# Embedding the UC Model into the IITM Model

Daniel Rausch<sup>1</sup>, Ralf Küsters<sup>1</sup>, and Céline Chevalier<sup>2</sup>

<sup>1</sup> University of Stuttgart {daniel.rausch, ralf.kuesters}@sec.uni-stuttgart.de <sup>2</sup> CRED, Paris-Panthéon-Assas University celine.chevalier@ens.fr

Abstract. Universal Composability is a widely used concept for the design and analysis of protocols. Since Canetti's original UC model and the model by Pfitzmann and Waidner several different models for universal composability have been proposed, including, for example, the IITM model, GNUC, CC, but also extensions and restrictions of the UC model, such as JUC, GUC, and SUC. These were motivated by the lack of expressivity of existing models, ease of use, or flaws in previous models. Cryptographers choose between these models based on their needs at hand (e.g., support for joint state and global state) or simply their familiarity with a specific model. While all models follow the same basic idea, there are huge conceptually differences, which raises fundamental and practical questions: (How) do the concepts and results proven in one model relate to those in another model? Do the different models and the security notions formulated therein capture the same classes of attacks? Most importantly, can cryptographers re-use results proven in one model in another model, and if so, how?

In this paper, we initiate a line of research with the aim to address this lack of understanding, consolidate the space of models, and enable cryptographers to re-use results proven in other models. As a start, here we focus on Canetti's prominent UC model and the IITM model proposed by Küsters et al. The latter is an interesting candidate for comparison with the UC model since it has been used to analyze a wide variety of protocols, supports a very general protocol class and provides, among others, seamless treatment of protocols with shared state, including joint and global state. Our main technical contribution is an embedding of the UC model into the IITM model showing that all UC protocols, security and composition results carry over to the IITM model. Hence, protocol designers can profit from the features of the IITM model while being able to use all their results proven in the UC model. We also show that, in general, one cannot embed the full IITM model into the UC model.

## 1 Introduction

Universal composability is a widely used approach for the modular design and analysis of cryptographic protocols. Protocols are shown to be secure in arbitrary (polynomial-time) contexts, which allows for composing protocols and re-using security results. Security properties are stated in terms of ideal functionalities/protocols. To prove a real protocol  $\pi$  secure w.r.t. an ideal functionality  $\phi$  one shows that for all network adversaries  $\mathcal{A}$  attacking  $\pi$  there is an ideal adversary, called simulator, that interacts with  $\phi$  such that no (polynomial-time) environment  $\mathcal{E}$  can distinguish between the real and the ideal world. We write  $\pi \leq \phi$  in this case. Composition theorems then immediately imply that one can replace subroutines  $\phi$  used by an arbitrary higher-level protocol  $\rho$  with their realization  $\pi$  such that  $\rho^{\phi \to \pi} \leq \rho$ , where in  $\rho^{\phi \to \pi}$  the protocol  $\rho$  uses (possibly multiple instances of)  $\pi$  instead of  $\phi$ .

The UC model by Canetti [6,7] and the reactive simulatability model by Pfitzmann and Waidner [22] pioneered this line of research. Since then many different models implementing the same idea of universal composability have been proposed, generally motivated by issues in other existing models such as a lack of expressiveness, overly complicated computational models, and also formal flaws in theorems. To name just a few examples: The JUC [10] and GUC [9] models were proposed as extended variants of the UC model which allow for modeling larger classes of protocols, namely those with joint state (where some state, such as a signature key, is used by multiple protocol sessions) or global state (which is shared with arbitrary other protocols), respectively. The SUC model [8] is a simpler variant of the UC model tailored towards secure multi party computation. The GNUC model [12] was designed as a sound alternative to the UC model, fixing several issues that formally invalidated the UC composition theorem at the time. The IITM model [14, 20] offers a simple computational model that supports a very general class of protocols and composition theorems, which, out of the box, support joint state, global state, and arbitrarily shared state, also in combination. The CC model [21] follows a more abstract approach that does not fix a specific computational model, runtime notion, instantiation mechanism, or class of environments.

In the literature, cryptographers often choose the security model based on their needs at hand (for instance support for joint or global state), syntax preferences, or simply their familiarity with a specific model. While all of the above models follow the same basic idea of universal composability, the details are (sometimes drastically) different. It is hence generally unclear how different models and the security results obtained therein relate to each other: Is one model strictly more powerful than another? Can all protocols formalized in one model also be formalized in the other? Are security notions compatible? Do security results carry over from one model to the other? This lack of a deeper understanding of the relationship of models is quite disturbing. For example, we might miss some practical attacks in our security proofs because we, due to a lack of knowledge, chose a model that might actually offer only a weaker security notion than other models. Perhaps worst of all, security results proven in one model currently cannot be used in another model. This drastically limits reusability of security results, contradicting one of the key features of universally composable security and more generally modular analyses.

**Our Goal.** In this paper, we initiate a line of research with the aim to address this lack of understanding and clarify the relationships between models for universal composability. One of our main goals is to identify, as far as possible, classes of protocols and security results that can be transferred from one model to another. This would enable protocol designers to use a model of their choice, based on their personal preference, the specific needs at hand, as well as the features offered by the model, while still being able to benefit from results shown in another model. This would also provide insights into the concepts employed in one model compared to other models and the strength of the security results obtained within a specific model, potentially justifying that such results are reasonable and cover all practical attacks. Besides consolidating and re-using results, this research can also help consolidating and unifying the space of models themselves. A complete classification of *all* universally composable models is of course out of reach of a single paper. As a first step towards our objective, we here focus on embedding the UC model into the IITM model. We also prove that, in general, the IITM model cannot be embedded into the UC model. To the best of our knowledge our work is the first to study such embeddings, and hence, relate complete models for universal composability. So far, only specific aspects have been considered. For example, in [15] the relationship between security notions employed in various models has been studied, although the study was carried out in one model, and [13,14] discuss runtime notions employed in different models.

On the UC and IITM models. We choose the UC model [6] since it is currently the most widely used model in the literature on universal composability. The IITM model [14] has also already been used intensively to analyze a wide variety of protocols, including cryptographic protocols (e.g., [16, 17]) and also more generally security protocols such as blockchains and distributed ledgers (e.g., [11]). The IITM model is an interesting candidate for a comparison since it supports a very general protocol class and comes with composition theorems which cover joint state and global state out of the box as well as protocols with arbitrary shared state (joint and global state are special cases of shared state) and protocols without pre-established session IDs [17], i.e., parties in one session are not required to share the same SID or fix it upfront (see [4, 17, 19, 20] for overviews of these features). Moreover, all these features can be freely combined since they are all covered within one framework. While recently it has been shown that the UC model directly supports global state [2], combinations of, for example, joint state and global state or features like general shared state and protocols without pre-established SIDs are not yet supported in UC. Hence, an embedding as carried out in this paper enables protocol designers to profit from such features of the IITM model while still being able to access the wide range of existing results shown in the UC model.

For both the UC and IITM model there are recent journal publications; the UC model has been published in the Journal of the ACM [7] and the IITM model in the Journal of Cryptology [20]. These provide a solid basis for a comparison. Such a comparison is far from trivial since the computational frameworks of both models are defined in very different ways, using sometimes drastically different

concepts where it is far from obvious how they relate and whether there is a meaningful relationship at all.

**Our contributions.** Conceptual Differences. After recalling the most important definitions and theorems of the UC and IITM models in Section 2, we first highlight in Section 3.1 the major conceptual differences between the two models: Diff. 1 concerns support for dynamically generated machine code, Diff. 2 to 8 are about message routing and sender/receiver authentication, Diff. 9 concerns the different polynomial runtime notions employed in the two models, Diff. 10 concerns the classes of environments considered, and Diff. 11 to 14 are about requirements of the UC security notion and composition theorem that are not present in the IITM model.

Mapping of Protocols. With that analysis in mind, one main contribution, given in Sections 3.2 and 3.5, consists in mapping UC protocols to IITM protocols. This requires bridging the mentioned differences and to show that the mapping is faithul, i.e., the original and the mapped protocols have the same behavior (functional, security, complexity) in all contexts they run in (Lemmas 1 and 2). This then implies that all UC protocols can be expressed as IITM protocols.

Mapping and Preservation of Security Results. We show in Section 3.3 that this mapping also preserves security results. That is,  $\pi_{UC} \leq_{UC} \phi_{UC}$  in the UC model iff  $\pi_{IITM} \leq_{IITM} \phi_{IITM}$  for the mapped protocols in the IITM model (Theorem 4). For the direction from IITM to UC, we require that  $\pi_{IITM} \leq_{IITM} \phi_{IITM}$  can be shown for simulators meeting the UC runtime notion (Theorem 5). Assuming the existence of time-lock puzzles, we also show that this direction does not hold in general since the class of IITM simulators is strictly larger than the class of UC simulators (Lemma 3). This latter result is independent of a specific protocol mapping, and hence, is a fundamental difference between the models, which we further discuss in Section 3.3.

