\mathcal{O}_{SK}$ oracle to generate at most one key-pair per party—namely, $(\mathtt{pk}_0, \mathtt{sk}_0)$—since by assumption no computationally unbounded adversary $(\varepsilon_{\mathrm{PKE}})$-breaks the (n_{PKE})-Correctness of Π_{PKE}, it follows

$$\left|\Pr[\mathbf{A}\mathbf{G}^1 = \mathtt{win}] - \Pr[\mathbf{A}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \mathtt{win}]\right| \leq \varepsilon_{\mathrm{PKE}}.$$

$\mathbf{G}^1 \rightsquigarrow \mathbf{G}^2$**.** This game hop is just like the previous one (i.e. $\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} \rightsquigarrow \mathbf{G}^1$), the only difference being that the key-pair which is assumed to be a correct one is now $(\mathtt{pk}_1, \mathtt{sk}_1)$. It follows

$$\left|\Pr[\mathbf{A}\mathbf{G}^2 = \mathtt{win}] - \Pr[\mathbf{A}\mathbf{G}^1 = \mathtt{win}]\right| \leq \varepsilon_{\mathrm{PKE}}.$$

To finish the proof we now prove the following claim:

Claim. For any adversary $\mathbf{A}$

$$\Pr[\mathbf{A}\mathbf{G}^2 = \mathtt{win}] = 0.$$

Proof. Recall that an adversary can only win game $\mathbf{G}^2$ if it makes a query to $\mathcal{O}_V$ on some input $(A_i, B_j, \vec{V}, m, \sigma := (p, \vec{c}, c_{\mathsf{pp}}))$ such that σ was output by a query to $\mathcal{O}_{Forge}$ on input $(A_i, \vec{V}, m, \mathcal{C})$ satisfying $B_j \in \vec{V}$, $B_j \notin \mathcal{C}$, and $\mathcal{O}_V$ outputs 1. First, note that, by the definition of $\mathcal{O}_{Forge}$, this means that B_j's secret verifier key is not given as input when the oracle is forging the signature using algorithm *Forge* of $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$. Furthermore, by the definition of $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$'s *Forge* algorithm (Algorithm 30), it follows that for every $l \in \{1, \dots, |\vec{V}|\}$ such that $V_l = B_j$, the two ciphertexts in c_l (i.e. $c_{l,0}$ and $c_{l,1}$) are encryptions of 0. Finally, by the assumption that the two PKE key-pairs of B_j—i.e. $(\mathtt{pk}_0, \mathtt{sk}_0)$ and $(\mathtt{pk}_1, \mathtt{sk}_1)$—are correct, the decryption of either $c_{l,0}$ or $c_{l,1}$ will result in 0 being output, and so the signature will not verify as valid by $\mathcal{O}_V$. □

□

F.2.2 Public-Key Collision Resistance

Definition 23 (n-Instance ε-Image Collision Resistance [20]). *A* OWF $\Pi = (S, F)$ *is n-Instance ε-Image Collision Resistant if*

$$\Pr\left[\big|\{\Pi.F(x_1), \dots, \Pi.F(x_n)\}\big| < n \;\middle|\; \begin{array}{c} x_1 \leftarrow \Pi.S(1^k) \\ \dots \\ x_n \leftarrow \Pi.S(1^k) \end{array} \right] \leq \varepsilon.$$

Refer to [20] for a proof of Lemma 1.

Lemma 1. *If no adversary (ε, t)-breaks the (n)-One-Wayness of a* OWF $\Pi = (S, F)$*, with $t \gtrapprox n \cdot (t_S + t_F)$—where t_S and t_F are, respectively, the times to run S and F—then Π is n-Instance ε'-Image Collision-Resistant, with $\varepsilon' \leq 2 \cdot \varepsilon$.*

Corollary 5. *If no adversary $(\varepsilon_{\mathrm{OWF}}, t_{\mathrm{OWF}})$-breaks the (n_{OWF})-One-Wayness of $\Pi_{\mathrm{OWF}} = (S, F)$, with $t_{\mathrm{OWF}} \gtrapprox n_{\mathrm{OWF}} \cdot (t_S + t_F)$—where t_S and t_F are, respectively, the times to run $\Pi_{\mathrm{OWF}}.S$ and $\Pi_{\mathrm{OWF}}.F$—then $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$ is*

$$(n := \max(n_{\mathrm{OWF}} - n_V, 0), \ell := \max(n_{\mathrm{OWF}} - n_S, 0))\text{-}\textit{Party}$$
$$\varepsilon\text{-}\textit{Public-Key Collision-Resistant}$$

with $\varepsilon \geq 2 \cdot \varepsilon_{\mathrm{OWF}}$.

Proof. Follows from the definition of $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$ (Algorithm 30), from Lemma 1 and from the assumption on Π_{OWF}. □

G Security of Maurer et al.'s MDRS-PKE Construction [35]

In this section we prove the security of Maurer et al.'s MDRS-PKE construction [35] (see Algorithm 31) with respect to our new security notions (see Section 6.2).

G.1 Consistency

Theorem 7. *If no adversary $(\varepsilon_{\text{PKEBC}}, t_{\text{PKEBC}})$-breaks the $(n_{\text{PKEBC}}, d_{E\text{PKEBC}}, q_{E\text{PKEBC}}, q_{D\text{PKEBC}})$-Consistency of Π_{PKEBC}, no adversary $(\varepsilon_{\text{MDVS}}, t_{\text{MDVS}})$-breaks the $(n_{S\text{MDVS}}, n_{V\text{MDVS}}, d_{S\text{MDVS}}, d_{F\text{MDVS}}, q_{S\text{MDVS}}, q_{V\text{MDVS}}, q_{F\text{MDVS}})$-Consistency of Π_{MDVS} and $\Pi_{\text{DSS}}.\mathit{Vfy}$ is a deterministic algorithm, then no adversary (ε, t)-breaks $\Pi_{\text{MDRS-PKE}}$'s*

$$\begin{aligned}(&n_S \coloneqq n_{S\text{MDVS}}, n_R \coloneqq \min(n_{\text{PKEBC}}, n_{V\text{MDVS}}),\\ &d_E \coloneqq \min(d_{E\text{PKEBC}}, d_{S\text{MDVS}}), q_E \coloneqq \min(q_{E\text{PKEBC}}, q_{S\text{MDVS}}),\\ &q_D \coloneqq \min(q_{D\text{PKEBC}}, q_{V\text{MDVS}}))\text{-}\mathit{Consistency},\end{aligned}$$

with $\varepsilon > \varepsilon_{\text{PKEBC}} + \varepsilon_{\text{MDVS}}$, and $t_{\text{PKEBC}}, t_{\text{MDVS}} \approx t + t_{\mathsf{Cons}}$, where t_{Cons} is the time to run $\Pi_{\text{MDRS-PKE}}$'s $\mathbf{G}^{\mathsf{Cons}}$ game.

Proof. Follows from the proof of [35, Theorem 7] and the definition of $\Pi_{\text{MDRS-PKE}}$'s decryption (see Algorithm 31). □

G.2 Replay Unforgeability

Theorem 8. *If no adversary $(\varepsilon_{\text{MDVS}}, t_{\text{MDVS}})$-breaks the $(n_{S\text{MDVS}}, n_{V\text{MDVS}}, d_{S\text{MDVS}}, q_{S\text{MDVS}}, q_{V\text{MDVS}})$-Unforgeability of Π_{MDVS} and no adversary $(\varepsilon_{\text{DSS}}, t_{\text{DSS}})$-breaks the $(n_{\text{DSS}}, q_{S\text{DSS}}, q_{V\text{DSS}})$-1-$\mathsf{sEUF\text{-}CMA}$ security of Π_{DSS}, then no adversary $\mathbf{A}$ (ε, t)-breaks $\Pi_{\text{MDRS-PKE}}$'s*

$$\begin{aligned}(&n_S \coloneqq n_{S\text{MDVS}}, n_R \coloneqq n_{V\text{MDVS}},\\ &d_E \coloneqq d_{S\text{MDVS}}, q_E \coloneqq \min(q_{S\text{MDVS}}, n_{\text{DSS}}, q_{S\text{DSS}}),\\ &q_D \coloneqq \min(q_{V\text{MDVS}}, q_{V\text{DSS}}))\text{-}\mathit{Replay\ Unforgeability},\end{aligned}$$

with $\varepsilon > \varepsilon_{\text{DSS}} + \varepsilon_{\text{MDVS}}$, and $t_{\text{DSS}}, t_{\text{MDVS}} \approx t + t_{\mathsf{R\text{-}Unforg}}$, where $t_{\mathsf{R\text{-}Unforg}}$ is the time to run $\Pi_{\text{MDRS-PKE}}$'s $\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ game.

Proof. This proof proceeds via a sequence of games.

$\mathbf{G}^{\mathsf{R\text{-}Unforg}} \rightsquigarrow \mathbf{G}^1$: The difference between $\mathbf{G}^1$ and $\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ is that in $\mathbf{G}^1$, when $\mathcal{O}_D$ is queried on an input $(B_j, c \coloneqq (\mathtt{vk}, \sigma', c'))$ such that there is a query $\mathcal{O}_E(A_i, \vec{V}, m)$ that output $c^* \coloneqq (\mathtt{vk}^*, \sigma'^*, c'^*)$ with $c \neq c^*$ and $\mathtt{vk} = \mathtt{vk}^*$, $\mathcal{O}_D$ simply outputs $\perp$.

One can reduce distinguishing the two games to breaking the 1-$\mathsf{sEUF\text{-}CMA}$ security of Π_{DSS}: the reduction holds all MDVS and PKEBC secret keys and can sign ciphertexts using the $\mathcal{O}_S$ oracle from Π_{DSS}'s $\mathbf{G}^{1\text{-}\mathsf{sEUF\text{-}CMA}}$ game so it can handle any oracle queries. If $\mathbf{A}$ only makes at most $q_E \leq \min(n_{\text{DSS}}, q_{S\text{DSS}})$ and $q_D \leq q_{V\text{DSS}}$ queries to $\mathcal{O}_E$ and $\mathcal{O}_D$, respectively, since by assumption no adversary $(t_{\text{DSS}}, \varepsilon_{\text{DSS}})$-breaks the

$$(n_{\text{DSS}}, q_{S\text{DSS}}, q_{V\text{DSS}})\text{-}1\text{-}\mathsf{sEUF\text{-}CMA}$$

of Π_{DSS}, it follows

$$\left|\Pr[\mathbf{AG}^{1} = \texttt{win}] - \Pr[\mathbf{AG}^{\textsf{R-Unforg}} = \texttt{win}]\right| \leq \varepsilon_{\mathrm{DSS}}.$$

We can now directly reduce to the Unforgeability game of Π_{MDVS}. To see why, note that $\mathbf{G}^1$ already outputs $\perp$ for any query $\mathcal{O}_D(B_j, c \coloneqq (\texttt{vk}, \sigma', c'))$ such that there was a query $\mathcal{O}_E(A_i, \vec{V}, m)$ that output $c^* \coloneqq (\texttt{vk}^*, {\sigma'}^*, {c'}^*)$ with $c \neq c^*$ and $\texttt{vk} = \texttt{vk}^*$. This then means that we only have to make sure that no decryption query $\mathcal{O}_D(B_j, c \coloneqq (\texttt{vk}, \sigma', c'))$ such that there was no query $\mathcal{O}_E(A_i, \vec{V}, m)$ that output $c^* \coloneqq (\texttt{vk}^*, {\sigma'}^*, {c'}^*)$ with $c \neq c^*$ and $\texttt{vk} = \texttt{vk}^*$ allows the adversary to win the game. On one hand, if $c = c^*$ then the adversary does not win the game (see Definition 18); on the other hand, if some query $\mathcal{O}_D(B_j, c \coloneqq (\texttt{vk}, \sigma', c'))$ (where $\texttt{vk}$ was not output as part of any challenge ciphertext) outputs something other than $\perp$, then the MDVS signature encrypted by c' actually verified as being a valid signature on a triple $(\vec{v}_{\mathrm{PKEBC}}, m, \texttt{vk})$ which was never signed (since $\texttt{vk}$ was not output as part of any $\mathcal{O}_E$ ciphertext).

Since by assumption no adversary $(\varepsilon_{\mathrm{MDVS}}, t_{\mathrm{MDVS}})$-breaks the

$$(n_{S\mathrm{MDVS}}, n_{V\mathrm{MDVS}}, d_{S\mathrm{MDVS}}, q_{S\mathrm{MDVS}}, q_{V\mathrm{MDVS}})\text{-Unforgeability}$$

of Π_{MDVS}, if $\mathbf{A}$ only queries for at most $n_S \leq n_{S\mathrm{MDVS}}$ (resp. $n_R \leq n_{V\mathrm{MDVS}}$) different sender keys (resp. different receiver keys), makes up to $q_E \leq q_{S\mathrm{MDVS}}$ queries to $\mathcal{O}_E$ and up to $q_D \leq q_{V\mathrm{MDVS}}$ queries to $\mathcal{O}_D$, and the sum of lengths of the party vectors input to $\mathcal{O}_E$ is at most $d_E \leq d_{S\mathrm{MDVS}}$, it follows

$$\Pr[\mathbf{AG}^{1} = \texttt{win}] \leq \varepsilon_{\mathrm{MDVS}}.$$

□

G.3 {IND, IK}-CCA-2 Security

Theorem 9. *If no adversary $(\varepsilon_{\mathrm{MDVS}}, t_{\mathrm{MDVS}})$-breaks the $(n_{S\mathrm{MDVS}}$, $n_{V\mathrm{MDVS}}$, $d_{S\mathrm{MDVS}}$, $q_{S\mathrm{MDVS}}$, $q_{V\mathrm{MDVS}})$-Message-Bound Validity of Π_{MDVS}, no adversary $(\varepsilon_{\mathrm{DSS}}, t_{\mathrm{DSS}})$-breaks the $(n_{\mathrm{DSS}}, q_{S\mathrm{DSS}}, q_{V\mathrm{DSS}})$-1-*sEUF-CMA *security of Π_{DSS}, no adversary $(\varepsilon_{\mathrm{PKEBC}}, t_{\mathrm{PKEBC}})$-breaks the $(n_{\mathrm{PKEBC}}$, $d_{E\mathrm{PKEBC}}$, $q_{E\mathrm{PKEBC}}$, $q_{D\mathrm{PKEBC}})$-Robustness of Π_{PKEBC}, and no adversary $(\varepsilon_{\mathrm{PKEBC}}, t_{\mathrm{PKEBC}})$-breaks the $(n_{\mathrm{PKEBC}}$, $d_{E\mathrm{PKEBC}}$, $q_{E\mathrm{PKEBC}}$, $q_{D\mathrm{PKEBC}})$-*{IND, IK}-CCA-2 *security of Π_{PKEBC}, then no adversary $\mathbf{A}$ (ε, t)-breaks $\Pi_{\text{MDRS-PKE}}$'s*

$$\Big(n_S \coloneqq n_{S\mathrm{MDVS}}, n_R \coloneqq \min(n_{\mathrm{PKEBC}}, n_{V\mathrm{MDVS}}), d_E \coloneqq \min(d_{E\mathrm{PKEBC}}, d_{S\mathrm{MDVS}}),$$
$$q_E \coloneqq \min(q_{E\mathrm{PKEBC}}, q_{S\mathrm{MDVS}}, n_{\mathrm{DSS}}, q_{S\mathrm{DSS}}),$$
$$q_D \coloneqq \min(q_{D\mathrm{PKEBC}}, q_{V\mathrm{MDVS}}, q_{V\mathrm{DSS}})\Big)\text{-}\{\textsf{IND}, \textsf{IK}\}\text{-}\textsf{CCA-2}\ \textit{security},$$

with

$$\varepsilon > 2 \cdot (\varepsilon_{\textsf{DSS-1-EUF-CMA}} + \varepsilon_{\textsf{MDVS-Bound-Val}} + \varepsilon_{\textsf{PKEBC-Rob}}) + 4 \cdot \varepsilon_{\textsf{PKEBC-Corr}} + \varepsilon_{\textsf{PKEBC-\{IND,IK\}-CCA-2}},$$

and with $t_{\mathrm{DSS}}, t_{\mathrm{MDVS}}, t_{\mathrm{PKEBC}} \approx t + t_{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$*, where* $t_{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$ *is the time to run* Π*'s* $\mathbf{G}^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$ *games.*

Proof. One can prove Theorem 9 by following the same sequence of hybrids (and the same arguments) from [19, Proof of Theorem 13], with only minor differences. Below we only explain the differences from [19, Proof of Theorem 13], i.e. we explain how to handle queries $\mathcal{O}_E\big((A_{i,0}, \vec{V_0}, m_0), (A_{i,1}, \vec{V_1}, m_1)\big)$ for the case where $(A_{i,0}, \vec{V_0}, m_0) = (A_{i,1}, \vec{V_1}, m_1)$—which are not considered in the $\{\mathsf{IND},\mathsf{IK}\}$-$\mathsf{CCA\text{-}2}$ notion from [19] for the case where either not all receivers in $\vec{V_0}$ and $\vec{V_1}$ are honest (i.e. the adversary may query for secret keys), vectors $\vec{V_0}$ and $\vec{V_1}$ are of different lengths, or m_0 and m_1 have different sizes.