Mapping and Preservation of Composition Results. Section 3.4 discusses composition. One easily observes that Theorem 4 already implies that security results for composed protocols carry over from UC to IITM by first applying the UC composition theorem and then mapping the resulting UC protocols to the IITM model (Corollary 1). But this result does not relate the composition theorems employed in the models themselves. We therefore show that Corollary 1 can be obtained directly in the IITM model using the IITM composition theorem and without relying on the UC theorem (Corollary 2).

This result also enables composition of mapped UC protocols with arbitrary other IITM protocols within the IITM model, including those that do not have a UC counter part and which use features of the IITM model that are out of the scope of the UC model. We discuss these options in Section 3.6.

The Other Direction: Limitations. We discuss in Section 4 the other direction of translating IITM protocols and security results to the UC model. To summarize, [20] has already shown that the IITM runtime notion permits natural protocols that cannot be expressed in the UC model. Combined with our results, this shows that the class of IITM protocols is strictly larger than the class of UC protocols.

Our result from Lemma 3 further shows that also the class of IITM simulators is strictly larger than the class of UC simulators due to their runtime notions. So the best one can hope for is a mapping for the class of IITM protocols and simulators that follow the UC runtime notion. We also discuss further obstacles of an embedding of the UC model into the IITM model. We leave it to future work to study this in more details and provide an embedding of (a subset of) the IITM model into the UC model.

Further Insights and Results Obtained Through the Embedding. Firstly, we develop a modeling technique that allows for obtaining a new type of composition as a corollary from existing UC and IITM composition theorems as well as similar models (cf. Section 3.4). Secondly, we found several previously unknown technical issues in the UC model that, among others, formally invalidate the UC composition theorem (cf. Sections 2.1, 3.3, 3.4). We propose fixes for all of these issues which should be compatible with and hence retroactively apply to existing UC protocols from the literature.

Altogether, our paper provides deep insights into the UC and IITM models, clarifies the purpose of different concepts employed by the models for achieving similar goals, relates them, also in terms of expressiveness, and uncovers how security results compare to each other. Our main result shows that all protocols, security, and composability results from the UC model carry over to the IITM model. As an immediate practical benefit, this opens up entirely new options for protocol designers so far working in the UC model: they can use all their results also in the IITM model, combine their work with protocols in the IITM model and benefit from IITM features including seamless support for joint, global, shared state, and protocols without pre-established session IDs, as well as arbitrary combinations thereof.

## 2 A Brief Overview of the UC and IITM Models

In this section, we provide brief overviews of the UC and IITM models. We refer the reader to [7, 20] for more in-depth information about both models. The presentation here should suffice to follow the rest of the paper.

#### 2.1 The UC Model

The general computational model of the UC model is defined in terms of systems of interactive Turing machines (ITMs or just machines, for short). An interactive Turing machine M in the UC model is a probabilistic Turing machine with three special communication tapes, called *input*, *subroutine-output* (or simply *output*), and *backdoor tape*. In a run of a system of machines (see also below), *machine instances* are created. Every instance has some machine code that it runs when activated and some identifier. More specifically, each instance has a unique so-called *extended ID eid* = (c, id), consisting of its machine code c and some identify string *id* that, except for the environment (which has id = 0), is of the form id = (pid, sid) for a process/party identifier *pid* and a session identifier

sid. Machine instances have access to two special operations: a read next message instruction which moves the head of one of the three mentioned communication tapes to the start of the next received message within a single unit of time and an external-write instruction which allows a machine instance to append a message m to one of the (three) communication tapes of another machine instance, and hence, send m to that other instance. On an input tape machine instances receive messages from higher-level protocols or the environment, on subroutine-output tapes they receive messages from subroutines, and on backdoor tapes they receive messages from the network/the adversary.

A system of machines (M, C) consists of the machine code M of the first ITM to be activated and a so-called control function C which can prohibit or alter external-write operations; this is later used to define the security experiment. The first instance to be activated with external input a in a run of this system is a machine instance running code M with ID 0. During a run of such a system, at any time only one machine instance is active and all other machine instances wait for new input via the external write operation. When a machine sends a message m via an external write operation to one of the three communication tapes of another machine, say tape t, there are two main options to specify the recipient: Firstly, by giving an extended ID *eid*. If there does not exist a machine instance with this extended identity yet, then such an instance running the code cspecified in its eid is first created. Then, m is written to the tape t of the machine instance with extended ID *eid* and that machine becomes active (the sender becomes inactive). This first case is also called *forced-write*. Secondly, by giving a predicate P on extended IDs. In this case, m is written to the tape t of the first existing machine instance (sorted by the order of their first creation) with eid such that P(eid) holds true. We will refer to this second case as non-forced-write. For both types of external write operations, the sender can either hide or reveal its own extended identity towards the recipient. If an external write operation does not succeed, e.g., when there is no existing machine instance matching the predicate P, then the initial ITM instance (M, 0) is activated again. A run ends when the initial ITM reaches a final halting state. The overall output of such a run is the first bit written on a specific tape of the initial ITM instance.

Two systems of machines (M, C) and (M', C') are called indistinguishable (and we write  $(M, C) \equiv (M', C')$ ) if the difference between the probability that (M, C) outputs 1 and the probability that (M', C') outputs 1 is negligible in the security parameter  $\eta$  and the external input  $a^{3}$ 

**Runtime.** Machine instances can receive and send so-called *import* as part of their messages m to/from other machine instances, where import is encoded as a *binary* number contained in a special field of m. A machine M is called *probabilistic polynomial-time* (ppt) iff (i) there is a polynomial p such that the overall runtime of (an instance of) M during all points of a run is upper bounded by  $p(n_I - n_O)$ , where  $n_I$  is the sum of all imports received by (that instance of) M to other machines,

<sup>&</sup>lt;sup>3</sup> A function  $f: \mathbb{N} \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$  is called *negligible* if for all  $c, d \in \mathbb{N}$  there exists  $\eta_0 \in \mathbb{N}$  such that for all  $\eta > \eta_0$  and all  $a \in \bigcup_{n' < n^d} \{0,1\}^{\eta'}: f(\eta, a) < \eta^{-c}$ .

and (ii) whenever M uses a forced write operation to a machine instance with code M', then M' is also ppt for the same polynomial p.<sup>4</sup> Furthermore, all machines are parameterized with a security parameter  $\eta$ . All machine instances are required to run only when they hold at least  $\eta$  import, i.e.,  $n_I - n_O \geq \eta$ .

Simulation-Based Security. Security of a protocol  $\pi$  is defined via a security experiment involving an adversary  $\mathcal{A}$  and an environment  $\mathcal{E}$ , where each of these components is modeled via an ITM with code  $\pi$ ,  $\mathcal{A}$ , and  $\mathcal{E}$  respectively. More specifically, the experiment is defined via the system  $(\mathcal{E}, C_{EXEC}^{\pi,\mathcal{A}})$  where  $C_{EXEC}^{\pi,\mathcal{A}}$  is a control function that enforces the following rules of communication:

- The environment  $\mathcal{E}$  (with ID 0) can write only to input tapes, only via forced write, and only to IDs of the form (pid, sid) where *sid* must be the same as in previous write operations (if any exist). This uniquely defined *sid* is also called *challenge session ID*  $sid_c$ . If *pid* is the special symbol  $\diamond$ , then the control function changes the code of the recipient to  $\mathcal{A}$ ; otherwise, the code is changed to  $\pi$ . So  $\mathcal{E}$  can talk to  $\mathcal{A}$  or to  $\pi$  (in session  $sid_c$ ). Unlike all other machines, the environment is given the additional freedom to freely choose the extended identity that is claimed as a sender of a message.
- The adversary  $\mathcal{A}$  (with ID ( $\diamond$ ,  $sid_c$ )) may write only to backdoor tapes of other machines and may not use the forced-write mechanism (i.e., he can write only to already existing instances using non-forced-writes).<sup>5</sup>
- All other machine instances (which are part of the protocol stack of  $\pi$ , including subroutines) must always reveal their own sender extended identities. They may write to the backdoor tape of (the unique instance of)  $\mathcal{A}$  using non-forced-write without specifying the code of the adversary and without providing import. They may write to input and output tapes of instances other than (the unique instances of)  $\mathcal{E}$  and  $\mathcal{A}$ , subject to the following modification: If the sending instance (M, (pid, sid)) has code  $M = \pi$ ,  $sid = sid_c$ , the recipient tape is the output tape, and the recipient instance does not exist yet, then the message is instead redirected to the output tape of  $\mathcal{E}$  with the code M removed from the extended sender identity. The extended identity of the originally intended receiver is also written to the output tape of  $\mathcal{E}$ .

The initial import for environments is defined to be the length of the external input a, which is at most some polynomial in the security parameter  $\eta$  (as per the

<sup>&</sup>lt;sup>4</sup> Intuitively, each import serves as a runtime token that can be passed between instances. This is one possible mechanism for ensuring that the whole system, where an unbounded number of instances can be created, runs in overall polynomial time. Hence, a polynomial time environment can internally simulate such a system.

<sup>&</sup>lt;sup>5</sup> The journal version of the UC model [7] formally does not prevent the adversary from revealing its sender extended identity, including its code, to other machines. We found that this option actually causes several severe issues, including a failure of the composition theorem (cf. Appendix J.1 for details). This appears to be an oversight rather than an intended feature. Indeed, previous versions of the UC model, such as the one from 2013 [5], included a mechanism that always removed the sender identity from backdoor tape messages. In what follows, we therefore assume that adversaries must also hide their own sender extended identity. This fixes the issue and is compatible with existing results in the literature.

definition of negligible functions with external input). Environments are required to be *balanced*, i.e., provide at least as much import to the adversary as they provide in total to all instances of the challenge protocol  $\pi$ , i.e., all instance with extended IDs of the form  $(\pi, (pid, sid_c))$ , where  $sid_c$  is the fixed challenge SID. Given a set of extended identities  $\xi$ , an environment is called  $\xi$ -*identity-bounded* if it claims only sender extended identities from  $\xi$ . The set  $\xi$  may be determined dynamically via a polytime predicate over the current configuration of the whole system at the time the input it sent to the protocol, which includes (the states of) all existing instances of the environment, adversary, and protocol machines. Given this terminology, the security notion for protocols is defined as follows:

**Definition 1.** Let  $\pi$  and  $\phi$  be ppt protocols. Then  $\pi$  realizes  $\phi$  w.r.t.  $\xi$ -identitybounded environments ( $\pi \leq_{UC}^{\xi} \phi$ ) if for all ppt adversaries  $\mathcal{A}$  there exists a ppt adversary  $\mathcal{S}$  (a simulator or an ideal adversary) such that for all ppt  $\xi$ -identitybounded environments  $\mathcal{E}$  it holds true that ( $\mathcal{E}, C_{EXEC}^{\pi, \mathcal{A}}$ )  $\equiv (\mathcal{E}, C_{EXEC}^{\phi, \mathcal{S}})$ .<sup>6</sup>

**Composition Theorem.** To state the composition theorem, a bit more terminology is needed. A session (with SID sid) of a protocol  $\pi$  consists of all instances running code  $\pi$  with SID sid. We call these instances highest-level instances, i.e., those are exactly the instances that can receive inputs and provide outputs to the environment in the security experiment. The session sid of  $\pi$  further includes all instances, i.e., subroutines, that have received an input or output from another instance that is part of the session (except for outputs by highest-level instances, which are intended for the environment/higher-level protocols using the session of  $\pi$ ).

A protocol  $\pi$  is called *subroutine respecting* if a protocol session of  $\pi$  interacts with other existing machine instances not belonging to the session only via inputs to and outputs from the highest level instances of the session, even when  $\pi$  is used as a subroutine within a higher-level protocol  $\rho$ .<sup>7</sup> The UC model provides a standard implementation of the subroutine respecting property via a subroutine respecting shell code that is added as a wrapper on top of the code of  $\pi$  and its subroutines. A protocol  $\pi$  is called *subroutine exposing* if every session *s* of the protocol provides an interface to the adversary that the adversary can use to learn whether some extended identity *eid* (specified by the adversary) is part of the session *s*. The UC model proposes a standard implementation of this mechanism by adding a so-called *directory machine*.

A (higher-level) protocol  $\rho$  is called  $(\pi, \phi, \xi)$ -compliant if (i) all instances of all sessions of  $\rho$  perform write requests to input tapes only via forced-write

<sup>&</sup>lt;sup>6</sup> The UC model also defines security w.r.t. the *dummy adversary*  $\mathcal{A}_{Dum}$ , which essentially simply forwards messages between the environment and the protocol, and shows this conceptually simpler definition, where only the dummy adversary is considered rather than quantification over all adversaries, to be equivalent. Also, if  $\xi$  always permits all identities, then one simply writes  $\leq UC$  instead of  $\leq_{UC}^{\xi}$ .

<sup>&</sup>lt;sup>7</sup> The subroutine respecting property ensures that  $\pi$  running within a larger protocol  $\rho$  still behaves as in the security experiment, where an environment can interact only with one session of  $\pi$  and only via the highest-level instances of that session.

and ignore outputs from instances that do not reveal their extended identities, (*ii*) there are never two external write requests (made by any instances of any session of  $\rho$ ) for the same SID but one for code  $\pi$  while the other is for code  $\phi$ , and (*iii*) the extended identities of all instances in all sessions of  $\rho$  that pass inputs to an instance with code  $\pi$  or  $\phi$  satisfy the polytime predicate  $\xi$ . Given such a ( $\pi, \phi, \xi$ )-compliant protocol  $\rho$ , the protocol  $\rho^{\phi \to \pi}$  is defined just as  $\rho$  but replaces (input write requests to) subroutine instances of  $\phi$  with (input write requests to) subroutine instances of  $\pi$ . In subroutines of  $\rho$  this replacement is done as well. <sup>8</sup> Now, the composition theorem is as follows:

**Theorem 1 (UC Composition** [7]). Let  $\rho, \pi, \phi$  be ppt protocols, let  $\xi$  be a ppt predicate, such that  $\rho$  is  $(\pi, \phi, \xi)$ -compliant,  $\pi$  and  $\phi$  are both subroutine respecting and subroutine exposing, and  $\pi \leq_{UC}^{\xi} \phi$ . Then  $\rho^{\phi \to \pi} \leq_{UC} \rho$ .

#### 2.2 The IITM Model

The general computational model of the IITM model is defined in terms of systems of (inexhaustible) interactive Turing machines (IITMs or just machines, for short). An interactive Turing machine in the IITM model is a probabilistic Turing machine with an arbitrary number of named bidirectional communication tapes.<sup>9</sup> The names are used for determining pairwise connections between machines in a system of machines.<sup>10</sup> Each machine specifies a CheckAddress and a Compute mode that it can run in, where the former is a ppt algorithm used for addressing individual copies/instances of the same machine and the latter is an algorithm describing the actual computations of instances of the machine (see below).

A system Q of IITMs is a set of IITMs of the form  $Q = \{M_1, \dots, M_k, !M'_1, \dots, !M'_{k'}\}^{11}$  where the  $M_i$  and  $M'_j$  are machines and each tape name is shared by at most two machines in the system. Two machines are called *connected* if they have tapes with the same name. The operator '!' indicates that in a run of a system an unbounded number of (fresh) instances of a machine may be generated (e.g., to model multiple protocol sessions); for machines without this operator there is at most one instance of this machine in every run of the system. The first instance to be activated with external input a in a run of Q is an instance of the so-called master IITM; this machine is the only one with a so-called master (input) tape on which it receives external input a given to the system (jumping slightly ahead, the master IITM will be part of the environment). In a run of a system Q, at any time only one machine instance is active and all other instances wait for

<sup>&</sup>lt;sup>8</sup> Formally,  $\rho^{\phi \to \pi}$  contains an additional so-called UC composition shell code which acts as a wrapper that replaces these write requests. The wrapper also modifies sender identities in outputs of  $\pi$  to include the code of  $\phi$  instead to keep the subroutine replacement hidden from higher level machines.

<sup>&</sup>lt;sup>9</sup> Formally, the IITM model uses unidirectional tapes. These can be paired to create bidirectional tapes as a special case, as shown in, e.g., [3,4].

<sup>&</sup>lt;sup>10</sup> Tape names are hidden from and non-accessible to the logic of the machines. Hence, they can be renamed and even reconnected without changing the logic of the machine.

<sup>&</sup>lt;sup>11</sup> Also written  $M_1 | \cdots | M_k | ! M'_1 | \cdots | ! M'_{k'}$ .

new input. If, in  $\mathcal{Q}$ , machines M and M' are connected via a tape, say a tape named n, then an (instance of) M can send a message m to and thus trigger an (instance of) M' by writing m on its tape named n. To determine which instance of M' (if any) gets to process m, the following is done: The message is copied to the tape named n of the first existing instance of M', where instances are sorted by the order of their first creation. (The case that no instance of M' exists yet, is handled below.) That instance then processes m using its CheckAddress algorithm, which either accepts or rejects the input. If the input is accepted, this instance continues processing m using the Compute algorithm. Otherwise, if the input is rejected, then its state is reset to the point before m was written to its tape and the next instance of M' is activated with message m in mode CheckAddress. If none of the existing copies accept and M' is in the scope of a '!', or no copies of M' exist yet, then a new instance of M' is created and runs in mode CheckAddress with input m on tape n. If it accepts, it gets to process musing **Compute**; otherwise, the fresh instance is deleted again and, as a fallback, an instance of the master IITM of  $\mathcal{Q}$  is activated with empty input. The same fallback is also used if an instance (except for instances of the master IITM) stops without sending a message. A run stops if an instance of the master IITM does not produce output or a machine outputs a message on a special tape named decision (just as for the master IITM, only environments have such a special tape). Such a message is considered to be the overall output of the system.