First, note that regardless of whether an adversary is interacting with game $\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$ or $\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$, the ciphertexts generated by the oracle in such queries have exactly the same distribution, and therefore we only need to ensure the reductions have everything needed to produce such ciphertexts.

For $\beta \in \{0,1\}$, [19, Proof of Theorem 13] proceeds via a sequence of hybrids $\mathbf{G}_{\beta}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}} \rightsquigarrow \mathbf{G}_{\beta}^{1}$, $\mathbf{G}_{\beta}^{1} \rightsquigarrow \mathbf{G}_{\beta}^{2}$, $\mathbf{G}_{\beta}^{2} \rightsquigarrow \mathbf{G}_{\beta}^{3}$, $\mathbf{G}_{\beta}^{3} \rightsquigarrow \mathbf{G}_{\beta}^{4}$ and $\mathbf{G}_{\beta}^{4} \rightsquigarrow \mathbf{G}_{\beta}^{5}$, and gives reductions from distinguishing each of these pairs to breaking a security property of one of the underlying building blocks. Concretely, [19, Proof of Theorem 13] reduces distinguishing

S.1 $\mathbf{G}_{\beta}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA\text{-}2}}$ and $\mathbf{G}_{\beta}^{1}$ to breaking the 1-$\mathsf{sEUF\text{-}CMA}$ security of the underlying Π_{DSS};
S.2 $\mathbf{G}_{\beta}^{1}$ and $\mathbf{G}_{\beta}^{2}$ to breaking the robustness of the underlying Π_{PKEBC};
S.3 $\mathbf{G}_{\beta}^{2}$ and $\mathbf{G}_{\beta}^{3}$ to breaking the correctness of the underlying Π_{PKEBC};
S.4 $\mathbf{G}_{\beta}^{3}$ and $\mathbf{G}_{\beta}^{4}$ to breaking the message-bound validity of the underlying Π_{MDVS};
S.5 $\mathbf{G}_{\beta}^{4}$ and $\mathbf{G}_{\beta}^{5}$ to breaking the correctness of the underlying Π_{PKEBC}; and
S.6 $\mathbf{G}_{\mathbf{0}}^{5}$ and $\mathbf{G}_{\mathbf{1}}^{5}$ to breaking the $\{\mathsf{IND},\mathsf{IK}\}$-$\mathsf{CCA\text{-}2}$ security of the underlying Π_{PKEBC}.

The reductions corresponding to **S.2**, **S.3**, **S.5** and **S.6** are to PKEBC notions and therefore have access to all the DSS and MDVS secret keys; since only public keys are needed to generate PKEBC ciphertexts, and the reductions have access to these, they can handle the additional queries. (Note that, for the $\{\mathsf{IND},\mathsf{IK}\}$-$\mathsf{CCA\text{-}2}$ reduction, since the reduction generates the additional ciphertexts without relying on the $\mathcal{O}_E$ oracle provided by games, it can then still use the $\mathcal{O}_D$ oracle provided by the games to handle decryption queries even if the PKEBC component of such ciphertexts is left unchanged by the adversary).

The reduction corresponding to **S.4** is to MDVS message-bound validity, which does allow adversaries to query for MDVS secret keys of senders. Since the reduction is to an MDVS notion, it has access to all PKEBC and DSS secret keys; as it also has access to the secret keys of senders, it can generate any ciphertexts for the additional queries too.[35]

[35] We note that one can also adapt the analogous reduction to the unforgeability of Π_{MDVS} of [35,36, Proof of Theorem 10]—which captures the setting where the secret

The reduction corresponding to **S.1** is to the 1-sEUF-CMA security of Π_{DSS} and therefore has access to all the PKEBC and MDVS secret keys. Noting that the MDRS-PKE encryption algorithm samples a fresh Π_{DSS} key-pair for encryption, for these additional queries one can have the reduction simply sample the Π_{DSS} key-pair itself, and therefore can generate the ciphertext as intended. (Alternatively, one could rely on the 1-sEUF-CMA game of Π_{DSS} to sample the key-pairs, and then use the $\mathcal{O}_S$ oracle provided by the game to generate the signatures, but this is not necessary for this reduction.) □

G.4 Forgery Invalidity

Theorem 10. *If no adversary $(\varepsilon_{\text{PKEBC}}, t_{\text{PKEBC}})$-breaks the $(n_{\text{PKEBC}}, d_{E\text{PKEBC}}, q_{E\text{PKEBC}}, q_{D\text{PKEBC}})$-Correctness of Π_{PKEBC} and no adversary $(\varepsilon_{\text{MDVS}}, t_{\text{MDVS}})$-breaks the $(n_S, n_V, d_S, d_F, q_S, q_V, q_F)$-Forgery Invalidity of Π_{MDVS} then no adversary (ε, t)-breaks $\Pi_{\text{MDRS-PKE}}$'s*

$$\begin{aligned}(n_S &:= n_{S\text{MDVS}},\\ n_R &:= \min(n_{\text{PKEBC}}, n_{V\text{MDVS}}),\\ d_F &:= \min(d_{E\text{PKEBC}}, d_{S\text{MDVS}}),\\ q_F &:= \min(q_{E\text{PKEBC}}, q_{F\text{MDVS}}),\\ q_D &:= \min(q_{D\text{PKEBC}}, q_{V\text{MDVS}}))\text{-}\textit{Forgery Invalidity},\end{aligned}$$

with $\varepsilon > \varepsilon_{\text{PKEBC}} + \varepsilon_{\text{MDVS}}$ and $t_{\text{PKEBC}}, t_{\text{MDVS}} \approx t + t_{\textsf{Forge-Invalid}}$, where $t_{\textsf{Forge-Invalid}}$ is the time to run $\Pi_{\text{MDRS-PKE}}$'s $\mathbf{G}^{\textsf{Forge-Invalid}}$ game.

Proof. We prove this result via game hopping.

$\mathbf{G}^{\textsf{Forge-Invalid}} \rightsquigarrow \mathbf{G}^1$: The only difference between games $\mathbf{G}^{\textsf{Forge-Invalid}}$ and $\mathbf{G}^1$ is that in $\mathbf{G}^1$ some decryption queries are handled differently. More concretely, when $\mathcal{O}_D$ is queried on an input $(B_j, c := (\texttt{vk}, \sigma', c'))$ where c was output by a query $\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C})$ such that $B_j \in \text{Set}(\vec{V}) \setminus \mathcal{C}$, oracle $\mathcal{O}_D$ works as follows: let $(\texttt{spk}_i, \vec{v}_{\text{MDVS}}, m, \sigma)$ be the plaintext that was encrypted by $\Pi_{\text{PKEBC}}.E$ under $\vec{v}_{\text{PKEBC}}$ (which resulted in ciphertext c'), where $\texttt{spk}_i$ is A_i's public key,

$$\begin{aligned}\vec{v}_{\text{MDVS}} &:= (\texttt{vpk}_{\text{MDVS}1}, \dots, \texttt{vpk}_{\text{MDVS}|\vec{v}|}), \text{ and}\\ \vec{v}_{\text{PKEBC}} &:= (\texttt{pk}_{\text{PKEBC}1}, \dots, \texttt{pk}_{\text{PKEBC}|\vec{v}|})\end{aligned}$$

are, respectively, the vectors of public MDVS verifier keys and public PKEBC receiver keys corresponding to $\vec{V}$, and where

$$\sigma \leftarrow \Pi_{\text{MDVS}}.\mathit{Forge}_{\texttt{pp}_{\text{MDVS}}}(\texttt{spk}_{\text{MDVS}i}, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \texttt{vk}), \vec{s}_{\text{MDVS}}),$$

keys of honest senders do not leak—to handle the additional encryption queries, because the MDVS unforgeability game provides a signing oracle which the reduction could use to generate the necessary MDVS signatures.

is a forged MDVS signature on $(\vec{v}_{\text{PKEBC}}, m, \texttt{vk})$ using vector of secret keys $\vec{s}_{\text{MDVS}}$ (as defined by oracle $\mathcal{O}_{Forge}$), and $\texttt{vk}$ being the DSS verification key in c; oracle $\mathcal{O}_D$ no longer decrypts c' using $\Pi_{\text{PKEBC}}.D$ with B_j's PKEBC secret key, and instead simply assumes decryption outputs $(\vec{v}_{\text{PKEBC}}, (\texttt{spk}_i, \vec{v}_{\text{MDVS}}, m, \sigma))$.

It is easy to see that one can reduce distinguishing the two games to breaking the correctness of Π_{PKEBC}: since the reduction holds all secret keys, it can handle any oracle queries. If $\mathbf{A}$ only queries for at most $n_R \leq n_{\text{PKEBC}}$ different receivers, the sum of lengths of the vectors input to $\mathcal{O}_{Forge}$ is at most $d_F \leq d_{E\text{PKEBC}}$, and makes at most $q_F \leq q_{E\text{PKEBC}}$ and $q_D \leq q_{D\text{PKEBC}}$ queries to oracles $\mathcal{O}_{Forge}$ and $\mathcal{O}_D$, respectively, since by assumption no adversary $(t_{\text{PKEBC}}, \varepsilon_{\text{PKEBC}})$-breaks the

$$(n_{\text{PKEBC}}, d_{E\text{PKEBC}}, q_{E\text{PKEBC}}, q_{D\text{PKEBC}})\text{-Correctness}$$

of Π_{PKEBC}, it follows

$$\left|\Pr[\mathbf{AG}^1 = \texttt{win}] - \Pr[\mathbf{AG}^{\textsf{Forge-Invalid}} = \texttt{win}]\right| \leq \varepsilon_{\text{PKEBC}}.$$

$\mathbf{G}^1 \rightsquigarrow \mathbf{G}^2$: Game $\mathbf{G}^2$ is just like $\mathbf{G}^1$, except that once again some decryption queries are handled differently. In contrast to the previous hop—where $\mathbf{G}^1$ differed from $\mathbf{G}^{\textsf{Forge-Invalid}}$ in that it assumed the ciphertext c' in each ciphertext $c \coloneqq (\texttt{vk}, \sigma', c')$ output by a query $\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C})$ decrypted correctly when $\mathcal{O}_D$ was queried on (B_j, c), with $B_j \in \text{Set}(\vec{V}) \setminus \mathcal{C}$—game $\mathbf{G}^2$ differs from $\mathbf{G}^1$ in that it now assumes that each MDVS signature σ generated by $\mathcal{O}_{Forge}$ using $\Pi_{\text{MDVS}}.Forge$ does not verify as being valid when $\mathcal{O}_D$ is queried on a matching input. To be more precise, for a query $\mathcal{O}_{Forge}(A_i, \vec{V}, m, \mathcal{C})$: let $(\vec{v}_{\text{PKEBC}}, m, \texttt{vk})$ be the plaintext on which an MDVS signature was forged with respect to $\texttt{spk}_i$ and $\vec{v}_{\text{MDVS}}$ using $\Pi_{\text{MDVS}}.Forge$, where $\texttt{spk}_i$ is A_i's public key,

$$\begin{aligned}\vec{v}_{\text{MDVS}} &\coloneqq (\texttt{vpk}_{\text{MDVS}1}, \ldots, \texttt{vpk}_{\text{MDVS}|\vec{v}|}), \text{ and}\\ \vec{v}_{\text{PKEBC}} &\coloneqq (\texttt{pk}_{\text{PKEBC}1}, \ldots, \texttt{pk}_{\text{PKEBC}|\vec{v}|})\end{aligned}$$

are, respectively, the vectors of public MDVS verifier keys and public PKEBC receiver keys corresponding to $\vec{V}$; let σ be the resulting forged signature

$$\sigma \leftarrow \Pi_{\text{MDVS}}.Forge_{\texttt{pp}_{\text{MDVS}}}(\texttt{spk}_{\text{MDVS}i}, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \texttt{vk}), \vec{s}_{\text{MDVS}}),$$

where $\vec{s}_{\text{MDVS}}$ is as defined by $\mathcal{O}_{Forge}$; and let c be the ciphertext output by the $\mathcal{O}_{Forge}$ query. Then, when queried on input (B_j, c) such that $B_j \in \text{Set}(\vec{V}) \setminus \mathcal{C}$, $\mathcal{O}_D$ no longer verifies if σ is valid by running

$$\Pi_{\text{MDVS}}.\,Vfy(\texttt{pp}, \texttt{spk}_i, \texttt{vsk}_j, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \texttt{vk}))$$

and instead simply assumes the MDVS signature verification outputs 0—implying $\mathcal{O}_D$ outputs $\perp$.

It is easy to see that one can reduce distinguishing $\mathbf{G}^1$ and $\mathbf{G}^2$ to breaking the Forgery Invalidity of Π_{MDVS}: since the reduction holds all secret keys, it can handle any oracle queries. If $\mathbf{A}$ only queries for at most $n_S \leq n_{S\text{MDVS}}$ different senders and $n_R \leq n_{V\text{MDVS}}$ different receivers, the sum of lengths of the vectors input to $\mathcal{O}_{Forge}$ is at most $d_F \leq d_{F\text{MDVS}}$, and makes at most $q_F \leq q_{F\text{MDVS}}$ and $q_D \leq q_{D\text{MDVS}}$ queries to oracles $\mathcal{O}_{Forge}$ and $\mathcal{O}_D$, respectively, since by assumption no adversary $(t_{\text{MDVS}}, \varepsilon_{\text{MDVS}})$-breaks Π_{MDVS}'s

$$(n_{S\text{MDVS}}, n_{V\text{MDVS}}, d_{S\text{MDVS}}, d_{F\text{MDVS}}, q_{S\text{MDVS}}, q_{V\text{MDVS}}, q_{F\text{MDVS}})\text{-}$$

Forgery Invalidity, it follows

$$\left|\Pr[\mathbf{AG}^2 = \mathtt{win}] - \Pr[\mathbf{AG}^1 = \mathtt{win}]\right| \leq \varepsilon_{\text{MDVS}}.$$

To conclude the proof note that no adversary can win $\mathbf{G}^2$, implying

$$\Pr[\mathbf{AG}^2 = \mathtt{win}] = 0.$$

□

G.5 Public-Key Collision Resistance

Corollary 6. *If Π_{MDVS} is $(n_{\text{MDVS}}, \ell_{\text{MDVS}})$-Party ε-Public-Key Collision Resistant then $\Pi_{\text{MDRS-PKE}}$ is*

$$(n \coloneqq n_{\text{MDVS}}, \ell \coloneqq \ell_{\text{MDVS}})\text{-}\textit{Party}$$
$$\varepsilon\text{-}\textit{Public-Key Collision-Resistant.}$$

Proof. Follows from the definition of $\Pi_{\text{MDRS-PKE}}$ (Algorithm 31) and the assumption on Π_{MDVS}. □

H Application Semantics of MDRS-PKE Game-Based Notions

As in Section 3, we consider a set of parties $\mathcal{F}$ consisting of all senders and receivers, i.e. $\mathcal{F} \coloneqq \mathcal{S} \cup \mathcal{R}$. The theorems below establish composable semantics for the MDRS-PKE game-based notions we introduced in Section 6 together with the ones from [19,35].

Theorem 11. *Consider simulator* sim *defined in Algorithms 32 and 35, reductions $\mathbf{C}^{\mathsf{Cons}\text{-}H}$, $\mathbf{C}^{\mathsf{Cons}}$, $\mathbf{C}^{\mathsf{Corr}}$, $\mathbf{C}^{\mathsf{R\text{-}Unforg}}$, $\mathbf{C}^{\mathsf{CCA}}$ and $\mathbf{C}^{\mathsf{OTR}}$ (defined, respectively, in Algorithms 33, 34 and 36, Algorithms 33, 34 and 37, Algorithms 33, 34 and 38, Algorithms 33, 34 and 39, Algorithms 33, 34 and 40, and Algorithms 33, 34 and 41), and reductions $\perp^J\cdot\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$, $\perp^J\cdot\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}$, $\perp^J\cdot\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}$, $\perp^J\cdot\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}$, $\perp^J\cdot\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}$, $\perp^J\cdot\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}$, $\perp^J\cdot\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}$ and $\perp^J\cdot\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}$ (where $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}$ and $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}$ are defined, respectively, in Algorithms 33 and 49, Algorithms 33 and 50, Algorithms 33 and 51, Algorithms 33 and 52, Algorithms 33 and 53,* *Algorithms 33 and 54, Algorithms 33 and 55, and Algorithms 33 and 56). If the* MDRS-PKE *scheme is (m,n)-Party ε-Public Key Collision Resistant, then for any distinguisher* $\mathbf{D}$*,*

$$\begin{aligned}
&\Delta^{\mathbf{D}}\Bigg(\mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}\bot^J[\mathbf{KGA},\mathsf{Net}\cdot\mathbf{INS}],\\
&\quad \mathsf{sim}^{\overline{\mathcal{P}^H}}\cdot\mathsf{ConfAnon}^{\overline{\mathcal{P}^H}}\cdot\mathsf{Otr}^{\overline{\mathcal{P}^H}}\cdot\\
&\qquad\left[\begin{array}{c}\mathsf{Net}\cdot\bot^{\mathsf{Auth\text{-}Intf}}\cdot\left[\langle A_i\to\vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{Set(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{\substack{A_i\in\mathcal{S}\\ \vec{V}\in\mathcal{R}^+}}\\ \left[\langle[Forge]A_i\to\vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+}\end{array}\right]\Bigg)\\
&\quad\le 4\cdot\Big(Adv^{\mathsf{OTR}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1})+Adv^{\mathsf{Corr}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1})\\
&\qquad\quad+Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big)\\
&\qquad+Adv^{\mathsf{OTR}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2})+Adv^{\mathsf{Cons}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2})\\
&\qquad+Adv^{\mathsf{Cons}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2})+Adv^{\mathsf{Corr}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2})\\
&\qquad+Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^J\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2})+Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}})\\
&\qquad+Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}})+Adv^{\mathsf{R\text{-}Unforg}}(\mathbf{D}\mathbf{C}^{\mathsf{R\text{-}Unforg}})+\varepsilon\\
&\qquad+Adv^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}}(\mathbf{D}\mathbf{C}^{\mathsf{CCA}})+Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}}).
\end{aligned}$$

Theorem 12. *Consider simulator* sim *defined in Algorithms 32 and 42, reductions* $\mathbf{C}^{\mathsf{Cons}\text{-}H}$, $\mathbf{C}^{\mathsf{Cons}}$, $\mathbf{C}^{\mathsf{Corr}}$, $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}}$, $\mathbf{C}^{\mathsf{CCA}}$ *and* $\mathbf{C}^{\mathsf{OTR}}$ *(defined, respectively, in Algorithms 33, 34 and 43, Algorithms 33, 34 and 44, Algorithms 33, 34 and 45, Algorithms 33, 34 and 46, Algorithms 33, 34 and 47 and Algorithms 33, 34 and 48), and reductions* $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}$, $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}$, $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}$ *and* $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}$ *(defined, respectively, in Algorithms 33 and 49, Algorithms 33 and 50, Algorithms 33 and 51, Algorithms 33 and 52, Algorithms 33 and 53, Algorithms 33 and 54, Algorithms 33 and 55, and Algorithms 33 and 56). If the* MDRS-PKE *scheme is (m,n)-Party ε-Public Key*

Collision Resistant, then for any distinguisher $\mathbf{D}$,

$$\begin{aligned}
&\Delta^{\mathbf{D}}(\mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}],\\
&\quad \mathsf{sim}^{\overline{\mathcal{P}^H}} \cdot \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \begin{bmatrix} \mathsf{Net} \cdot \left[\langle A_i \to \vec{V}\rangle^{\overline{\mathcal{P}^H}}_{Set(\vec{V})\cup\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \\ \left[\langle [Forge]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix})\\
&\leq 4\cdot\Big(Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1})\\
&\qquad + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big)\\
&\quad + Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}\text{-}\mathit{0}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}\text{-}\mathit{1}\text{-}\xi_2})\\
&\quad + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2})\\
&\quad + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}}) + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}})\\
&\quad + \varepsilon + Adv^{\{\mathsf{IND,\,IK}\}\text{-}\mathsf{CCA\text{-}2}}(\mathbf{DC}^{\mathsf{CCA}}) + Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}}).
\end{aligned}$$

H.1 Proofs

For simplicity, in Algorithm 32 we describe the behavior of the simulators we will consider in the proofs of Theorems 11 and 12 for the (sub-)interfaces of dishonest parties that correspond to an interface of the **KGA** resource in the real world system. Similarly, in Algorithm 33 we describe the behavior of the reductions we will consider in these proofs for the same (sub-)interfaces.

H.1.1 Helper Claims Below we state two useful results that help in simplifying the proofs of Theorems 11 and 12. (See Sections H.1.4 and H.1.5 for their proofs.) Consider the following events:

Event ξ_1 There are two WRITE queries at the interface of an honest party $A_i \in \mathcal{S}^H$ that output `id` and `id′` with `id` $\neq$ `id′`, such that the contents of the registers with these identifiers (i.e. `id` and `id′`) are the same.

Event ξ_2 There is a WRITE query at a dishonest party's interface with input ciphertext c that outputs a register identifier `id` and there is a later WRITE query at the interface of an honest party $A_i \in \mathcal{S}^H$ that outputs a register identifier `id′` such that the contents of the registers with identifiers `id` and `id′` are the same.

At a high level, Lemmata 2 and 3 bound the probability of events ξ_1 and ξ_2 to occur. The reason why these results are necessary is that in the real world duplicate ciphertexts are filtered out (to protect against replay attacks), and so if either event would occur one would be able to distinguish the real and ideal worlds. In the following $\mathbf{R}$ is defined as in Section 6.1, i.e.

$$\mathbf{R} := \mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{(\mathcal{F}\times\{\mathsf{Forge}\})}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}]. \tag{H.1}$$

Lemma 2. *For any distinguisher* $\mathbf{D}$*, the probability that event* ξ_1 *occurs when it interacts with the real world system* $\mathbf{R}$ *(Equation H.1) is upper bounded by*

$$4 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}) \\ + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big).$$

where $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$*,* $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}$ *and* $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}$ *are the reductions given in Algorithms 33 and 49, Algorithms 33 and 50, and Algorithms 33 and 51.*

A proof of Lemma 2 is given in Section H.1.4.

Lemma 3. *For any distinguisher* $\mathbf{D}$*, the probability that event* ξ_2 *occurs when it interacts with the real world system* $\mathbf{R}$ *(Equation H.1) is upper bounded by*

$$Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}) \\ + Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}).$$

where $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}$*,* $\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}$*,* $\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}$*,* $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}$ *and* $\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}$ *are the reductions given in Algorithms 33 and 52, Algorithms 33 and 53, Algorithms 33 and 54, Algorithms 33 and 55, and Algorithms 33 and 56.*

A proof of Lemma 3 is given in Section H.1.5.

Remark 1. Lemmata 2 and 3 rely on the Forgery Invalidity of the MDRS-PKE scheme. One can alternatively rely on the Unforgeability against Replays if the secret keys of honest senders do not leak.

H.1.2 Proof of Theorem 11

Proof. Let $\mathbf{R}$ be the real world system

$$\mathbf{R} := \mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}\bot^{J}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}],$$

$\mathbf{T}$ be the ideal repository defined in Equation 3.5, i.e.

$$\mathbf{T} := \begin{pmatrix} \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \\ \cdot\,\mathsf{Otr}^{\overline{\mathcal{P}^H}} \end{pmatrix} \cdot \begin{bmatrix} \mathsf{Net}\cdot\bot^{\mathsf{Auth\text{-}Intf}}\cdot\Big[\langle A_i \to \vec{V}\rangle^{\{A_i\}\cup\overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\Big]_{\substack{A_i\in\mathcal{S}\\ \vec{V}\in\mathcal{R}^+}} \\ \Big[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\Big]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix}, \quad \text{(H.2)}$$

which, by Equation 3.4, is equivalent to

$$\mathbf{T} := \begin{pmatrix} \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \\ \cdot\,\mathsf{Otr}^{\overline{\mathcal{P}^H}} \end{pmatrix} \cdot \begin{bmatrix} \mathsf{Net}\cdot\begin{bmatrix} \Big[\langle A_i \to \vec{V}\rangle^{\{A_i\}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\Big]_{A_i\in\mathcal{S}^H,\vec{V}\in\mathcal{R}^+} \\ \Big[\langle A_i \to \vec{V}\rangle^{\overline{\mathcal{P}^H}}_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}\Big]_{A_i\in\overline{\mathcal{S}^H},\vec{V}\in\mathcal{R}^+} \end{bmatrix} \\ \Big[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle^{\mathcal{F}}_{\overline{\mathcal{P}^H}}\Big]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{bmatrix}, \quad \text{(H.3)}$$

and let sim be the simulator specified in Algorithms 32 and 35. The remainder of the proof bounds $\Delta^{\mathbf{D}}(\mathbf{R}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T})$ by proceeding in a sequence of hybrids. In the following, we consider the reduction systems defined in the lemma's statement.

$\mathbf{R} \rightsquigarrow \mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}$: It is easy to see that $\mathbf{R}$ and $\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}$ are the same sequence of conditional probability distributions (conditioned on event ξ_1 not occurring, see Section H.1.1) by considering, on one hand, the definition of $\mathbf{R}$—i.e. the definitions of converters Snd, Rcv and Forge (Algorithm 16), the definition of the **KGA** resource (Algorithms 15 and 17), and the definitions of **INS** (Algorithm 1) and of Net (Algorithm 3)—and, on the other hand, the definition of $\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}$—i.e. the definition of $\mathbf{G}^{\mathsf{Cons}}$ and its oracles (Definition 17 and Section 6.2) and the definition of $\mathbf{C}^{\mathsf{Cons\text{-}H}}$ (Algorithms 33, 34 and 36). By Lemma 2, it follows

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}) \leq 4 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big).$$

$\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}} \rightsquigarrow \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$: As for the previous step $\mathbf{R} \rightsquigarrow \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$, it is easy to see that the two systems are the exact same sequence of conditional probability distributions conditioned on event ξ_2 not occurring (see Section H.1.1). It then follows by Lemma 3

$$\begin{aligned}\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}) &\leq Adv^{\mathsf{OTR}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}) \\ &+ Adv^{\mathsf{Cons}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Cons\text{-}0}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Cons\text{-}1}\text{-}\xi_2}) \\ &+ Adv^{\mathsf{Corr}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}).\end{aligned}$$

$\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}} \rightsquigarrow \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$: The only difference between $\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$ and $\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$ is that in $\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$ each ciphertext that is either generated by a WRITE operation at the interface of an honest sender $A_i \in \mathcal{S}^H$ or input to a WRITE operation at the interface of a dishonest party $P \in \overline{\mathcal{P}^H}$ is decrypted only once, and decryption uses the secret key $\texttt{rsk}_{\text{pp}}$ corresponding to the public parameters public key $\texttt{rpk}_{\text{pp}}$. (For more details see Algorithms 37 and 38). Given $\mathbf{C}^{\mathsf{Cons}}$ does not query for the secret key of any receiver $B_j \in \mathcal{R}^H$ nor for the secret key of B_{pp}, the advantage of a distinguisher $\mathbf{D}$ in distinguishing $\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$ and $\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$ is bounded by the advantage of adversary $\mathbf{D}\mathbf{C}^{\mathsf{Cons}}$ in winning the consistency game $\mathbf{G}^{\mathsf{Cons}}$ (note that $\mathbf{C}^{\mathsf{Cons}}$ makes a query to $\mathcal{O}_D$ on party B_{pp} when queried for any WRITE operation, and when queried for a READ operation at the interface of a receiver $B_j \in \mathcal{R}^H$ makes a query to $\mathcal{O}_D$ for each $(\texttt{id}, c)$ in the reduction's internal repository **INS**) implying

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}) \leq Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}}).$$

$\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}} \rightsquigarrow \mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}$: System $\mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ differs from $\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$ in that ciphertexts generated by WRITE operations issued at the interface of honest senders are no longer decrypted by a query to $\mathcal{O}_D$ on party B_{pp}, and instead the result of their decryption is simply assumed to be the correct label-message pair. $\mathbf{D}$'s advantage in distinguishing $\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$ and $\mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ is upper bounded by the advantage of $\mathbf{D}\mathbf{C}^{\mathsf{Corr}}$ in winning the correctness game $\mathbf{G}^{\mathsf{Corr}}$, implying

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}, \mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}) \leq Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}}).$$