Two systems Q and  $\mathcal{R}$  are called indistinguishable ( $Q \equiv \mathcal{R}$ ) if the difference between the probability that Q outputs 1 (on the decision tape) and the probability that  $\mathcal{R}$  outputs 1 is negligible in the security parameter  $\eta$  and the external input *a* (see Footnote 3).

**Types of Systems and Their Runtime.** We need the following terminology. For a system Q, the tapes of machines in Q that do not have a matching tape, i.e., there does not exist another machine in Q with a tape of the same name, are called *external*. External tapes are grouped into I/O and *network tapes/interfaces* modeling direct connections to subroutines/higher-level protocols and network communication, respectively. We consider three different types of systems, modeling i) *real* and *ideal protocols/functionalities*, ii) *adversaries* and *simulators*, and iii) *environments*: Protocol systems (protocols) and environmental systems (environments) are systems which have an external I/O and network interface, i.e., they may have I/O and network tapes. Adversarial systems (adversaries) only have an external network interface. Environmental systems may contain a master machine and may produce output on the decision tape.

An environment must be *universally bounded*, i.e., the overall runtime of all instances in a run of an environmental system must be bounded by a single unique polynomial (in the security parameter and length of the external input) even when connected to and running with arbitrary systems. Protocols are required to be *environmentally bounded*, i.e., when combined with an environment, the overall system (which includes all instances of all machines) must run in polynomial time (in the security parameter and length of the external input), except for potentially a negligible set of runs. Note that the polynomial can depend on

the environment. Given a protocol, an adversary for that protocol has to satisfy the following condition: the system obtained by combining the adversary and the protocol needs to be environmentally bounded. (Note that, e.g., dummy adversaries belong to this class for all protocols.)

Simulation-Based Security. We can now define the security notion of strong simulatability:<sup>12</sup>

**Definition 2.** Let  $\mathcal{P}$  and  $\mathcal{F}$  be protocols with the same I/O interface, the real and the ideal protocol, respectively. Then,  $\mathcal{P}$  realizes  $\mathcal{F}$  ( $\mathcal{P} \leq_{IITM} \mathcal{F}$ ) if there exists an adversary  $\mathcal{S}$  (a simulator or an ideal adversary) such that the systems  $\mathcal{P}$ and  $\mathcal{S} | \mathcal{F}$  have the same external interface and for all environments  $\mathcal{E}$ , connecting only to the external network and I/O interface of  $\mathcal{P}$  (and hence, the external interface of  $\mathcal{S} | \mathcal{F}$ ), it holds true that  $\mathcal{E} | \mathcal{P} \equiv \mathcal{E} | \mathcal{S} | \mathcal{F}$ .

**Composition Theorems**. The main IITM composition theorem handles concurrent composition of a fixed number of (potentially different) protocols:

**Theorem 2 (IITM Composition [20]).** Let  $\mathcal{Q}, \mathcal{P}, \mathcal{F}$  be protocols such that  $\mathcal{Q}$ and  $\mathcal{P}$  as well as  $\mathcal{Q}$  and  $\mathcal{F}$  connect only via their external I/O interfaces with each other and  $\mathcal{P} \leq_{IITM} \mathcal{F}$ . Then,  $\mathcal{Q} | \mathcal{P} \leq_{IITM} \mathcal{Q} | \mathcal{F}$ .

The IITM model also provides another security notion and a composition theorem for unbounded self-composition, which intuitively states the following:

**Theorem 3 ((Informal) IITM Unbounded Self Composition).** Let  $\mathcal{P}, \mathcal{F}$  be protocols with disjoint sessions. If there exists a simulator  $\mathcal{S}$  such that no environment interacting with just a single session of  $\mathcal{P}, \mathcal{F}$  can distinguish  $\mathcal{P}$  and  $\mathcal{S} \mid \mathcal{F}$ , then  $\mathcal{P} \leq_{IITM} \mathcal{F}$ .

In other words, it is sufficient to analyze the security of a single session of such a protocol to then conclude security of an unbounded number of sessions. For interested readers we recall the formal definition of this security notion and composition theorem in Appendix I. This second theorem can be combined with the main composition theorem to obtain a statement similar to Theorem 1 of the UC model since from the assumption of Theorem 3 and if Q connects only to the external I/O interface of  $\mathcal{P}$  (and hence,  $\mathcal{F}$ ), we not only get  $\mathcal{P} \leq_{IITM} \mathcal{F}$  but immediately also  $\mathcal{Q} | \mathcal{P} \leq_{IITM} \mathcal{Q} | \mathcal{F}$ . Roughly,  $\mathcal{Q}$  corresponds to (higher-level machines of)  $\rho$  in Theorem 1, and  $\mathcal{P}$  and  $\mathcal{F}$  to the subroutines  $\pi$  and  $\phi$ , respectively.

## 3 Embedding the UC Model in the IITM Model

We now show the embedding of the UC model into the IITM model. Formally, we consider arbitrary protocols  $\pi_{UC}$ ,  $\phi_{UC}$ ,  $\rho_{UC}$  defined in the UC model such

<sup>&</sup>lt;sup>12</sup> The IITM model also supports further security notions, including simulation w.r.t. the dummy adversary  $\mathcal{A}_{Dum}$  or w.r.t. arbitrary adversaries  $\mathcal{A}$  in the real world. All of these notions have been shown to be equivalent in the IITM model [20].

that  $\pi_{UC} \leq \xi_{UC} \phi_{UC}$  and the UC composition theorem can be applied to  $\rho_{UC}$  to obtain  $\rho_{UC}^{\phi \to \pi} \leq_{UC} \rho_{UC}$ . The overall goal of this section is to show that these protocols, security, and composability results naturally carry over to the IITM model (and then can further be used in the IITM model). We discuss the embedding in the other direction in Section 4.

#### 3.1 Main Conceptual Differences

Let us first list the key conceptual and technical differences of the UC and IITM models in terms of computational models, security definitions, and theorems. We further give pointers to where these difference are bridged.

#### Support for dynamically generated machine code (cf. Section 3.5).

1. The UC model directly supports dynamically determining the machine code of new machine instances. The IITM model fixes a finite static set of machine codes that can be instantiated during the run of a system.

Message routing and sender/receiver authentication (cf. Section 3.2).

- 2. Both the UC and IITM models provide an operation for machine instances to send messages to other instances. The UC model allows an instance to send messages to any other instance (subject to a few restrictions imposed by the security experiment). In the IITM model two instances can send messages to each other iff they are instances of two different machines  $M_1$  and  $M_2$  that share a tape with the same name.
- 3. The UC model distinguishes between two types of messages between protocol machines, namely those that provide input to a subroutine and those that provide output to a higher-level protocol. The IITM model does not have such a distinction but rather uses I/O tapes for both types of messages.
- 4. The UC model uses IDs of the form (*pid*, *sid*) to address messages to different protocol instances with the same machine code. The IITM model instead uses the generic CheckAddress mechanism which can be freely instantiated by protocol designers to capture the desired way of addressing of instances.
- 5. The UC model authenticates the sender of a message (within a protocol) by revealing its extended ID, consisting of the machine code and the ID of the instance. The IITM model authenticates the sender via the tape a message is received on, but does not guarantee that the receiver learns an ID identifying a specific instance or the code of the sender.
- 6. The adversary in the UC model cannot spawn any new protocol machine instances; he may only communicate with existing instances. The adversary in the IITM model can spawn new instances.
- 7. The UC model allows for specifying the receiver of a message via a predicate over the extended IDs of all existing machine instances (non-forced-writes). The CheckAddress algorithm of the IITM model bears some similarity, but runs only over the IDs of instances that share the same machine code.
- 8. In the UC model, outputs sent from the highest level protocol machines are redirected to the environment under certain conditions, in which case they are also modified by removing the machine code of the sender. Protocols in

the IITM model send messages to the environment iff they are written to an external I/O tape, without redirections or modifications.

#### Polynomial runtime notions (cf. Section 3.2).

9. The UC and IITM models use different runtime notions, with the former being defined for individual machines that use runtime tokens while the latter is defined for entire systems and does not mandate a specific mechanism for enforcing runtime.