$\mathbf{C}^{\textsf{R-Unforg}}\mathbf{G}^{\textsf{R-Unforg}} \rightsquigarrow \mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}$**:** The main things to note for this step are that 1. $\mathbf{D}$ has no access to the secret key corresponding to $\texttt{rpk}_{\textsf{pp}}$ (i.e. the public parameters public key); 2. since J has a converter $\perp$ attached to her interface, $\mathbf{D}$ also has no access to the secret key of any honest sender $A_i \in \mathcal{S}^H$; and 3. the only case in which $\mathbf{C}^{\textsf{R-Unforg}}\mathbf{G}^{\textsf{R-Unforg}}$ may differ from $\mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}$ is if $\mathbf{D}$ makes a query for a WRITE operation at the interface of a dishonest party $P \in \overline{\mathcal{S}^H} \cup \overline{\mathcal{R}^H}$ with input ciphertext c whose decryption results in a label $\langle A_i \to \vec{V}\rangle$ where $A_i \in \mathcal{S}^H$ and yet there was no WRITE operation at the interface of A_i that resulted in ciphertext c. This allows us to bound the advantage of $\mathbf{D}$ in distinguishing $\mathbf{C}^{\textsf{R-Unforg}}\mathbf{G}^{\textsf{R-Unforg}}$ and $\mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}$ by the advantage of $\mathbf{D}\mathbf{C}^{\textsf{R-Unforg}}$ in winning $\mathbf{G}^{\textsf{R-Unforg}}$, implying

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\textsf{R-Unforg}}\mathbf{G}^{\textsf{R-Unforg}}, \mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}) \leq Adv^{\textsf{R-Unforg}}(\mathbf{D}\mathbf{C}^{\textsf{R-Unforg}}).$$

$\mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}} \rightsquigarrow \mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\textsf{OTR}}$**:** Systems $\mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}$ and $\mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\textsf{OTR}}$ are perfectly indistinguishable. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\textsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\textsf{IND,IK}\}\textsf{-CCA-2}}, \mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\textsf{OTR}}) = 0.$$

$\mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\textsf{OTR}} \rightsquigarrow \textsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}$**:** $\mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\textsf{OTR}}$ and $\textsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}$ are perfectly indistinguishable unless there is a pair of senders with the same public key or a pair of receivers with the same public key—in which case procedure GETLABEL may return $\perp$. Since by assumption the MDRS-PKE scheme is (m, n)-Party ε-Public Key Collision Resistant and there are m senders and n receivers, it follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\textsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\textsf{OTR}}, \textsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}) \leq \varepsilon.$$

To conclude the proof we use triangle inequality:

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{R}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}) \leq{}& \Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{C}^{\mathsf{Cons}\text{-}\mathrm{H}}\mathbf{G}^{\mathsf{Cons}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons}\text{-}\mathrm{H}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}, \mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}, \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}, \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND},\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}, \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}, \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}) \\
\leq 4 \cdot \Big(& Adv^{\mathsf{OTR}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}) \\
&\quad + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big) \\
&+ Adv^{\mathsf{OTR}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}) \\
&+ Adv^{\mathsf{Cons}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}) + Adv^{\mathsf{Corr}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}) \\
&+ Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\bot^{J}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}}) \\
&+ Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}}) + Adv^{\mathsf{R\text{-}Unforg}}(\mathbf{D}\mathbf{C}^{\mathsf{R\text{-}Unforg}}) + \varepsilon \\
&+ Adv^{\{\mathsf{IND},\,\mathsf{IK}\}\text{-}\mathsf{CCA}\text{-}2}(\mathbf{D}\mathbf{C}^{\mathsf{CCA}}) + Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}}).
\end{aligned}$$

□

H.1.3 Proof of Theorem 12

Proof. Let $\mathbf{R}$ be the real world system

$$\mathbf{R} := \mathsf{Snd}^{\mathcal{S}^H}\mathsf{Rcv}^{\mathcal{R}^H}\mathsf{Forge}^{\mathcal{F}}[\mathbf{KGA}, \mathsf{Net}\cdot\mathbf{INS}],$$

$\mathbf{S}$ be the ideal world's repository from Equation 3.2

$$\mathbf{S} := \mathsf{ConfAnon}^{\overline{\mathcal{P}^H}} \cdot \mathsf{Otr}^{\overline{\mathcal{P}^H}} \cdot \left[\begin{array}{c} \mathsf{Net} \cdot \left[\langle A_i \to \vec{V}\rangle_{\mathrm{Set}(\vec{V})\cup\overline{\mathcal{P}^H}}^{\overline{\mathcal{P}^H}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \\ \left[\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle_{\overline{\mathcal{P}^H}}^{\mathcal{F}}\right]_{A_i\in\mathcal{S},\vec{V}\in\mathcal{R}^+} \end{array}\right], \tag{H.4}$$

and let sim be the simulator specified in Algorithms 32 and 42. One can bound $\Delta^{\mathbf{D}}(\mathbf{R}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{S})$ by proceeding in a sequence of hybrids that is similar to the one given in the proof of Theorem 11. In the following, we consider the reduction systems defined in the lemma's statement.

The main thing to note in the reductions is that the distinguisher is not given access to the secret keys of any honest receivers nor to the secret key of B_{pp} (this is necessary to ensure we can use the adversary to win the underlying security games).

$\mathbf{R} \rightsquigarrow \mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}$: Analogous to step $\mathbf{R} \rightsquigarrow \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$ in the proof of Theorem 11. By Lemma 2, it follows

$$\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}) \leq 4 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}) \Big).$$

$\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}} \rightsquigarrow \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$: As for step $\mathbf{R} \rightsquigarrow \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}$ in the proof of Theorem 11, it is easy to see that the two systems are the exact same sequence of conditional probability distributions conditioned on event ξ_2 not occurring (see Section H.1.1). It then follows by Lemma 3

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}) \leq Adv^{\mathsf{OTR}}(\mathbf{D}\mathbf{C}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons\text{-}0}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons\text{-}1}\text{-}\xi_2}) + Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}).$$

$\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}} \rightsquigarrow \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$: Analogous to step $\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}} \rightsquigarrow \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}$ in the proof of Theorem 11. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}) \leq Adv^{\mathsf{Cons}}(\mathbf{D}\mathbf{C}^{\mathsf{Cons}}).$$

$\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}} \rightsquigarrow \mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}$: This step is analogous to step $\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}} \rightsquigarrow \mathbf{C}^{\mathsf{R\text{-}Unforg}}\mathbf{G}^{\mathsf{R\text{-}Unforg}}$ in the proof of Theorem 11. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}, \mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}) \leq Adv^{\mathsf{Corr}}(\mathbf{D}\mathbf{C}^{\mathsf{Corr}}).$$

$\mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} \rightsquigarrow \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}}$: Very similar to the previous step—the only difference is that in $\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}}$ the decryption of ciphertexts generated via a write operation at the interface of a party $P \in \overline{\mathcal{S}^H} \cup \overline{\mathcal{R}^H}$ by B_{pp} is assumed to result in $\perp$. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}, \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}}) \leq Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{D}\mathbf{C}^{\mathsf{Forge\text{-}Invalid}}).$$

$\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}} \rightsquigarrow \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$: Systems $\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}}$ and $\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$ are perfectly indistinguishable. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\mathsf{\{IND,IK\}\text{-}CCA\text{-}2}}, \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}) = 0.$$

$\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} \rightsquigarrow \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{S}$**:** Analogous to step $\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}} \rightsquigarrow \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{T}$ in the proof of Theorem 11. It follows

$$\Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{S}) \leq \varepsilon.$$

Once again we conclude by using triangle inequality:

$$\begin{aligned}
\Delta^{\mathbf{D}}(\mathbf{R}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{S}) \leq{}& \Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons\text{-}H}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Cons}}\mathbf{G}^{\mathsf{Cons}}, \mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Corr}}\mathbf{G}^{\mathsf{Corr}}, \mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{Forge\text{-}Invalid}}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}}, \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{0}}^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}}, \mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{CCA}}\mathbf{G}_{\mathbf{1}}^{\{\mathsf{IND,IK}\}\text{-}\mathsf{CCA\text{-}2}}, \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}, \mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}) \\
&+ \Delta^{\mathbf{D}}(\mathbf{C}^{\mathsf{OTR}}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}, \mathsf{sim}^{\overline{\mathcal{P}^H}}\mathbf{S}) \\
\leq{}& 4 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}) \\
&\quad + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1})\Big) \\
&+ Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons\text{-}0}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons\text{-}1}\text{-}\xi_2}) \\
&+ Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}}) \\
&+ Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}}) \\
&+ \varepsilon + Adv^{\{\mathsf{IND,\,IK}\}\text{-}\mathsf{CCA\text{-}2}}(\mathbf{DC}^{\mathsf{CCA}}) + Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}}).
\end{aligned}$$

□

H.1.4 Proof of Helper Claim: Lemma 2 Consider adversary $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ interacting with $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$: if event ${\xi_1}'$ occurs[36] $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ wins the game; if ${\xi_1}'$ does not occur, $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ wins the game with probability 1/2. Now, suppose $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ interacts with $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$: if ${\xi_1}'$ does not occur $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ wins the game with probability 1/2. If event ${\xi_1}'$ occurs then $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ does not win $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$; however, one can bound the probability of event ${\xi_1}'$ occurring (when $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}$ is interacting with $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$) by the probability that $\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}$ wins the correctness game plus the probability that $\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}$ wins the Forgery Invalidity game. It follows

$$\begin{aligned}
\Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} \neq \mathtt{win}] \leq \frac{1}{2} &+ \Pr[\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}\mathbf{G}^{\mathsf{Corr}} = \mathtt{win}] \\
&+ \Pr[\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \mathtt{win}].
\end{aligned}$$

[36] See Algorithm 49 for a definition of event ${\xi_1}'$.

By Definition 20,

$$\begin{aligned}
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) &:= \Big|\Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} = \texttt{win}] + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}] - 1\Big| \\
&= \Big|\Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} = \texttt{win} \mid {\xi_1}'] \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \\
&\quad + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} = \texttt{win} \mid \neg{\xi_1}'] \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow \neg{\xi_1}'] \\
&\quad + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}] - 1\Big| \\
&= \Big|\Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] + \frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow \neg{\xi_1}'] \\
&\quad + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}] - 1\Big| \\
&= \Big|\Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \cdot \frac{1}{2} - \frac{1}{2} + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}]\Big|.
\end{aligned}$$

We consider the two possible cases:

$$Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) = \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \cdot \frac{1}{2} - \frac{1}{2} + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}],$$

and

$$Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) = \frac{1}{2} - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \cdot \frac{1}{2} - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}].$$

For the first case, we have

$$\begin{gathered}
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) = \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \cdot \frac{1}{2} - \frac{1}{2} + \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}] \\
\Leftrightarrow \\
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + \frac{1}{2} - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \texttt{win}] = \frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \\
\Rightarrow \\
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + \frac{1}{2} - \Bigg(\frac{1}{2} - \Big(\Pr[\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}\mathbf{G}^{\mathsf{Corr}} = \texttt{win}] + \\
\Pr[\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \texttt{win}]\Big)\Bigg) \geq \frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \\
\Leftrightarrow \\
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + \Pr[\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}\mathbf{G}^{\mathsf{Corr}} = \texttt{win}] \\
+ \Pr[\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \texttt{win}] \geq \frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'].
\end{gathered}$$

For the second, we have

$$
\begin{aligned}
Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) = \frac{1}{2} - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \cdot \frac{1}{2} - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \mathtt{win}] \\
\Leftrightarrow \\
\frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] = \frac{1}{2} - Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \mathtt{win}] \\
\Rightarrow \\
\frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \leq \frac{1}{2} + Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) - \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}} = \mathtt{win}] \\
\Rightarrow \\
\frac{1}{2} \cdot \Pr[\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}} \Rightarrow {\xi_1}'] \leq Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + \Pr[\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}\mathbf{G}^{\mathsf{Corr}} = \mathtt{win}] \\
+ \Pr[\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \mathtt{win}].
\end{aligned}
$$

Putting things together one can then upper bound the probability that ${\xi_1}'$ occurs by

$$
\begin{aligned}
& 2 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + \Pr[\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}\mathbf{G}^{\mathsf{Corr}} = \mathtt{win}] \\
& \qquad + \Pr[\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}\mathbf{G}^{\mathsf{Forge\text{-}Invalid}} = \mathtt{win}] \Big) \\
= & 2 \cdot \Big(Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_1}) + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_1}) \\
& \qquad + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_1}) \Big).
\end{aligned}
$$

To conclude, note that the probability for event ${\xi_1}'$ to occur is half of the probability that event ξ_1 occurs. □

H.1.5 Proof of Helper Claim: Lemma 3 The proof of this result follows similar lines to the proof of Lemma 2; in the following, events $\xi_{2,0}$ and $\xi_{2,1}$ are as defined in Algorithm 52.

Interacting with $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$*:* First, consider adversary $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ interacting with $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$: if $\xi_{2,0}$ occurs $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ wins the game; if $\xi_{2,1}$ occurs, it does not win the game; and otherwise it wins the game with probability $1/2$. We can bound the probability that $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ does not win $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$ due to event $\xi_{2,1}$ occurring by reducing to winning either the consistency or the correctness games. Concretely, we bound the probability of $\xi_{2,1}$ occurring when $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ is interacting with $\mathbf{G}_{\mathbf{0}}^{\mathsf{OTR}}$ by

$$Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons\text{-}0}\text{-}\xi_2}) + Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_2}).$$

Interacting with $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$*:* Conversely, consider $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ is now interacting with $\mathbf{G}_{\mathbf{1}}^{\mathsf{OTR}}$: if $\xi_{2,0}$ occurs $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ does not win; if $\xi_{2,1}$ occurs, it wins the game, and otherwise it wins the game with probability $1/2$. As before we bound the probability that $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ does not win $\mathbf{G}_\mathbf{1}^{\mathsf{OTR}}$ due to event $\xi_{2,0}$ occurring by reducing to winning either the consistency or the forgery invalidity games. This means the probability of $\xi_{2,0}$ occurring when $\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}$ is interacting with $\mathbf{G}_\mathbf{1}^{\mathsf{OTR}}$ is bounded by

$$Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}\text{-}1\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}).$$

Obtaining the final bound: Putting these facts together then allows to upper bound the probability of event ξ_2 occurring:

$$\begin{aligned}&Adv^{\mathsf{OTR}}(\mathbf{DC}^{\mathsf{OTR}\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}\text{-}0\text{-}\xi_2}) + Adv^{\mathsf{Cons}}(\mathbf{DC}^{\mathsf{Cons}\text{-}1\text{-}\xi_2})\\ &+ Adv^{\mathsf{Corr}}(\mathbf{DC}^{\mathsf{Corr}\text{-}\xi_2}) + Adv^{\mathsf{Forge\text{-}Invalid}}(\mathbf{DC}^{\mathsf{Forge\text{-}Invalid}\text{-}\xi_2}).\end{aligned}$$

□

Algorithm 30 MDVS construction $\Pi^{\mathsf{adap}}_{\mathrm{MDVS}}$ from [19]. The building blocks are a PKE scheme $\Pi_{\mathrm{PKE}} = (G, E, D)$, a One Way Function $\Pi_{\mathrm{OWF}} = (S, F)$, and a Non Interactive Zero Knowledge scheme $\Pi_{\mathrm{NIZK}} = (G, P, V, S \coloneqq (S_G, S_P))$.

```
$S(1^k)$
  $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Pi_{\mathrm{PKE}}.G(1^k)$
  **return** $\mathsf{pp} \coloneqq (1^k, \mathsf{crs} \leftarrow \Pi_{\mathrm{NIZK}}.G(1^k), \mathsf{pk})$

$G_S(\mathsf{pp})$
  $(x_0, x_1) \leftarrow (\Pi_{\mathrm{OWF}}.S(1^k), \Pi_{\mathrm{OWF}}.S(1^k))$
  $(y_0, y_1) \leftarrow (\Pi_{\mathrm{OWF}}.F(x_0), \Pi_{\mathrm{OWF}}.F(x_1))$
  $b \leftarrow \mathit{RandomCoin}$
  **return** $(\mathsf{spk} \coloneqq (y_0, y_1), \mathsf{ssk} \coloneqq (\mathsf{spk}, x \coloneqq x_b))$

$G_V(\mathsf{pp})$
  $((\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1)) \leftarrow (\Pi_{\mathrm{PKE}}.G(1^k), \Pi_{\mathrm{PKE}}.G(1^k))$
  $(x_0, x_1) \leftarrow (\Pi_{\mathrm{OWF}}.S(1^k), \Pi_{\mathrm{OWF}}.S(1^k))$
  $(y_0, y_1) \leftarrow (\Pi_{\mathrm{OWF}}.F(x_0), \Pi_{\mathrm{OWF}}.F(x_1))$
  $b \leftarrow \mathit{RandomCoin}$
  **return** $(\mathsf{vpk} \coloneqq (\mathsf{pk}_0, y_0, \mathsf{pk}_1, y_1), \mathsf{vsk} \coloneqq (\mathsf{vpk}, b, \mathsf{sk} \coloneqq \mathsf{sk}_b, x \coloneqq x_b))$

$\mathit{Sig}_{\mathsf{pp}}(\mathsf{ssk}, \vec{v} \coloneqq (\mathsf{vpk}_1, \ldots, \mathsf{vpk}_{|\vec{v}|}), m)$
  **for** $i \in \{1, \ldots, |\vec{v}|\}$ **:**
    $(c_{i,0}, c_{i,1}) \leftarrow (\Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_0}(1; r_{i,0}), \Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_1}(1; r_{i,1}))$
  $(\vec{c}, \vec{r}) \leftarrow (((c_{1,0}, c_{1,1}), \ldots, (c_{|\vec{v}|,0}, c_{|\vec{v}|,1})), ((r_{1,0}, r_{1,1}), \ldots, (r_{|\vec{v}|,0}, r_{|\vec{v}|,1})))$
  $\vec{\alpha} \leftarrow (\alpha_1 \coloneqq (1, \mathsf{ssk}.x), \ldots, \alpha_{|\vec{v}|} \coloneqq (1, \mathsf{ssk}.x))$
  $c_{\mathsf{pp}} \leftarrow \Pi_{\mathrm{PKE}}.E_{\mathsf{pp}.\mathsf{pk}}((m, 1, \vec{\alpha}); r_{\mathsf{pp}})$
  $p \leftarrow \Pi_{\mathrm{NIZK}}.P_{\mathsf{crs}}\big((\mathsf{pp}.\mathsf{pk}, \mathsf{spk}, \vec{v}, m, \vec{c}, c_{\mathsf{pp}}) \in L_{\mathrm{MDVS}^{\mathsf{adap}}}, (\vec{\alpha}, \vec{r}, r_{\mathsf{pp}}, 1)\big)$
  **return** $\sigma \coloneqq (p, \vec{c}, c_{\mathsf{pp}})$

$\mathit{Vfy}_{\mathsf{pp}}(\mathsf{spk}, \mathsf{vsk}, \vec{v}, m, \sigma \coloneqq (p, \vec{c}, c_{\mathsf{pp}}))$
  **if** $\Pi_{\mathrm{NIZK}}.V_{\mathsf{crs}}\big((\mathsf{pp}.\mathsf{pk}, \mathsf{spk}, \vec{v}, m, \vec{c}, c_{\mathsf{pp}}) \in L_{\mathrm{MDVS}^{\mathsf{adap}}}, p\big) = 1$ **:**
    **for** $i = 1, \ldots, |\vec{v}|$ **do**
      **if** $\mathsf{vsk}.\mathsf{vpk} = v_i$ **:**
        **return** $\Pi_{\mathrm{PKE}}.D_{\mathsf{vsk}.\mathsf{sk}}(c_{i,\mathsf{vsk}.b})$
  **return** $0$

$\mathit{Forge}_{\mathsf{pp}}(\mathsf{spk}, \vec{v} \coloneqq (\mathsf{vpk}_1, \ldots, \mathsf{vpk}_{|\vec{v}|}), m, \vec{s} \coloneqq (\mathsf{vsk}_1, \ldots, \mathsf{vsk}_{|\vec{s}|}))$ // Assumed: $|\vec{v}| = |\vec{s}|$
  **for** $i \in \{1, \ldots, |\vec{v}|\}$ **:**
    **if** $s_i \neq \bot$ **:**
      $(c_{i,0}, c_{i,1}) \leftarrow (\Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_0}(1; r_{i,0}), \Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_1}(1; r_{i,1}))$
      $\alpha_i \coloneqq (1, \mathsf{vsk}_i.x)$
    **else**
      $(c_{i,0}, c_{i,1}) \leftarrow (\Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_0}(0; r_{i,0}), \Pi_{\mathrm{PKE}}.E_{\mathsf{vpk}_i.\mathsf{pk}_1}(0; r_{i,1}))$
      $\alpha_i \coloneqq (0, 0)$
  $(\vec{c}, \vec{r}) \leftarrow (((c_{1,0}, c_{1,1}), \ldots, (c_{|\vec{v}|,0}, c_{|\vec{v}|,1})), ((r_{1,0}, r_{1,1}), \ldots, (r_{|\vec{v}|,0}, r_{|\vec{v}|,1})))$
  $\vec{\alpha} \leftarrow (\alpha_1, \ldots, \alpha_{|\vec{v}|})$
  $c_{\mathsf{pp}} \leftarrow \Pi_{\mathrm{PKE}}.E_{\mathsf{pp}.\mathsf{pk}}((m, 0, \vec{\alpha}); r_{\mathsf{pp}})$
  $p \leftarrow \Pi_{\mathrm{NIZK}}.P_{\mathsf{crs}}\big((\mathsf{pp}.\mathsf{pk}, \mathsf{spk}, \vec{v}, m, \vec{c}, c_{\mathsf{pp}}) \in L_{\mathrm{MDVS}^{\mathsf{adap}}}, (\vec{\alpha}, \vec{r}, r_{\mathsf{pp}}, 0)\big)$
  **return** $\sigma \coloneqq (p, \vec{c}, c_{\mathsf{pp}})$
```

Algorithm 31 MDRS-PKE construction $\Pi_{\text{MDRS-PKE}}$ from [35]. The building blocks are a PKEBC scheme $\Pi_{\text{PKEBC}} = (S, G, E, D)$, an MDVS scheme $\Pi_{\text{MDVS}} = (S, G_S, G_V, \mathit{Sig}, \mathit{Vfy}, \mathit{Forge})$ and a DSS $\Pi_{\text{DSS}} = (G, \mathit{Sig}, \mathit{Vfy})$.

```
$S(1^k)$
  $\mathtt{pp}_{\text{MDVS}} \leftarrow \Pi_{\text{MDVS}}.S(1^k)$
  $\mathtt{pp}_{\text{PKEBC}} \leftarrow \Pi_{\text{PKEBC}}.S(1^k)$
  **return** $\mathtt{pp} \coloneqq (\mathtt{pp}_{\text{MDVS}}, \mathtt{pp}_{\text{PKEBC}}, 1^k)$

$G_S(\mathtt{pp})$
  $(\mathtt{spk}_{\text{MDVS}}, \mathtt{ssk}_{\text{MDVS}}) \leftarrow \Pi_{\text{MDVS}}.G_S(\mathtt{pp}_{\text{MDVS}})$
  **return** $(\mathtt{spk} \coloneqq \mathtt{spk}_{\text{MDVS}}, \mathtt{ssk} \coloneqq (\mathtt{spk}, \mathtt{ssk}_{\text{MDVS}}))$

$G_R(\mathtt{pp})$
  $(\mathtt{vpk}_{\text{MDVS}}, \mathtt{vsk}_{\text{MDVS}}) \leftarrow \Pi_{\text{MDVS}}.G_V(\mathtt{pp}_{\text{MDVS}})$
  $(\mathtt{pk}_{\text{PKEBC}}, \mathtt{sk}_{\text{PKEBC}}) \leftarrow \Pi_{\text{PKEBC}}.G(\mathtt{pp}_{\text{PKEBC}})$
  **return** $(\mathtt{rpk} \coloneqq (\mathtt{vpk}_{\text{MDVS}}, \mathtt{pk}_{\text{PKEBC}}), \mathtt{rsk} \coloneqq (\mathtt{rpk}, (\mathtt{vsk}_{\text{MDVS}}, \mathtt{sk}_{\text{PKEBC}})))$

$E_{\mathtt{pp}}(\mathtt{ssk}, \vec{v} \coloneqq (\mathtt{rpk}_1, \dots, \mathtt{rpk}_{|\vec{v}|}), m)$
  $\vec{v}_{\text{PKEBC}} \coloneqq (\mathtt{rpk}_1.\mathtt{pk}_{\text{PKEBC}}, \dots, \mathtt{rpk}_{|\vec{v}|}.\mathtt{pk}_{\text{PKEBC}})$
  $\vec{v}_{\text{MDVS}} \coloneqq (\mathtt{rpk}_1.\mathtt{vpk}_{\text{MDVS}}, \dots, \mathtt{rpk}_{|\vec{v}|}.\mathtt{vpk}_{\text{MDVS}})$
  $(\mathtt{vk}, \mathtt{sk}) \leftarrow \Pi_{\text{DSS}}.G(\mathtt{pp}.1^k)$
  $\sigma \leftarrow \Pi_{\text{MDVS}}.\mathit{Sig}_{\mathtt{pp}_{\text{MDVS}}}(\mathtt{ssk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \mathtt{vk}))$
  $c \leftarrow \Pi_{\text{PKEBC}}.E_{\mathtt{pp}_{\text{PKEBC}}}(\vec{v}_{\text{PKEBC}}, (\mathtt{spk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, m, \sigma))$
  $\sigma' \leftarrow \Pi_{\text{DSS}}.\mathit{Sig}_{\mathtt{sk}}(c)$
  **return** $(\mathtt{vk}, \sigma', c)$

$D_{\mathtt{pp}}(\mathtt{rsk}, c \coloneqq (\mathtt{vk}, \sigma', c'))$
  **if** $\Pi_{\text{DSS}}.\mathit{Vfy}_{\mathtt{vk}}(c', \sigma') = 0$ **:**
    **return** $\perp$
  $(\vec{v}_{\text{PKEBC}}, (\mathtt{spk} \coloneqq \mathtt{spk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, m, \sigma)) \leftarrow \Pi_{\text{PKEBC}}.D_{\mathtt{pp}_{\text{PKEBC}}}(\mathtt{rsk}.\mathtt{sk}_{\text{PKEBC}}, c')$
  **if** $(\vec{v}_{\text{PKEBC}}, (\mathtt{spk}, \vec{v}_{\text{MDVS}}, m, \sigma)) = \perp \;\vee\; |\vec{v}_{\text{PKEBC}}| \neq |\vec{v}_{\text{MDVS}}|$ **:**
    **return** $\perp$
  $\vec{v} \coloneqq ((v_{\text{MDVS}1}, v_{\text{PKEBC}1}), \dots, (v_{\text{MDVS}|\vec{v}_{\text{PKEBC}}|}, v_{\text{PKEBC}|\vec{v}_{\text{PKEBC}}|}))$
  **if** $\mathtt{rsk}.\mathtt{rpk} \notin \vec{v}$ **:**
    **return** $\perp$
  **if** $\Pi_{\text{MDVS}}.\mathit{Vfy}_{\mathtt{pp}_{\text{MDVS}}}(\mathtt{spk}, \mathtt{vsk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \mathtt{vk}), \sigma) \neq \mathtt{valid}$ **:**
    **return** $\perp$
  **return** $(\mathtt{spk}, \vec{v}, m)$

$\mathit{Forge}_{\mathtt{pp}}(\mathtt{spk}, \vec{v} \coloneqq (\mathtt{rpk}_1, \dots, \mathtt{rpk}_{|\vec{v}|}), m, \vec{s} \coloneqq (\mathtt{rsk}_1, \dots, \mathtt{rsk}_{|\vec{s}|}))$
  $\vec{v}_{\text{PKEBC}} \coloneqq (\mathtt{rpk}_1.\mathtt{pk}_{\text{PKEBC}}, \dots, \mathtt{rpk}_{|\vec{v}|}.\mathtt{pk}_{\text{PKEBC}})$
  $\vec{v}_{\text{MDVS}} \coloneqq (\mathtt{rpk}_1.\mathtt{vpk}_{\text{MDVS}}, \dots, \mathtt{rpk}_{|\vec{v}|}.\mathtt{vpk}_{\text{MDVS}})$
  $\vec{s}_{\text{MDVS}} \coloneqq (\mathtt{rsk}_1.\mathtt{vsk}_{\text{MDVS}}, \dots, \mathtt{rsk}_{|\vec{s}|}.\mathtt{vsk}_{\text{MDVS}})$
  $(\mathtt{vk}, \mathtt{sk}) \leftarrow \Pi_{\text{DSS}}.G(\mathtt{pp}.1^k)$
  $\sigma \leftarrow \Pi_{\text{MDVS}}.\mathit{Forge}_{\mathtt{pp}_{\text{MDVS}}}(\mathtt{spk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, (\vec{v}_{\text{PKEBC}}, m, \mathtt{vk}), \vec{s}_{\text{MDVS}})$
  $c \leftarrow \Pi_{\text{PKEBC}}.E_{\mathtt{pp}_{\text{PKEBC}}}(\vec{v}_{\text{PKEBC}}, (\mathtt{spk}_{\text{MDVS}}, \vec{v}_{\text{MDVS}}, m, \sigma))$
  $\sigma' \leftarrow \Pi_{\text{DSS}}.\mathit{Sig}_{\mathtt{sk}}(c)$
  **return** $(\mathtt{vk}, \sigma', c)$
```