#### Support for specific classes of environments (cf. Section 3.3).

10. The UC security notion supports identity bounded environments that use only sender identities as specified by a polytime predicate  $\xi$ . Environments in the IITM model are not required to adhere to any type of predicate.

# Additional requirements of the UC security notion and composition theorem (cf. Theorem 4 and Corollary 2).

Theorem 4 ensures that if  $\pi_{UC} \leq _{UC} \phi_{UC}$  in the UC model, then  $\pi_{IITM} \leq_{IITM} \phi_{IITM}$  for the mapped protocols in the IITM model, bridging Diff. 11 and 12. Security results for composed protocols also carry over (Section 3.4): Corollary 2 explains how the additional requirements of the UC model (Diff. 13 and Diff. 14) are reflected in the mapped IITM protocols while not being necessary for general IITM protocols.

- 11. The UC model requires environments to be balanced, providing a minimal amount of import to the adversary. Environments in the IITM model are not restricted in a similar way since adversaries in the IITM model do not require import to be able to run.
- 12. The UC security notion analyzes the security of a single session of a protocol. The IITM model offers two security notions: A single session security notion and a more general multi session security notion.
- 13. Protocols in the UC model have to be subroutine respecting and subroutine exposing to support composition. The main composition theorem of the IITM model does not have analogous requirements since the underlying security notion considers a more general multi-session setting, where sessions can share state with each other and subroutines can communicate with the environment.
- 14. Higher-level protocols in the UC model have to be compliant to support composition. Composition in the IITM model instead requires only that higher-level protocols may not connect to the network interface (the UC model enforces the latter at the level of its security notion).

These sometimes drastic technical differences create several challenges that we have to resolve while embedding the UC into the IITM model. For simplicity of presentation, in what follows we at first focus on the case where protocols use only machine codes from an arbitrary static but fixed set of different machine codes, rather than using (ad hoc) dynamically generated code (see Diff. 1); we denote this fixed set by **Codes**. Note that this is a very natural class of protocols which includes virtually all protocols proposed in the UC literature: generally, the machine codes of honest parties in a protocol are defined and fixed upfront,

potentially as a parameter, before the protocol is analyzed. While corrupted parties are typically allowed to choose (almost) arbitrary receiver machine codes for their messages, spawning a new machine with machine code that is not used/recognized by an honest party does not give any additional power to the adversary; the adversary can just internally simulate that machine to obtain the same results. Hence, w.l.o.g. one can assume that corrupted parties in those protocols also communicate only with machines running some code from the set **Codes**, thereby meeting the above property. Nevertheless, to make our mapping formally complete, we show in Section 3.5 how our embedding and all security and composability results can easily be extended to handle also protocols with dynamically generated machine code, i.e., how to bridge Diff. 1.

## 3.2 Mapping Protocols

Let  $\pi_{UC}$  be a protocol that uses a finite set of machine codes Codes with n := |Codes|. Note that  $\pi_{UC}$  itself is also one of those codes, in what follows denoted by  $c_{\pi} \in \text{Codes}$  to make the distinction between the protocol code  $c_{\pi}$  and the overall protocol  $\pi_{UC}$  clear.

**Normalization.** W.l.o.g., let us first bring the protocol  $\pi_{UC}$  and the codes Codes into a normalized form. These purely syntactical changes remove some technical edge cases that would otherwise needlessly complicate the mapping. Recall that, since  $\pi_{UC}$  is subroutine respecting, instances of that protocol can be grouped into disjoint protocol sessions. In each of those sessions, the only instances that can communicate with a higher-level protocol/the environment are instances of the highest-level machine with code  $c_{\pi}$  and with a certain SID  $sid_c$  that is specific to that protocol session. We refer to such a protocol session by  $sid_c$ . We assume that  $\pi_{UC}$  is such that within each protocol session  $sid_c$ there are no instances running code  $c_{\pi}$  with an SID different from  $sid_c$ . Most protocols from the literature naturally meet this property. Other protocols can trivially be modified by, e.g., adding a dummy forwarder on top of  $\pi_{UC}$  that forwards messages between the environment and those instances running code  $c_{\pi}$  with SID sid<sub>c</sub>. This dummy then meets our assumption since the dummy code is now the highest level code and one can easily ensure that it is never called (as a subroutine) with an SID different from  $sid_c$ . Note that introducing a dummy does not affect any of the properties of and security results for  $\pi_{UC}$ so is indeed without loss of generality. We also assume that the protocol  $\pi_{UC}$ uses the standard mechanism proposed by the UC model for implementing the "subroutine-respecting" requirement, i.e., all codes in Codes already include the standard subroutine respecting shell code that acts as a wrapper. Among others, this wrapper guarantees that subroutine instances are aware of the SID  $sid_c$ of their protocol session since all subroutine instances have SIDs of the form  $(sid_c, sid')$ . Again, this is already the case for virtually all protocols from the literature. If a protocol does not use this mechanism, it can be added on top of the protocol since this also does not affect any of the security properties of and results proven for  $\pi_{UC}$  given that  $\pi_{UC}$  is already assumed to be subroutine respecting.

**IITMs and tapes.** We model the protocol  $\pi_{IITM}$  in the IITM model via a system containing machines  $M_{c_{\pi}}, M_{c_1}, \ldots, M_{c_{n-1}}$ , where  $\mathsf{Codes} = \{c_{\pi}, c_1, \ldots, c_{n-1}\}$  and instances of  $M_c$  essentially run code  $c \in \mathsf{Codes}$ ; see the left hand-side of Figure 1 for an illustration of the static structure, i.e., the set of machines and I/O tape connections, of the mapped protocol system  $\pi_{IITM}$ . Just as  $\pi_{UC}$ ,  $\pi_{IITM}$  is able to create an unbounded number of instances of these machines running any of the codes in **Codes** (see below). Each pair of machines  $M_c, M_{c'}$  is connected by a pair (t, t') of uniquely named internal I/O tapes. One of the tapes, say t, is used by (instances of)  $M_c$  to provide subroutine inputs to and receive subroutine outputs from  $M_{c'}$ , while the other tape is used for the reverse direction where  $M_{c'}$  provides subroutine inputs to and receives subroutine outputs from  $M_c$ .<sup>13</sup> Altogether these connections allow instances of an arbitrary machine to send inputs and outputs to and receive outputs and inputs from any (instance of) another machine in the system simply by choosing the appropriate tape. While generally not required by protocols from the literature, if required by  $\pi_{UC}$  we can also extend the protocol  $\pi_{IITM}$  to allow for sending messages between different instances of the same machine. This is done by adding a special bouncer machine  $M_{\rm bc}$  to the system  $\pi_{IITM}$ .  $M_{\rm bc}$  connects to all machines in the system via a pair of I/O tapes each. Each time (a session-specific instance of) this machine receives a message, it returns the same message on the same tape. Hence, a machine  $M_c$  can send a message to  $M_{\rm bc}$  to effectively send that message to an instance of itself (see below for how we ensure that this message is delivered to the correct receiving instance of  $M_c$ ). Altogether, these internal I/O tapes bridge Diff. 2 and Diff. 3. In addition to these internal I/O tapes, each machine  $M_c$  has one external (unconnected) network tape that can be used to communicate with the network adversary. The machine  $M_{c_{\pi}}$  also has an unconnected I/O tape which can be used to receive inputs from and send outputs to higher-level protocols/the environment. These external tapes capture permitted communication flows between the protocol, the adversary, and the environment as defined in the security game of the UC model.

In addition to the above machines, we also add another machine  $M_{\text{msg}}$ . Jumping slightly ahead, session specific instances of this machine are responsible for (i) implementing some of the more advanced message transmission and message redirection features of the UC model and (ii) forcing the environment to be balanced, i.e., to provide a minimal amount of import to the adversary. This machine connects via a pair of I/O tapes to all machines  $M_c$ ,  $c \in \mathsf{Codes}$ , and offers one external network tape for communication with the adversary. We describe the behavior of  $M_{\text{msg}}$  along with the description of machines  $M_c$  below.

Addressing of instances. In  $\pi_{UC}$ , an instance of a machine running code c is uniquely addressed by an ID of the form (pid, sid) and learns the ID  $(pid_s, sid_s)$ and code  $c_s$  of senders who provide input or subroutine output. Furthermore, by our initial normalization of  $\pi_{UC}$ , we have that  $sid = (sid_c, sid')$  for internal instances, i.e., instances running code  $c \neq c_{\pi}$ , and  $sid = sid_c$  for instances running

<sup>&</sup>lt;sup>13</sup> Typically, the subroutine relation goes only in one direction and in this case just one tape is needed. But in general the relationship is allowed to go both ways, in which case using two tapes allows for distinguishing which relationship is meant.