Algorithm 32 Description of (part of) the simulators considered in the proofs of Theorems 11 and 12 for the (sub-)interfaces of (dishonest) parties that correspond to an interface of the **KGA** resource in the real world system. In the following, $k \in \mathbb{N}$ is the (implicitly defined) security parameter.

```
Initialization
  INS-Initialization
  Ctxts ← $\emptyset$
  CtxtSet ← $\emptyset$
  $\mathtt{pp} \leftarrow \mathit{\Pi}.S(1^k)$
  $(\mathtt{rpk}_{\mathtt{pp}}, \mathtt{rsk}_{\mathtt{pp}}) \leftarrow \mathit{\Pi}.G_R(\mathtt{pp})$
  for $A_i \in \mathcal{S}$ :
    $(\mathtt{spk}_i, \mathtt{ssk}_i) \leftarrow \mathit{\Pi}.G_S(\mathtt{pp})$
  for $B_j \in \mathcal{R}$ :
    $(\mathtt{rpk}_j, \mathtt{rsk}_j) \leftarrow \mathit{\Pi}.G_R(\mathtt{pp})$

$(P \in \overline{\mathcal{P}^H})$-PublicParameters
  Output$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}})$

$(P \in \overline{\mathcal{P}^H})$-SenderKeyPair$(A_i \in \overline{\mathcal{S}^H})$
  Output$(\mathtt{spk}_i, \mathtt{ssk}_i)$

$(P \in \overline{\mathcal{P}^H})$-SenderPublicKey$(A_i \in \mathcal{S})$
  Output$(\mathtt{spk}_i)$

$(P \in \overline{\mathcal{P}^H})$-ReceiverKeyPair$(B_j \in \overline{\mathcal{R}^H})$
  Output$(\mathtt{rpk}_j, \mathtt{rsk}_j)$

$(P \in \overline{\mathcal{P}^H})$-ReceiverPublicKey$(B_j \in \mathcal{R})$
  Output$(\mathtt{rpk}_j)$


GetLabel$(\mathtt{spk}, \vec{v}')$  // Local procedure. Not available at outside interface.
  $\mathcal{S}_{\mathtt{spk}} := \{A_i \mid \mathtt{spk} = \mathtt{spk}_i\}$
  if $|\mathcal{S}_{\mathtt{spk}}| \neq 1 \vee {v_1}' \neq \mathtt{rpk}_{\mathtt{pp}}$ :
    return $\bot$
  for $l \in \{2, \dots, |\vec{v}'|\}$ :
    $\mathcal{R}_{{v_l}'} := \{B_k \mid {v_l}' = \mathtt{rpk}_k\}$
    if $|\mathcal{R}_{{v_l}'}| \neq 1$ :
      return $\bot$
    else
      Let $B_k$ be the element of $\mathcal{R}_{{v_l}'}$
      $V_{l-1} = B_k$
  Let $A_i$ be the element of $\mathcal{S}_{\mathtt{spk}}$
  Let $\vec{V} := (V_1, \dots, V_{|\vec{v}'|-1})$
  return $\langle A_i \to \vec{V} \rangle$
```

Algorithm 33 Description of the reductions considered in the proofs of Lemmata 2 and 3 and Theorems 11 and 12 for the (sub-)interfaces of (dishonest) parties that correspond to **KGA** interface in the real world system, plus the Deliver interface.

```
Initialization
  INS-Initialization
  Dec ← ∅
  CtxtDec ← ∅
  CtxtHon ← ∅
  CtxtHonForge ← ∅
  CtxtDis ← ∅
  $(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}}) \leftarrow (\mathcal{O}_{PP}, \mathcal{O}_{RPK}(B_{\mathtt{pp}}))$
  for $A_i \in \mathcal{S}$ :
     $\mathcal{O}_{SPK}(A_i)$
  for $B_j \in \mathcal{R}$ :
     $\mathcal{O}_{RPK}(B_j)$
     Received$[B_j] \leftarrow \emptyset$

$(P \in \overline{\mathcal{P}^H})$-PublicParameters
  Output$(\mathtt{pp}, \mathtt{rpk}_{\mathtt{pp}})$

$(P \in \overline{\mathcal{P}^H})$-SenderKeyPair$(A_i \in \overline{\mathcal{S}^H})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-SenderPublicKey$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SPK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-ReceiverKeyPair$(B_j \in \overline{\mathcal{R}^H})$
  Output$(\mathcal{O}_{RK}(B_j))$

$(P \in \overline{\mathcal{P}^H})$-ReceiverPublicKey$(B_j \in \mathcal{R})$
  Output$(\mathcal{O}_{RPK}(B_j))$

Deliver$(P, \mathtt{id})$
  Received$[P] \leftarrow$ Received$[P] \cup \{\mathtt{id}\}$


GetLabel$(\mathtt{spk}, \vec{v}')$  // Local procedure. Not available at outside interface.
  $\mathcal{S}_{\mathsf{spk}} := \{A_i \mid \mathtt{spk} = \mathtt{spk}_i\}$
  if $|\mathcal{S}_{\mathsf{spk}}| \neq 1 \vee {v_1}' \neq \mathtt{rpk}_{\mathtt{pp}}$ :
     return $\perp$
  for $l \in \{2, \ldots, |\vec{v}'|\}$ :
     $\mathcal{R}_{{v_l}'} := \{B_k \mid {v_l}' = \mathtt{rpk}_k\}$
     if $|\mathcal{R}_{{v_l}'}| \neq 1$ :
        return $\perp$
     else
        Let $B_k$ be the element of $\mathcal{R}_{{v_l}'}$
        $V_{l-1} = B_k$
  Let $A_i$ be the element of $\mathcal{S}_{\mathsf{spk}}$
  Let $\vec{V} := (V_1, \ldots, V_{|\vec{v}'|-1})$
  return $\langle A_i \to \vec{V} \rangle$
```

Algorithm 34 Helper functions used in the reductions considered in the proofs of Lemmata 2 and 3 and Theorems 11 and 12.

```
Decryption$(B, c)$  // Local procedure. Not available at outside interface.
  $(\texttt{spk}, \vec{v}', m) \leftarrow \mathcal{O}_D(B, c)$
  if $(\texttt{spk}, \vec{v}', m) \neq \perp$ :
    $\langle A_i \rightarrow \vec{V} \rangle \leftarrow$ GetLabel$(\texttt{spk}, \vec{v}')$
    if $\langle A_i \rightarrow \vec{V} \rangle \neq \perp$ :
      return $(\langle A_i \rightarrow \vec{V} \rangle, m)$
  return $\perp$

GetDelivered$(P, \text{list})$  // Local procedure. Not available at outside interface.
  $\text{filteredList} \leftarrow \emptyset$
  for $(\texttt{id}, x) \in \text{list}$ with $\texttt{id} \in \text{Received}[P]$ :
    $\text{filteredList} \leftarrow \text{filteredList} \cup \{(\texttt{id}, x)\}$
  return filteredList

Forge$(A_i, \vec{V}, m, \mathcal{C})$  // Local procedure. Not available at outside interface.
  if $\vec{V} \in (\mathcal{R}^H)^+$ :
    return $\Pi.Forge_{\texttt{pp}}(\texttt{spk}_1, {\texttt{rpk}_{\texttt{pp}}}^{|\vec{V}|+1}, 0^{|m|}, \perp^{|\vec{V}|+1})$
  $\vec{v}' := (\texttt{rpk}_{\texttt{pp}}, v_1, \ldots, v_{|\vec{V}|})$
  $\vec{s} := (\perp \quad , \texttt{rsk}_1, \ldots, \texttt{rsk}_{|\vec{V}|})$  // For each $V_l$: if $V_l \in \mathcal{C}$, $s_{l+1}$ is $V_l$'s secret key; else it is $\perp$.
  return $\Pi.Forge_{\texttt{pp}}(\texttt{spk}_i, \vec{v}', m, \vec{s})$
```

Algorithm 35 Description of the behavior of the simulator considered in the proof of Theorem 11 for the (sub-)interfaces of dishonest parties that correspond to an interface of $\mathsf{Net} \cdot \mathbf{INS}$ in the real world system. In the following, $\mathbf{T}$ is as defined in Equations 3.5 and H.2.

```
$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtSet :
    CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
    $(\texttt{spk}, \vec{v}', m) \leftarrow \Pi.D_{\texttt{pp}}(\texttt{rsk}_{\texttt{pp}}, c)$
    if $(\texttt{spk}, \vec{v}', m) \neq \bot$ :
      $\langle A_i \to \vec{V} \rangle \leftarrow$ GetLabel$(\texttt{spk}, \vec{v}')$
      if $\langle A_i \to \vec{V} \rangle \neq \bot \wedge A_i \in \overline{\mathcal{S}^H}$ :
        $\texttt{id} \leftarrow \mathbf{T}$-Write$(\langle A_i \to \vec{V} \rangle, m)$
        Ctxts$[\texttt{id}] \leftarrow c$  // Add entry to map Ctxts.
        Output$(\texttt{id})$
  Output$(\mathbf{INS}$-Write$(c))$

$(P \in \overline{\mathcal{P}^H})$-Read
  outputList $\leftarrow \emptyset$
  for $(\langle A_i \to \vec{V} \rangle, \texttt{id}, m) \in \mathbf{T}$-Read :
    if $\texttt{id} \notin$ Ctxts :  // Check existence of entry with given key in map Ctxts.
      $\vec{s} := (\bot, \texttt{rsk}_1, \ldots, \texttt{rsk}_{|\vec{V}|})$  // For each $V_l$: if $V_l \in \overline{\mathcal{R}^H}$, $s_{l+1}$ is $V_l$'s secret key; else $\bot$.
      $c \leftarrow \Pi.Forge_{\texttt{pp}}(\texttt{spk}_i, \vec{v}' := (\texttt{rpk}_{\texttt{pp}}, v_1, \ldots, v_{|\vec{v}|}), m, \vec{s})$
      if $c \in$ CtxtSet :
        Abort
      CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
      Ctxts$[\texttt{id}] \leftarrow c$  // Add entry to map Ctxts.
    outputList $\leftarrow$ outputList $\cup \{(\texttt{id}, \text{Ctxts}[\texttt{id}])\}$ // Fetch value of entry from map Ctxts.
  for $(l \in \mathbb{N}, \texttt{id}, l' \in \mathbb{N}) \in \mathbf{T}$-Read :
    if $\texttt{id} \notin$ Ctxts :
      $c \leftarrow \Pi.Forge_{\texttt{pp}}(\texttt{spk}_1, {\texttt{rpk}_{\texttt{pp}}}^{l+1}, 0^{l'}, \bot^{l+1})$
      if $c \in$ CtxtSet :
        Abort
      CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
      Ctxts$[\texttt{id}] \leftarrow c$
    outputList $\leftarrow$ outputList $\cup \{(\texttt{id}, \text{Ctxts}[\texttt{id}])\}$
  Output(outputList $\cup$ $\mathbf{INS}$-Read)
```

Algorithm 36 Reduction $\mathbf{C}^{\mathsf{Cons\text{-}H}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' \coloneqq (B_{\mathsf{pp}}, V_1, \dots, V_{|\vec{V}|}), m\big)$
  if $c \in \text{CtxtHon}$ : // Event $\xi_1$.
    Abort // Ciphertext Already Exists
  $\text{CtxtHon} \leftarrow \text{CtxtHon} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\texttt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\texttt{id})$

$(P \in \mathcal{F})$-$\textsc{Write}(\langle [\text{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \textsc{Forge}\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  $\text{CtxtHonForge} \leftarrow \text{CtxtHonForge} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\texttt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\texttt{id})$

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  $\text{outputList} \leftarrow \emptyset$
  $\text{ciphertextSet} \leftarrow \emptyset$
  for $(\texttt{id}, c) \in \textsc{Read}$ with $c \notin \text{ciphertextSet}$ :
    $\text{ciphertextSet} \leftarrow \text{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow \textsc{Decryption}(B_j, c)$
    if $(\langle A_i \to \vec{V} \rangle, m) \neq \bot$ :
      $\text{outputList} \leftarrow \text{outputList} \cup \{(\texttt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \text{outputList}))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  if $c \notin \text{CtxtHon} \cup \text{CtxtHonForge}$ :
    $\text{CtxtDis} \leftarrow \text{CtxtDis} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\texttt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\texttt{id})$

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

Algorithm 37 Reduction $\mathbf{C}^{\mathsf{Cons}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  if $c \in$ CtxtHon $\cup$ CtxtDis : // Event $\xi_1$ or event $\xi_2$.
    Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(P \in \mathcal{F})$-Write$(\langle [\text{Forge}] A_i \to \vec{V}\rangle, m)$
  $c \leftarrow$ Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for (id, $c$) $\in$ Read with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow$ Decryption$(B_j, c)$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \perp$ :
      outputList $\leftarrow$ outputList $\cup \{($id$, (\langle A_i \to \vec{V}\rangle, m))\}$
  Output(GetDelivered$(B_j$, outputList))

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
    CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

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

Algorithm 38 Reduction $\mathbf{C}^{\mathsf{Corr}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' \coloneqq (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  **if** $c \in \mathrm{CtxtHon} \cup \mathrm{CtxtDis}$ **:** // Event $\xi_1$ or event $\xi_2$.
    Abort // Ciphertext Already Exists
  $\mathrm{CtxtHon} \leftarrow \mathrm{CtxtHon} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\mathtt{id})$

$(P \in \mathcal{F})$-$\textsc{Write}(\langle [\mathrm{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \textsc{Forge}\big(A_i, \vec{V}, m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  $\mathrm{CtxtHonForge} \leftarrow \mathrm{CtxtHonForge} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\mathtt{id})$

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  $\mathrm{outputList} \leftarrow \emptyset$
  $\mathrm{ciphertextSet} \leftarrow \emptyset$
  **for** $(\mathtt{id}, c) \in \textsc{Read}$ **with** $c \notin \mathrm{ciphertextSet}$ **:**
    $\mathrm{ciphertextSet} \leftarrow \mathrm{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow \mathrm{CtxtDec}[\mathtt{id}]$
    **if** $(\langle A_i \to \vec{V} \rangle, m) \neq \bot \wedge B_j \in \vec{V}$ **:**
      $\mathrm{outputList} \leftarrow \mathrm{outputList} \cup \{(\mathtt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \mathrm{outputList}))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  **if** $c \notin \mathrm{CtxtHon} \cup \mathrm{CtxtHonForge}$ **:**
    $\mathrm{CtxtDis} \leftarrow \mathrm{CtxtDis} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}(\mathtt{id})$

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

Algorithm 39 Reduction $\mathbf{C}^{\textsf{R-Unforg}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathtt{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  if $c \in \text{CtxtHon} \cup \text{CtxtDis}$ : // Event $\xi_1$ or event $\xi_2$.
    Abort // Ciphertext Already Exists
  $\text{CtxtHon} \leftarrow \text{CtxtHon} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\mathtt{id}] \leftarrow (\langle A_i \to \vec{V} \rangle, m)$
  $\textsc{Output}(\mathtt{id})$

$(P \in \mathcal{F})$-$\textsc{Write}(\langle [\text{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \textsc{Forge}\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  $\text{CtxtHonForge} \leftarrow \text{CtxtHonForge} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathtt{pp}}, c)$
  $\textsc{Output}(\mathtt{id})$

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  $\text{outputList} \leftarrow \emptyset$
  $\text{ciphertextSet} \leftarrow \emptyset$
  for $(\mathtt{id}, c) \in \textsc{Read}$ with $c \notin \text{ciphertextSet}$ :
    $\text{ciphertextSet} \leftarrow \text{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow \text{CtxtDec}[\mathtt{id}]$
    if $(\langle A_i \to \vec{V} \rangle, m) \neq \bot \wedge B_j \in \vec{V}$ :
      $\text{outputList} \leftarrow \text{outputList} \cup \{(\mathtt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \text{outputList}))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  if $c \notin \text{CtxtHon} \cup \text{CtxtHonForge}$ :
    $\text{CtxtDis} \leftarrow \text{CtxtDis} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\text{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathtt{pp}}, c)$
  $\textsc{Output}(\mathtt{id})$