16



**Fig. 1.** Left: Static structure of a protocol  $\pi_{IITM}$  using three Codes =  $\{c_{\pi}, c_1, c_2\}$  constructed by our mapping. Right: Static structure of the modified protocol  $\pi_{IITM}^{\xi \text{-id}}$  that enforces  $\xi$ -identity bounded environments (cf. Section 3.3). Lines denote internal connections via I/O tapes and the external I/O tape to the environment. Each Machine also has an external network tape to the adversary (not shown). In a run each machine can be instantiated arbitrarily often, with instances having IDs of the form (*pid*, *sid*).

 $c_{\pi}$ . To capture these unique IDs for instances in  $\pi_{IITM}$ , we use a suitable instantiation of the CheckAddress mode, cf. Figure 2. That is, instances of  $M_c$  expect incoming inputs/outputs m to be of the form  $((pid, sid, c), (pid_s, sid_s, c_s), m')$ , where m' is the actual message body.<sup>14</sup> Network messages from the adversary are expected to be of the form ((pid, sid, c), m').<sup>15</sup> Furthermore, if  $c \neq c_{\pi}$  (i.e., the current machine instance is an internal subroutine), then it is also required that sid = (sid', sid'') for some sid', sid'', where sid' is interpreted to be the SID  $sid_c$  of the protocol session. Messages not conforming to this format are rejected immediately. If the current instance is fresh, i.e., has not previously accepted any messages, then the message is accepted and (in mode **Compute**) this instance stores (*pid*, *sid*) as its own ID. If the instance is not fresh, i.e., has previously accepted a message with receiver ID  $(pid_0, sid_0)$ , then incoming messages are accepted if and only if they are prefixed by the same ID, i.e.,  $pid = pid_0$ ,  $sid = sid_0$ . Hence, each instance is effectively assigned a unique ID, namely, the first ID  $(pid_0, sid_0)$  that it has ever accepted. There will also never be a second instance accepting the same ID since all message for this ID will already be accepted by the first instance with that ID. Given this definition of CheckAddress, an instance  $(pid_s, sid_s)$  of machine  $M_{c_s}$  can send a message m' to the unique instance

<sup>&</sup>lt;sup>14</sup> The only exception are inputs received on the single external I/O tape from the environment, which use the header  $((pid, sid), (pid_s, sid_s, c_s), m')$ . This directly corresponds to the UC experiment, where environments specify only the receiver ID (pid, sid) but not the receiver code, which is rather determined by the experiment. We also note that, except for outputs returned from the protocol to the environment, it is actually not necessary to include c in the header of any messages on I/O tapes. After all, the receiving machines  $M_c$  are already aware of their own code. We chose to nevertheless include c in the header since this matches the format of write commands in the UC model more closely.

<sup>&</sup>lt;sup>15</sup> Network messages do not contain a sender identity since the sender is always know to be the network adversary.

Mode CheckAddress:

Let $m$ be the message received or	n some tape.
If this is the highest level machine	(i.e., $c = c_{\pi}$ ) and m was received on the single external
I/O tape from the environment,	then try to parse $m$ as $((pid, sid), (pid_s, sid_s, c_s), m')$ .
Otherwise, if $m$ was received	on another I/O tape then try to parse it as
$((pid, sid, c), (pid_s, sid_s, c_s), m').$	Try to parse $m$ as $((pid, sid, c), m')$ if it was received
	e, if $c \neq c_{\pi}$ , also try to parse <i>sid</i> as $(sid', sid'')$
	(id is a global variable that is $\perp$ iff this instance is
if $id = \perp \lor id = (pid, sid)$ then $\langle$	fresh. It is set to be the ID of the current instance
return accept.	in mode Compute upon accepting the first message.
else	
return reject.	
end if	

**Fig. 2.** Checkaddress mode of the machine  $M_c$  for  $c \in \mathsf{Codes}$ 

(pid, sid) of machine  $M_c$  by writing the message  $((pid, sid, c), (pid_s, sid_s, c_s), m')$  on one of the two tapes connecting to  $M_c$  (this bridges Diff. 4).

All machines  $M_c$  are defined in such a way that they never lie about the sender identity of a message, and hence, the receiver always learns the correct identity of the sender (see the **Compute** mode described below). Specifically, if an instance  $(pid_s, sid_s)$  of a machine  $M_{c_s}$  sends a message m on some tape, then it will either be of the form  $((pid, sid, c), (pid_s, sid_s, c_s), m')$  (if it is sent on an I/O tape) or of the form  $((pid_s, sid_s, c_s), m')$  (if it is sent on a network tape connected to the adversary). This bridges Diff. 5 by providing the same level of authentication of the sender instance in  $\pi_{IITM}$  as in  $\pi_{UC}$ .

**Runtime behavior.** The Compute mode of a machine  $M_c$  is mostly a direct implementation of the protocol logic given by code c (cf. Figure 3). Upon its first activation in this mode an instance (pid, sid) of  $M_c$  stores its own ID (pid, sid) in a global variable *id*. The machine then checks, also during subsequent activations, if it has already received any inputs/outputs on an I/O tape and stops the activation otherwise. This captures that the network adversary in the UC model is not allowed to spawn new machine instances. That is, even though spawning a new protocol machine instance is technically possible in  $\pi_{IITM}$ , the resulting instance will not do anything until it receives the first input or output from another protocol machine or the environment, which results in a behavior that is equivalent to the one in the UC model (this bridges Diff. 6). Once it receives its first input/output on an I/O tape (and therefore the corresponding instance in  $\pi_{UC}$  is created), the instance registers itself with the instance  $(\epsilon, sid_c)$  of  $M_{\rm msg}$  by sending  $((\epsilon, sid_c, c_{M_{msg}}), (pid, sid, c), \texttt{register})$  on an I/O tape connected to  $M_{msg}$ . This instance, which is specific to the protocol session  $sid_c$ , stores the ID (pid, sid, c)and immediately returns an acknowledgement. Finally, if this is an instance of the highest-level machine  $M_{c_{\pi}}$  and it receives some import i > 0 in a message from the environment, then it sends  $((\epsilon, sid_c, c_{M_{msg}}), (pid, sid, c), (notifyImport, i))$ to notify the session specific instance  $M_{\rm msg}$  about this amount.  $M_{\rm msg}$  stores iand returns an acknowledgement; we describe the purpose of registrations and import notifications later on.

Mode Compute: Let  $m = ((pid, sid, c), (pid_s, sid_s, c_s), m')$  be the message received on some I/O tape t respectively m = ((pid, sid, c), m') received on the network tape. if  $id = \bot$  then Store the ID of this instance such that the  $id \leftarrow (pid, sid)$ CheckAddress mode can use this information. end if if this instance has not received any message on an I/O tape yet then Stop the current activation of this instance. { This activates the environment. end if  ${\bf if}$  this is the first message received on an I/O tape  ${\bf then}$ Send  $((\epsilon, sid_c), (pid, sid, c), register)$  on the tape connected to  $M_{msg}$ , where  $sid_c$ can be parsed from *sid*. Wait for the response and then continue. end if if  $c = c_{\pi}$  and m' is a message on the external I/O tape containing import i > 0 then Send  $((\epsilon, sid_c), (pid, sid, c), (notifyImport, i))$  on the tape connected to  $M_{msg}$ , where  $sid_c$  can be parsed from sid. Wait for the response and then continue. end if // Main logic // Run code c using the sender information  $(pid_s, sid_s, c_s)$  (or  $\epsilon$  if the message is from the network adversary), incoming message m', and the tape type  $tt~\in$ {input, output, backdoor} that m' is written on determined from the tape t. When c wants to send a message, proceed as described in the paragraph "Sending messages" on Page 18. In particular, ensure that the resulting message contains the correct sender identity (pid, sid, c) in the header.

**Fig. 3.** Compute mode of the machine  $M_c$  for  $c \in \mathsf{Codes}$ 

Once all of the above steps are finished (and the instance has not aborted), the instance processes the incoming message m by running the code c. Note that this is indeed possible:  $M_c$  can determine whether m is an input, subroutine output, or a backdoor message depending on the tape m is received on. Inputs and outputs received from other protocol machines also contain the full extended identity of the sending instance, including the machine code, so  $M_c$  has access to the same information that instances in  $\pi_{UC}$  have in the UC model upon receiving a new message.

Sending messages. During the simulation of code c, whenever the code c wants to provide input/output m' to an instance (pid', sid') of a machine  $M_{c'}$ ,  $M_c$  chooses the I/O tape t that connects  $M_c$  and  $M_{c'}$  and which models an input/output from  $M_c$  to  $M_{c'}$ . Then  $M_c$  writes ((pid', sid', c'), (pid, sid, c), m') on tape t, where (pid, sid) is the ID of the current instance of  $M_c$ . If the code c wants to send a backdoor message m' to the network adversary,  $M_c$  writes the message ((pid, sid, c), m') on its network tape.

We still have to explain how we deal with Diff. 7. That is, code c might choose to use a non-forced-write command and specify the recipient of a message not by their extended ID but by a predicate P on extended identities. First, observe that if a message is sent to a backdoor tape, then it must be for the network adversary by definition of the UC security experiment. Hence, this case is easy to handle in  $M_c$ : if the non-forced-write request is to a backdoor tape, then the

message is sent as described above to the network adversary. Second, for inputs and outputs observe that those may not be sent directly to the identities of the environment or the adversary. So the predicate may match only identities of (existing) machines within the protocol, i.e., the message will be sent internally. We can easily mimic this in the IITM model via the machine  $M_{\rm msg}$ . Recall that, by the above construction, whenever a new machine instance receives the first input or output on an I/O tape in mode Compute, it registers its extended identity (pid,sid,c) at (a session dependent instance of)  $M_{\rm msg}.$  The machine  $M_{\rm msg}$  offers a "nonForcedWrite" command to the machines  $M_c$  that, given message body m', message type  $mt \in \{\text{input}, \text{output}\}$ , and predicate P, runs the predicate P on the list of existing protocol machine instances to find the first matching one. The message m is then delivered to that instance as described above, but with the I/O tape chosen based on mt and the sender of the message (which is written in the header of the message) set to be the machine instance (pid, sid, c) that called the **nonForcedWrite** command. If no matching instance is found, then  $M_{\rm msg}$  aborts and the environment is activated instead, just as in the UC model.

There is another special case that we have to deal with, namely the highestlevel protocol machine  $M_{c_{\pi}}$  sending a subroutine output (cf. Diff. 8). In the UC model, this output is redirected to the environment (without the code of the sender but instead including the code of the intended receiver) iff the current instance has challenge SID  $sid_c$  and the receiver extended identity does not yet exist as an instance of a protocol machine. Observe that by our normalization of  $\pi_{UC}$  all instances of  $M_{c_{\pi}}$  that are part of protocol session  $sid_c$  also have SID  $sid_c$ , i.e., the first condition is always met. The second condition can be checked using the information stored in the machine  $M_{msg}$ , yielding the following implementation. Whenever an instance  $(pid_s, sid_s)$  of  $M_{c_{\pi}}$  wants to send a subroutine output m' to an extended receiver identity  $eid_r = (pid_r, sid_r, c_r)$ ,  $M_{c_{\pi}}$  first asks  $M_{\rm msg}$  whether  $eid_r$  already exists in the system (via a special existsInstance? request). If so, the message is sent by  $M_{c_{\pi}}$  to the instance  $(pid_r, sid_r)$  of machine  $M_{c_r}$  as described above. If this instance does not exist yet, then the message  $m = ((pid_r, sid_r, c_r), (pid_s, sid_s), m')$  is sent on the single external I/O tape of  $M_{c_{\pi}}$  that is connected to the environment. Note that, unlike for other messages, the sender machine code  $c_{\pi}$  is not contained in the header of m in this case. Altogether, this precisely captures the behavior of the UC security experiment and hence bridges Diff. 8.

Import handling. We still have to explain the purpose of the notifyImport message. Instances of  $M_{\rm msg}$  use these notifications to keep track of the list of imports received from the environment in this protocol session. The adversary can send a special totalImport? request to learn the current list of imports. Jumping slightly ahead, this information will be used by the simulator constructed in Section 3.3 to bridge Diff. 11: Instead of requiring the environment in the IITM model to be balanced (i.e., it has to provide at least the same amount of import to the simulator as it provides to the protocol), the simulator rather indirectly enforces this property itself. That is, the simulator checks how much import the protocol has received already and, if the protocol has received more

than the simulator, adds the missing difference to its own received import. We note that the security notion of the UC model requires runtime bounds to be simulated correctly and hence adversaries/simulators generally must already be aware of the current protocol imports not just for the whole session but even for individual (highest-level) instances in a session. The added totalImport? request only makes this property explicit via a fixed mechanism. Nevertheless, we show in Appendix H that our results can actually also be obtained without adding a totalImport? request. This, however, requires a more involved mapping than the one we present here.

Finally, we encode runtime import for the machine codes c in unary instead of binary. This seemingly cosmetic change does not affect the behavior or security results obtained for the protocol  $\pi_{UC}$ . But it allows us to argue that an environment in the IITM model, which may send arbitrary inputs of at most polynomial length to the protocol, can send at most a polynomial amount of import just as an environment in the UC model.

Altogether, we define  $\pi_{IITM} := !M_{c_{\pi}} | !M_{c_1} | \dots | !M_{c_{n-1}} | !M_{msg} | !M_{bc}$ . Based on the construction and the discussion above, we can easily check that  $\pi_{UC}$  and  $\pi_{IITM}$  behave the same:

**Lemma 1.** For all unbounded (including runtime) environments interacting with  $\pi_{UC}/\pi_{IITM}$  by sending inputs/receiving outputs but also by directly interacting with arbitrary protocol instances over the network, there is a bijective mapping between runs of  $\pi_{UC}$  in the UC model and  $\pi_{IITM}$  in the IITM model such that both protocols behave identically. Both protocols have similar computational complexity.

*Proof.* By construction, the only difference between both protocols is the added explicit totalImport? request on the network in  $\pi_{IITM}$ . In the UC setting with  $\pi_{UC}$  this request can instead be internally simulated by the environment.

We show in the next lemma (proven in Appendix A) that  $\pi_{IITM}$  is a welldefined IITM protocol by showing that it meets the IITM runtime notion for protocols. This bridges Diff. 9 by relating the UC to the IITM runtime notion.

**Lemma 2.** The protocol  $\pi_{IITM}$  is environmentally bounded in the IITM model if  $\pi_{UC}$  is ppt in the UC model.

#### 3.3 UC Security Implies IITM Security

Having defined a mapping of protocols from the UC to the IITM model, we now prove that this mapping preserves security results. That is, if  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$ , then  $\pi_{IITM}^{\xi \cdot \mathrm{id}} \leq_{IITM} \phi_{IITM}^{\xi \cdot \mathrm{id}}$  for protocols mapped as described in Section 3.2 plus an additional mechanism to capture  $\xi$ -identity bounded environments in the IITM model, which unlike the UC model does not restrict environments. This mechanism does not change the IITM model. We rather show that  $\xi$ -identity bounded behavior can be enforced within protocols themselves, thereby bridging Diff. 10.

While designing this mechanism, we found that the definition of  $\xi$ -identity bounded environments as used in the UC model actually does not support composition and the proof of the UC composition theorem is flawed. We describe the issue in detail in Appendix J.3. In a nutshell, the issue is that the UC model allows for defining the identity set  $\xi$  via a predicate over the current configuration of the whole system. The configuration of the system and hence potentially the behavior of the predicate is very different in the security experiment, where there are only instances of the environment, adversary, and one session of  $\pi_{UC}$  respectively  $\phi_{UC}$ , compared to the composition theorem, where there are additional instances of a higher-level protocol  $\rho$  as well as potentially multiple sessions of  $\pi_{UC}/\phi_{UC}$ . Hence, even if  $\rho$  is  $\xi$ -compliant in the setting where instances of  $\rho$  and multiple sessions of  $\pi_{UC}/\phi_{UC}$  exist, this does not imply that an environment internally simulating  $\rho$  while running only with  $\pi_{UC}/\phi_{UC}$  (but with no actual instances of  $\rho$  and only a single session of  $\pi_{UC}/\phi_{UC}$  being present in the system) also is  $\xi$ -identity-bounded. Based on this observation, in Appendix J.3 we show a concrete counterexample for the UC composition theorem.

Therefore, instead of trying to translate the existing identity-bounded mechanism, which does not support composition in the UC model, and hence, would also not support composition when faithfully translated to the IITM model, we propose a fix for the UC model and then transfer that fixed version to the IITM model. Specifically, instead of defining  $\xi$  as a predicate over the configuration of the whole system, we define it as a predicate over the (whole history of) inputs sent and outputs received by the environment/ $\rho$  to/from one session of the subroutine  $\pi/\phi$ . This fix, which follows a similar idea as [1], indeed solves the problem: The sequence of messages between  $\rho$  and one of its subroutine sessions remains the same (for each respective subroutine session) even if we only simulate  $\rho$  within an environment. Hence, such an environment running directly with a single session of the subroutine  $\pi/\phi$  is indeed  $\xi$ -identity-bounded. This fixes this issue of the UC composition theorem and the proof thereof. This fix should also be sufficient for practical purposes; we are not aware of any protocols that have been proven secure for a  $\xi$  that falls outside this class. We provide an extended discussion in Appendix J.3.