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

Algorithm 40 Reduction $\mathbf{C}^{\mathsf{CCA}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V} \rangle, m)$
  $\alpha_{\mathbf{0}} \coloneqq (A_i, \vec{V}' \coloneqq (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m)$,        $\alpha_{\mathbf{1}} \coloneqq (A_1, \overbrace{(B_{\mathsf{pp}}, \ldots, B_{\mathsf{pp}})}^{|\vec{V}|+1 \text{ times}}, 0^{|m|})$
  if $\vec{V} \notin (\mathcal{R}^H)^+$ :
     $\alpha_{\mathbf{1}} \coloneqq \alpha_{\mathbf{0}}$
  $c \leftarrow \mathcal{O}_E(\alpha_{\mathbf{0}}, \alpha_{\mathbf{1}})$
  if $c \in$ CtxtHon $\cup$ CtxtDis : // Event $\xi_1$ or event $\xi_2$.
     Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow (\langle A_i \to \vec{V} \rangle, m)$
  Dec$[c] \leftarrow$ CtxtDec[id]
  Output(id)

$(P \in \mathcal{F})$-Write$(\langle [\text{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow$ Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  if $A_i \in \mathcal{S}^H$ :
     CtxtDec[id] $\leftarrow \bot$
  else // $A_i \in \overline{\mathcal{S}^H}$
     CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for (id, $c$) $\in$ Read with $c \notin$ ciphertextSet :
     ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
     $(\langle A_i \to \vec{V} \rangle, m) \leftarrow$ CtxtDec[id]
     if $(\langle A_i \to \vec{V} \rangle, m) \neq \bot \land B_j \in \vec{V}$ :
        outputList $\leftarrow$ outputList $\cup \{(\text{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  Output(GetDelivered$(B_j$, outputList))

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
     CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  if $c \in$ Dec :
     CtxtDec[id] $\leftarrow$ Dec$[c]$
  else
     $(\langle A_i \to \vec{V} \rangle, m) \leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
     if $A_i \in \mathcal{S}^H$ :
        Abort // Valid Ciphertext Forgery
     CtxtDec[id] $\leftarrow (\langle A_i \to \vec{V} \rangle, m)$
     Dec$[c] \leftarrow$ CtxtDec[id]
  Output(id)

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

Algorithm 41 Reduction $\mathbf{C}^{\mathsf{OTR}}$ for Theorem 11.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V}\rangle, m)$
  if $\vec{V} \in (\mathcal{R}^H)^+$ :
    $c \leftarrow \mathcal{O}_E(\mathtt{sig}, A_1, \overbrace{(B_{\mathtt{pp}}, \ldots, B_{\mathtt{pp}})}^{|\vec{V}|+1 \text{ times}}, 0^{|m|}, \emptyset)$
  else
    $c \leftarrow \mathcal{O}_E(\mathtt{sig}, A_i, \vec{V}' \coloneqq (B_{\mathtt{pp}}, V_1, \ldots, V_{|\vec{V}|}), m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H})$
  if $c \in \mathrm{CtxtHon} \cup \mathrm{CtxtDis}$ :
    Abort
  $\mathrm{CtxtHon} \leftarrow \mathrm{CtxtHon} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow (\langle A_i \to \vec{V}\rangle, m)$
  $\mathrm{Dec}[c] \leftarrow \mathrm{CtxtDec}[\mathtt{id}]$
  $\textsc{Output}(\mathtt{id})$

$(P \in \mathcal{F})$-$\textsc{Write}(\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \textsc{Forge}(A_i, \vec{V}, m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H})$
  $\mathrm{CtxtHonForge} \leftarrow \mathrm{CtxtHonForge} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \bot$
  $\textsc{Output}(\mathtt{id})$

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  $\mathrm{outputList} \leftarrow \emptyset$
  $\mathrm{ciphertextSet} \leftarrow \emptyset$
  for $(\mathtt{id}, c) \in \textsc{Read}$ with $c \notin \mathrm{ciphertextSet}$ :
    $\mathrm{ciphertextSet} \leftarrow \mathrm{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow \mathrm{CtxtDec}[\mathtt{id}]$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \bot \wedge B_j \in \vec{V}$ :
      $\mathrm{outputList} \leftarrow \mathrm{outputList} \cup \{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \mathrm{outputList}))$

$(J)$-$\textsc{SenderKeyPair}(A_i \in \mathcal{S})$
  $\textsc{Output}(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  if $c \notin \mathrm{CtxtHon} \cup \mathrm{CtxtHonForge}$ :
    $\mathrm{CtxtDis} \leftarrow \mathrm{CtxtDis} \cup \{c\}$
  $\mathtt{id} \leftarrow \textsc{Write}(c)$
  if $c \in \mathrm{Dec}$ :
    $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \mathrm{Dec}[c]$
  else
    $\mathrm{CtxtDec}[\mathtt{id}] \leftarrow \textsc{Decryption}(B_{\mathtt{pp}}, c)$
    $\mathrm{Dec}[c] \leftarrow \mathrm{CtxtDec}[\mathtt{id}]$
  $\textsc{Output}(\mathtt{id})$

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

Algorithm 42 Description of the behavior of the simulator considered in the proof of Theorem 12 for the (sub-)interfaces of dishonest parties in $\overline{\mathcal{P}^H}$ that correspond to an interface of $\mathsf{Net} \cdot \mathbf{INS}$ in the real world system. In the following $\mathbf{S}$ is as defined in Equation 3.2 and Equation H.4.

```
$(J)$-SenderKeyPair$(A_i \in \mathcal{S}^H)$
  Output$(\mathtt{spk}_i, \mathtt{ssk}_i)$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtSet :
    CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
    $(\mathtt{spk}, \vec{v}', m) \leftarrow \Pi.D_{\mathtt{pp}}(\mathtt{rsk}_{\mathtt{pp}}, c)$
    if $(\mathtt{spk}, \vec{v}', m) \neq \bot$ :
      $\langle A_i \to \vec{V} \rangle \leftarrow$ GetLabel$(\mathtt{spk}, \vec{v}')$
      if $\langle A_i \to \vec{V} \rangle \neq \bot$ :
        $\mathtt{id} \leftarrow \mathbf{S}$-Write$(\langle A_i \to \vec{V} \rangle, m)$
        Ctxts$[\mathtt{id}] \leftarrow c$  // Add entry to map Ctxts.
        Output$(\mathtt{id})$
  Output$(\mathbf{INS}$-Write$(c))$

$(P \in \overline{\mathcal{P}^H})$-Read
  outputList $\leftarrow \emptyset$
  for $(\langle A_i \to \vec{V} \rangle, \mathtt{id}, m) \in \mathbf{S}$-Read :
    if $\mathtt{id} \notin$ Ctxts : // $A_i \in \mathcal{S}^H$
      $\vec{s} := (\bot, \mathtt{rsk}_1, \ldots, \mathtt{rsk}_{|\vec{V}|})$  // For each $V_l$: if $V_l \in \overline{\mathcal{R}^H}$, $s_{l+1}$ is $V_l$'s secret key; else $\bot$.
      $c \leftarrow \Pi.Forge_{\mathtt{pp}}(\mathtt{spk}_i, \vec{v}' := (\mathtt{rpk}_{\mathtt{pp}}, v_1, \ldots, v_{|\vec{v}|}), m, \vec{s})$
      if $c \in$ CtxtSet : // Event $\xi_1$ or event $\xi_2$.
        Abort
      CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
      Ctxts$[\mathtt{id}] \leftarrow c$
    outputList $\leftarrow$ outputList $\cup \{(\mathtt{id}, \text{Ctxts}[\mathtt{id}])\}$
  for $(l \in \mathbb{N}, \mathtt{id}, l' \in \mathbb{N}) \in \mathbf{S}$-Read :
    if $\mathtt{id} \notin$ Ctxts : // $A_i \in \mathcal{S}^H$
      $c \leftarrow \Pi.Forge_{\mathtt{pp}}(\mathtt{spk}_1, \mathtt{rpk}_{\mathtt{pp}}{}^{l+1}, 0^{l'}, \bot^{l+1})$
      if $c \in$ CtxtSet : // Event $\xi_1$ or event $\xi_2$.
        Abort
      CtxtSet $\leftarrow$ CtxtSet $\cup \{c\}$
      Ctxts$[\mathtt{id}] \leftarrow c$
    outputList $\leftarrow$ outputList $\cup \{(\mathtt{id}, \text{Ctxts}[\mathtt{id}])\}$
  Output(outputList $\cup$ $\mathbf{INS}$-Read)
```

Algorithm 43 Reduction $\mathbf{C}^{\textsf{Cons-H}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  if $c \in$ CtxtHon : // Event $\xi_1$.
    Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow \textsc{Write}(c)$
  CtxtDec[id] $\leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}$(id)

$(P \in \mathcal{F})$-$\textsc{Write}(\langle [\text{Forge}] A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \textsc{Forge}\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow \textsc{Write}(c)$
  CtxtDec[id] $\leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}$(id)

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for $(\mathtt{id}, c) \in \textsc{Read}$ with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow \textsc{Decryption}(B_j, c)$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \perp$ :
      outputList $\leftarrow$ outputList $\cup \{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \text{outputList}))$

$(J)$-$\textsc{SenderKeyPair}(A_i \in \mathcal{S})$
  $\textsc{Output}(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
    CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow \textsc{Write}(c)$
  CtxtDec[id] $\leftarrow \textsc{Decryption}(B_{\mathsf{pp}}, c)$
  $\textsc{Output}$(id)

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

Algorithm 44 Reduction $\mathbf{C}^{\mathsf{Cons}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  if $c \in$ CtxtHon $\cup$ CtxtDis : // Event $\xi_1$ or event $\xi_2$.
    Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(P \in \mathcal{F})$-Write$(\langle[\text{Forge}]A_i \to \vec{V}\rangle, m)$
  $c \leftarrow$ Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for (id, $c$) $\in$ Read with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow$ Decryption$(B_j, c)$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \perp$ :
      outputList $\leftarrow$ outputList $\cup$ {(id, $(\langle A_i \to \vec{V}\rangle, m)$)}
  Output(GetDelivered$(B_j$, outputList))

$(J)$-SenderKeyPair$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
    CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

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

Algorithm 45 Reduction $\mathbf{C}^{\mathsf{Corr}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \dots, V_{|\vec{V}|}), m\big)$
  if $c \in$ CtxtHon $\cup$ CtxtDis : // Event $\xi_1$ or event $\xi_2$.
     Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(P \in \mathcal{F})$-Write$(\langle [\text{Forge}] A_i \to \vec{V}\rangle, m)$
  $c \leftarrow$ Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for (id, $c$) $\in$ Read with $c \notin$ ciphertextSet :
     ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
     $(\langle A_i \to \vec{V}\rangle, m) \leftarrow$ CtxtDec[id]
     if $(\langle A_i \to \vec{V}\rangle, m) \neq \perp \wedge B_j \in \vec{V}$ :
        outputList $\leftarrow$ outputList $\cup \{($id$, (\langle A_i \to \vec{V}\rangle, m))\}$
  Output(GetDelivered$(B_j$, outputList))

$(J)$-SenderKeyPair$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
     CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
  Output(id)

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

Algorithm 46 Reduction $\mathbf{C}^{\textsf{Forge-Invalid}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-WRITE$(\langle A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' \coloneqq (B_{\texttt{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  if $c \in$ CtxtHon $\cup$ CtxtDis : // Event $\xi_1$ or event $\xi_2$.
    Abort // Ciphertext Already Exists
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$WRITE$(c)$
  CtxtDec[id] $\leftarrow (\langle A_i \to \vec{V} \rangle, m)$
  OUTPUT(id)

$(P \in \mathcal{F})$-WRITE$(\langle [\text{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow$ FORGERED$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$WRITE$(c)$
  CtxtDec[id] $\leftarrow$ DECRYPTION$(B_{\texttt{pp}}, c)$
  OUTPUT(id)

$(B_j \in \mathcal{R}^H)$-READ
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for $(\texttt{id}, c) \in$ READ with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow$ CtxtDec[id]
    if $(\langle A_i \to \vec{V} \rangle, m) \neq \perp \land B_j \in \vec{V}$ :
      outputList $\leftarrow$ outputList $\cup \{(\texttt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  OUTPUT(GETDELIVERED$(B_j$, outputList$)$)

$(J)$-SENDERKEYPAIR$(A_i \in \mathcal{S})$
  OUTPUT$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-WRITE$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
    CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$WRITE$(c)$
  CtxtDec[id] $\leftarrow$ DECRYPTION$(B_{\texttt{pp}}, c)$
  OUTPUT(id)

$(P \in \overline{\mathcal{P}^H})$-READ
  OUTPUT(READ)

// The following procedure is not part of the reduction's interface.
FORGERED$(A_i, \vec{V}, m, \mathcal{C})$
  if $\vec{V} \in (\mathcal{R}^H)^+$ :
    return $\mathcal{O}_{Forge}(A_1, {B_{\texttt{pp}}}^{|\vec{V}|+1}, 0^{|m|}, \emptyset)$
  return $\mathcal{O}_{Forge}(A_i, \vec{V}' \coloneqq (B_{\texttt{pp}}, V_1, \ldots, V_{|\vec{V}|}), m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H})$
```

Algorithm 47 Reduction $\mathbf{C}^{\mathsf{CCA}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V} \rangle, m)$
  $\alpha_{\mathbf{0}} := (A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m)$,        $\alpha_{\mathbf{1}} := (A_1, \overbrace{(B_{\mathsf{pp}}, \ldots, B_{\mathsf{pp}})}^{|\vec{V}|+1 \text{ times}}, 0^{|m|})$
  if $\vec{V} \notin (\mathcal{R}^H)^+$ :
    $\alpha_{\mathbf{1}} := \alpha_{\mathbf{0}}$
  $c \leftarrow \mathcal{O}_E(\alpha_{\mathbf{0}}, \alpha_{\mathbf{1}})$
  if $c \in$ CtxtHon $\cup$ CtxtDis :
    Abort
  CtxtHon $\leftarrow$ CtxtHon $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow (\langle A_i \to \vec{V} \rangle, m)$
  Dec$[c] \leftarrow$ CtxtDec[id]
  Output(id)

$(P \in \mathcal{F})$-Write$(\langle [\text{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow$ Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtHonForge $\leftarrow$ CtxtHonForge $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  CtxtDec[id] $\leftarrow \perp$
  Output(id)

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for $(\texttt{id}, c) \in$ Read with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow$ CtxtDec[id]
    if $(\langle A_i \to \vec{V} \rangle, m) \neq \perp \wedge B_j \in \vec{V}$ :
      outputList $\leftarrow$ outputList $\cup \{(\texttt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  Output(GetDelivered$(B_j$, outputList))

$(J)$-SenderKeyPair$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  if $c \notin$ CtxtHon $\cup$ CtxtHonForge :
    CtxtDis $\leftarrow$ CtxtDis $\cup \{c\}$
  id $\leftarrow$Write$(c)$
  if $c \in$ Dec :
    CtxtDec[id] $\leftarrow$ Dec$[c]$
  else
    CtxtDec[id] $\leftarrow$ Decryption$(B_{\mathsf{pp}}, c)$
    Dec$[c] \leftarrow$ CtxtDec[id]
  Output(id)

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

Algorithm 48 Reduction $\mathbf{C}^{\mathsf{OTR}}$ for Theorem 12.

```
$(A_i \in \mathcal{S}^H)$-$\textsc{Write}(\langle A_i \to \vec{V} \rangle, m)$
  **if** $\vec{V} \in (\mathcal{R}^H)^+$ **:**
    $c \leftarrow \mathcal{O}_E(\texttt{sig}, A_1, \overbrace{(B_{\texttt{pp}}, \ldots, B_{\texttt{pp}})}^{|\vec{V}|+1 \text{ times}}, 0^{|m|}, \emptyset)$
  **else**
    $c \leftarrow \mathcal{O}_E(\texttt{sig}, A_i, \vec{V}' \coloneqq \big(B_{\texttt{pp}}, V_1, \ldots, V_{|\vec{V}|}\big), m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H})$
  **if** $c \in \mathrm{CtxtHon} \cup \mathrm{CtxtDis}$ **:**
    Abort
  $\mathrm{CtxtHon} \leftarrow \mathrm{CtxtHon} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\texttt{id}] \leftarrow (\langle A_i \to \vec{V} \rangle, m)$
  $\mathrm{Dec}[c] \leftarrow \mathrm{CtxtDec}[\texttt{id}]$
  $\textsc{Output}(\texttt{id})$

$(P \in \mathcal{F})$-$\textsc{Write}(\langle [\mathrm{Forge}] A_i \to \vec{V} \rangle, m)$
  $c \leftarrow \textsc{ForgeRed}\big(A_i, \vec{V}, m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  $\mathrm{CtxtHonForge} \leftarrow \mathrm{CtxtHonForge} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  $\mathrm{CtxtDec}[\texttt{id}] \leftarrow \bot$
  $\textsc{Output}(\texttt{id})$

$(B_j \in \mathcal{R}^H)$-$\textsc{Read}$
  $\mathrm{outputList} \leftarrow \emptyset$
  $\mathrm{ciphertextSet} \leftarrow \emptyset$
  **for** $(\texttt{id}, c) \in \textsc{Read}$ **with** $c \notin \mathrm{ciphertextSet}$ **:**
    $\mathrm{ciphertextSet} \leftarrow \mathrm{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V} \rangle, m) \leftarrow \mathrm{CtxtDec}[\texttt{id}]$
    **if** $(\langle A_i \to \vec{V} \rangle, m) \neq \bot \land B_j \in \vec{V}$ **:**
      $\mathrm{outputList} \leftarrow \mathrm{outputList} \cup \{(\texttt{id}, (\langle A_i \to \vec{V} \rangle, m))\}$
  $\textsc{Output}(\textsc{GetDelivered}(B_j, \mathrm{outputList}))$

$(J)$-$\textsc{SenderKeyPair}(A_i \in \mathcal{S})$
  $\textsc{Output}(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-$\textsc{Write}(c)$
  **if** $c \notin \mathrm{CtxtHon} \cup \mathrm{CtxtHonForge}$ **:**
    $\mathrm{CtxtDis} \leftarrow \mathrm{CtxtDis} \cup \{c\}$
  $\texttt{id} \leftarrow \textsc{Write}(c)$
  **if** $c \in \mathrm{Dec}$ **:**
    $\mathrm{CtxtDec}[\texttt{id}] \leftarrow \mathrm{Dec}[c]$
  **else**
    $\mathrm{CtxtDec}[\texttt{id}] \leftarrow \textsc{Decryption}(B_{\texttt{pp}}, c)$
    $\mathrm{Dec}[c] \leftarrow \mathrm{CtxtDec}[\texttt{id}]$
  $\textsc{Output}(\texttt{id})$

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

// The following procedure is not part of the reduction's interface.
$\textsc{ForgeRed}(A_i, \vec{V}, m, \mathcal{C})$
  **if** $\vec{V} \in (\mathcal{R}^H)^+$ **:**
    **return** $\mathcal{O}_E(\texttt{sim}, A_1, \overbrace{(B_{\texttt{pp}}, \ldots, B_{\texttt{pp}})}^{|\vec{V}|+1 \text{ times}}, 0^{|m|}, \emptyset)$
  **return** $\mathcal{O}_E(\texttt{sim}, A_i, \vec{V}' \coloneqq \big(B_{\texttt{pp}}, V_1, \ldots, V_{|\vec{V}|}\big), m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H})$
```

Algorithm 49 Reduction $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$ for Lemma 2.

```
Initialization
  $\mathrm{CtxtChall} \leftarrow \emptyset$ // Additional Initialization.
  $\mathrm{CtxtNonChall} \leftarrow \emptyset$

$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $b \leftarrow \$\{0,1\}$ // Sample bit $b$ uniformly at random.
  if $b = 0$ :
    $c \leftarrow \mathcal{O}_E\big(\mathtt{sig}, A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
    $\mathrm{CtxtChall} \leftarrow \mathrm{CtxtChall} \cup \{c\}$
  else
    $c \leftarrow \Pi.E_{\mathsf{pp}}(\mathtt{ssk}_i, \vec{v}' := (\mathtt{rpk}_{\mathsf{pp}}, v_1, \ldots, v_{|\vec{v}|}), m)$
    $\mathrm{CtxtNonChall} \leftarrow \mathrm{CtxtNonChall} \cup \{c\}$
  Define event ${\xi_1}'$ as: ${\xi_1}' := \mathrm{CtxtChall} \cap \mathrm{CtxtNonChall} \neq \emptyset$
  if ${\xi_1}'$ :
    Guess(0) // Makes reduction output 0 as its guess.
  Output(Write$(c)$)

$(P \in \mathcal{F})$-Write$(\langle[\mathrm{Forge}]A_i \to \vec{V}\rangle, m)$
  Output(Write(Forge$\big(A_i, \vec{V}, m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$))

$(B_j \in \mathcal{R}^H)$-Read
  $\mathrm{outputList} \leftarrow \emptyset$
  $\mathrm{ciphertextSet} \leftarrow \emptyset$
  for $(\mathtt{id}, c) \in$ Read with $c \notin \mathrm{ciphertextSet}$ :
    $\mathrm{ciphertextSet} \leftarrow \mathrm{ciphertextSet} \cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow$ Decryption$(B_j, c)$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \bot$ :
      $\mathrm{outputList} \leftarrow \mathrm{outputList} \cup \{(\mathtt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
  Output(GetDelivered$(B_j, \mathrm{outputList})$)

$(J)$-SenderKeyPair$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  Output(Write$(c)$)

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

Termination
  $b \leftarrow \$\{0,1\}$
  Guess$(b)$ // Makes reduction output $b$ as its guess.
```

Algorithm 50 Reduction $\mathbf{C}^{\mathsf{Corr}\text{-}\xi_1}$ for Lemma 2. Below, we only specify the reduction for operations for which it differs from $\mathbf{C}^{\mathsf{OTR}\text{-}\xi_1}$.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $b \leftarrow \$\{0,1\}$ // Sample bit $b$ uniformly at random.
  if $b = 0$ :
    $\vec{v}' := (\mathtt{rpk}_{\mathsf{pp}}, v_1, \ldots, v_{|\vec{V}|})$
    $\vec{s} := (\bot, \mathtt{rsk}_1, \ldots, \mathtt{rsk}_{|\vec{V}|})$  // For each $V_l$: if $V_l \in \overline{\mathcal{P}^H}$, $s_{l+1}$ is $V_l$'s secret key; else is $\bot$.
    $c \leftarrow \Pi.\mathit{Forge}_{\mathsf{pp}}(\mathtt{spk}_i, \vec{v}', m, \vec{s})$
  else
    $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathsf{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  $\mathcal{O}_D(B_{\mathsf{pp}}, c)$ // Allows winning the correctness and forgery invalidity games.
  Output(Write$(c)$)

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  Output(Write$(c)$)
```

Algorithm 51 Reduction $\mathbf{C}^{\textsf{Forge-Invalid-}\xi_1}$ for Lemma 2. As for Algorithm 50, we only specify the reduction for operations for which it differs from $\mathbf{C}^{\textsf{OTR-}\xi_1}$.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $b \leftarrow \$\{0,1\}$ // Sample bit $b$ uniformly at random.
  if $b = 0$ :
    $c \leftarrow \mathcal{O}_{Forge}\big(A_i, \vec{V}' := (B_{\text{pp}}, V_1, \dots, V_{|\vec{V}|}), m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  else
    $c \leftarrow \Pi.E_{\text{pp}}(\texttt{ssk}_i, \vec{v}' := (\texttt{rpk}_{\text{pp}}, v_1, \dots, v_{|\vec{v}|}), m)$
  $\mathcal{O}_D(B_{\text{pp}}, c)$ // Allows winning the correctness and forgery invalidity games.
  Output(Write$(c)$)

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  Output(Write$(c)$)
```

Algorithm 52 Reduction $\mathbf{C}^{\textsf{OTR-}\xi_2}$ for Lemma 3.

```
$(A_i \in \mathcal{S}^H)$-Write$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(\texttt{sig}, A_i, \vec{V}' := (B_{\text{pp}}, V_1, \dots, V_{|\vec{V}|}), m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  if $c \in$ Dec : // Event $\xi_2$
    Define event $\xi_{2,0} := (\text{Dec}[c] \neq \bot)$
    Define event $\xi_{2,1} := (\text{Dec}[c] = \bot)$
    if $\xi_{2,0}$ :
      Guess(0) // Makes reduction output guess 0.
    else // Event $\xi_{2,1}$ occurred
      Guess(1) // Reduction outputs guess 1.
  Output(Write$(c)$)

$(P \in \mathcal{F})$-Write$(\langle[\text{Forge}]A_i \to \vec{V}\rangle, m)$
  Output(Write(Forge$\big(A_i, \vec{V}, m, \text{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$))

$(B_j \in \mathcal{R}^H)$-Read
  outputList $\leftarrow \emptyset$
  ciphertextSet $\leftarrow \emptyset$
  for $(\texttt{id}, c) \in$ Read with $c \notin$ ciphertextSet :
    ciphertextSet $\leftarrow$ ciphertextSet $\cup \{c\}$
    $(\langle A_i \to \vec{V}\rangle, m) \leftarrow$ Decryption$(B_j, c)$
    if $(\langle A_i \to \vec{V}\rangle, m) \neq \bot$ :
      outputList $\leftarrow$ outputList $\cup \{(\texttt{id}, (\langle A_i \to \vec{V}\rangle, m))\}$
  Output(GetDelivered$(B_j$, outputList))

$(J)$-SenderKeyPair$(A_i \in \mathcal{S})$
  Output$(\mathcal{O}_{SK}(A_i))$

$(P \in \overline{\mathcal{P}^H})$-Write$(c)$
  $\alpha \leftarrow \mathcal{O}_D(B_{\text{pp}}, c)$
  if $\alpha \neq \texttt{test}$ : // $c$ not written before by Write operation at some interface $A_i \in \mathcal{S}^H$.
    Dec$[c] \leftarrow \alpha$
  Output(Write$(c)$)

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

Termination
  $b \leftarrow \$\{0,1\}$
  Guess$(b)$ // Makes reduction output $b$ as its guess.
```

Algorithm 53 Reduction $\mathbf{C}^{\mathsf{Cons\text{-}0\text{-}}\xi_2}$ for Lemma 3. We only specify the reduction for operations for which it differs from $\mathbf{C}^{\mathsf{OTR\text{-}}\xi_2}$.

```
INITIALIZATION
  CtxtChall ← ∅ // Additional Initialization.

$(A_i \in \mathcal{S}^H)$-WRITE$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathrm{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  $\mathrm{CtxtChall} \leftarrow \mathrm{CtxtChall} \cup \{c\}$
  if $c \in \mathrm{Dec}$ :
    $\mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)

$(P \in \overline{\mathcal{P}^H})$-WRITE$(c)$
  if $c \notin \mathrm{CtxtChall}$ :
    $\mathrm{Dec}[c] \leftarrow \mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)
```

Algorithm 54 Reduction $\mathbf{C}^{\mathsf{Cons\text{-}1\text{-}}\xi_2}$ for Lemma 3. As for Algorithm 53, we only specify the reduction for operations for which it differs from $\mathbf{C}^{\mathsf{OTR\text{-}}\xi_2}$.

```
INITIALIZATION
  CtxtChall ← ∅ // Additional Initialization.

$(A_i \in \mathcal{S}^H)$-WRITE$(\langle A_i \to \vec{V}\rangle, m)$
  $\vec{v}' := (\mathtt{rpk}_{\mathrm{pp}}, v_1, \ldots, v_{|\vec{V}|})$
  $\vec{s} := (\perp, \mathtt{rsk}_1, \ldots, \mathtt{rsk}_{|\vec{V}|})$ // For each $V_l$: if $V_l \in \overline{\mathcal{P}^H}$, $s_{l+1}$ is $V_l$'s secret key; else is $\perp$.
  $c \leftarrow \mathit{\Pi}.\mathit{Forge}_{\mathrm{pp}}(\mathtt{spk}_i, \vec{v}', m, \vec{s})$
  $\mathrm{CtxtChall} \leftarrow \mathrm{CtxtChall} \cup \{c\}$
  if $c \in \mathrm{Dec}$ :
    $\mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)

$(P \in \overline{\mathcal{P}^H})$-WRITE$(c)$
  if $c \notin \mathrm{CtxtChall}$ :
    $\mathrm{Dec}[c] \leftarrow \mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)
```

Algorithm 55 Reduction $\mathbf{C}^{\mathsf{Corr\text{-}}\xi_2}$ for Lemma 3. We only present this reduction for completeness (note that it is the same reduction as $\mathbf{C}^{\mathsf{Cons\text{-}0\text{-}}\xi_2}$).

```
INITIALIZATION
  CtxtChall ← ∅ // Additional Initialization.

$(A_i \in \mathcal{S}^H)$-WRITE$(\langle A_i \to \vec{V}\rangle, m)$
  $c \leftarrow \mathcal{O}_E\big(A_i, \vec{V}' := (B_{\mathrm{pp}}, V_1, \ldots, V_{|\vec{V}|}), m\big)$
  $\mathrm{CtxtChall} \leftarrow \mathrm{CtxtChall} \cup \{c\}$
  if $c \in \mathrm{Dec}$ :
    $\mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)

$(P \in \overline{\mathcal{P}^H})$-WRITE$(c)$
  if $c \notin \mathrm{CtxtChall}$ :
    $\mathrm{Dec}[c] \leftarrow \mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE$(c)$)
```

Algorithm 56 Reduction $\mathbf{C}^{\mathsf{Forge\text{-}Invalid\text{-}}\xi_2}$ for Lemma 3. As before, we only specify the reduction for operations for which it differs from $\mathbf{C}^{\mathsf{OTR\text{-}}\xi_2}$.

```
INITIALIZATION
  CtxtChall ← $\emptyset$ // Additional Initialization.

$(A_i \in \mathcal{S}^H)$-WRITE($(\langle A_i \rightarrow \vec{V} \rangle, m \in \mathcal{M})$)
  $c \leftarrow \mathcal{O}_{Forge}\big(A_i, \vec{V}' \coloneqq (B_{\mathrm{pp}}, V_1, \ldots, V_{|\vec{V}|}), m, \mathrm{Set}(\vec{V}) \cap \overline{\mathcal{P}^H}\big)$
  CtxtChall ← CtxtChall $\cup \{c\}$
  if $c \in$ Dec :
    $\mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE($c$))

$(P \in \overline{\mathcal{P}^H})$-WRITE($c$)
  if $c \notin$ CtxtChall :
    Dec[$c$] ← $\mathcal{O}_D(B_{\mathrm{pp}}, c)$
  OUTPUT(WRITE($c$))
```