We now embed this (fixed) definition of  $\xi$ -identity-bounded environments into the IITM model as follows. The obvious option would be to restrict environments in the IITM model in the same way. However, this would require us to change the IITM model and its theorems and proofs. We rather extend the protocols  $\pi_{IITM}$ and  $\phi_{IITM}$  in a generic way to manually enforce the  $\xi$ -identity-bounded property for all environments. This is a technique that is commonly used in the IITM model, see for example [16, 18]. Formally, we add to each protocol an additional dummy forwarder machine  $M_{identity}^{\xi}$  between the environment and the highestlevel machine  $M_{c_{\pi}}$  respectively  $M_{c_{\phi}}$ , creating new protocols  $\pi_{IITM}^{\xi-id}$  and  $\phi_{identity}^{\xi-id}$ (cf. right hand-side of Figure 1). In a run, (a session specific instance of)  $M_{identity}^{\xi}$ checks for every input whether  $\xi$  is met and, if not, drops the input, thereby activating the environment shat are already  $\xi$ -identity-bounded are not restricted

since for such environments the original protocols  $\pi_{IITM}/\phi_{IITM}$  and the modified protocols  $\pi_{IITM}^{\xi\text{-id}}/\phi_{IITM}^{\xi\text{-id}}$  behave identically. For any other environment  $\mathcal{E}$ , the combination of  $\mathcal{E}$  and  $M_{\text{identity}}^{\xi}$  constitutes a  $\xi$ -identity-bounded environment for the original protocol. Note that the extended protocols are still environmentally bounded as  $M_{\text{identity}}^{\xi}$  adds only a polynomial number of steps; in particular,  $\xi$ can be evaluated in polynomial time by definition. Altogether, this mechanism indeed bridges Diff. 10.

We can now show that  $\leq_{UC}$  security implies  $\leq_{IITM}$  security for the mapped protocols; we discuss the reverse implication afterwards. In Appendix H, we show that  $\leq_{UC}$  implies  $\leq_{IITM}$  in general by using a somewhat more involved protocol embedding. Here, using the (simpler) protocol embedding from Section 3.2, we formally show this result for a certain though very general class of simulators, in fact, a class of simulators containing virtually all simulators that have ever been considered in the literature so far, as further explained below.

More specifically, first recall that, as stated in the UC model, to prove  $\pi \leq_{UC} \phi$  instead of constructing a simulator for every adversary, it suffices to construct a simulator just for the dummy adversary. (From such a simulator, simulators for arbitrary adversaries can be constructed.) The dummy adversary as considered in the UC model allows the environment to provide import i via a special message, say op(i), which is different from network messages intended for the protocol. The dummy accepts this import and returns an acknowledgement to the environment without sending a message to the protocol. We therefore consider the class of simulators which also do not require an interaction with the ideal protocol upon receiving import via op from the environment. This is a natural requirement that should be trivially met by simulators for all reasonable protocol definitions, also considering that a protocol in reality cannot rely on the network adversary sending a notification each time the adversary decides to increase its runtime bound. Indeed, we are not aware of any UC protocols from the literature where the simulator has to interact with the ideal protocol upon receiving additional import via op. Simulators are rather defined in a black-box fashion where they implicitly simulate the dummy adversary and only specify their behavior for network messages that are forwarded by the dummy to the real protocol. Since the dummy adversary already handles the input op without sending any network messages to the protocol, all such black-box simulators trivially have the stipulated property. We note again that, as mentioned above, this (though natural) requirement on simulators is not formally necessary.

**Theorem 4.** Let  $\pi_{UC}$ ,  $\phi_{UC}$  be such that  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$ . Then it holds true that  $\pi_{IITM}^{\xi \text{-}id} \leq_{IITM} \phi_{IITM}^{\xi \text{-}id}$ .

*Proof (sketch).* We here show this theorem assuming that the simulator for proving  $\pi_{UC} \leq \xi_{UC} \phi_{UC}$  has the properties stipulated above and point to Appendix H for the general case. The proof proceeds in 4 steps (see Appendix B for the full proof):

**Reduction to UC.** We first define an IITM dummy adversary  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ and an IITM simulator  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  that adhere to the UC runtime notion and

23

enforce the balanced requirement for environments. Specifically, both machines are defined to internally run the UC (real and ideal) adversaries  $\mathcal{A}_{Dum,UC}$  and  $\mathcal{S}_{UC}$ , respectively, but add a wrapper around them. This wrapper handles the added totalImport? request on the network itself by forwarding it to  $M_{msg}$  and returning the response without involving the internally simulated UC adversary. Also, upon each activation the wrapper first checks whether its protocol has received one or more new imports (via a call to totalImport?) such that its total import now exceeds the total import directly provided by the environment to the adversary. If so, the wrapper adds these missing imports to the internally simulated UC adversary via (potentially several calls to) the operation *op*. Then, and in all other cases, the adversary continues as the internal UC adversary.

Consider an IITM environment  $\mathcal{E}_{IITM}^{\text{single},\xi}$  that sends inputs and network messages (via the dummy adversary) only to a single session of the proto-Inclusing control of the duffing adversary) only to a single session of the proto-col  $\pi_{IITM}/\phi_{IITM}$ , adheres to the  $\xi$ -identity bound, and tries to distinguish the worlds  $\mathcal{A}_{Dum,IITM}^{UC-bounded} | \pi_{IITM}$  and  $\mathcal{S}_{IITM}^{UC-bounded} | \phi_{IITM}$ . We can reduce this case to the indistinguishability of  $\mathcal{A}_{Dum,UC} | \pi_{UC}$  and  $\mathcal{S}_{UC} | \phi$  in UC by constructing an UC environment  $\mathcal{E}_{UC}$  that internally simulates  $\mathcal{E}_{IITM}^{single,\xi}$ .  $\mathcal{E}_{UC}$  further internally simulates responses to totalImport? requests. Each time  $\mathcal{E}_{IITM}$  wants to provide import as part of an input to the protocol such that the total protocol import exceeds the total import provided to the adversary so far,  $\mathcal{E}_{UC}$  first adds the missing difference via a call to op to the adversary and only then sends the input to the protocol. By construction,  $\mathcal{E}_{UC}$  is balanced. To see that  $\mathcal{E}_{UC}$  has the same distinguishing advantage as  $\mathcal{E}_{IITM}^{\text{single},\xi}$ , there are only two aspects that we have to argue. Firstly, in the IITM setting a protocol might obtain one or more imports that bring the total above the amount of import of the adversary. Then, as soon as the adversary wrapper becomes active the next time, it calls op for each of these imports, and then the internally simulated adversary processes the message. In the UC world,  $\mathcal{E}_{UC}$  first calls *op*, then provides import to the protocol. This might be repeated several times until, at some point, the adversary processes whatever message  $\neq op$  it receives next. So while the same number of calls to opwith the same import are used in both UC and IITM setting, formally the state of the protocol might be different when op is executed. Due to the definition of the dummy and assumption on the simulator, op is independent of the state of the protocol, i.e., this formal difference does not actually affect the behavior of the run. (This is the only case where a slight mismatch occurs. All other messages are processed at the same points in the run by construction.) Secondly, the UC environment is bounded in its current import, so might not be able to complete the simulation. We can find an external input of suitable length, which determines the initial import, such that this case does not occur.

**Environments without the**  $\xi$ -identity bound. The indistinguishability of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}$  for environments  $\mathcal{E}_{IITM}^{\text{single},\xi}$  is easily seen to be equivalent to indistinguishability of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi\text{-id}}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi\text{-id}}$  for arbitrary single session environments  $\mathcal{E}_{IITM}^{\text{single}}$ .

Indistinguishability for the IITM dummy. So far, we have only considered the dummy  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  which adheres to the UC runtime notion and hence might stop whenever he has to forward more bits than he has import. However, we actually have to show  $\leq_{IITM}$  for the IITM dummy  $\mathcal{A}_{Dum,IITM}$  which never stops and always forwards messages. The idea for constructing a simulator  $\mathcal{S}_{IITM}$  for  $\mathcal{A}_{Dum,IITM}$  is as follows: Observe that the only difference between  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ and  $\mathcal{A}_{Dum,IITM}$  is that  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  might stop if it has too little import, which  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  then also simulates. So we define the simulator  $\mathcal{S}_{IITM}$  to internally run  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  but, upon each activation, potentially generate additional import via a call to *op* such that an imaginary  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ , if given the same overall import, would not stop. We show that it is indeed possible to build such a simulator, also while remaining in the polynomial runtime notion of the IITM model (this is because the additional import is polynomial in the runtime of the environment and hence the same argument as in Lemma 2 still applies).

We can then reduce a single session environment  $\mathcal{E}_{IITM}^{\text{single}}$  trying to distinguish  $\mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi \cdot \text{id}}$  and  $\mathcal{S}_{IITM} | \phi_{IITM}^{\xi \cdot \text{id}}$  to indistinguishability of the worlds  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi \cdot \text{id}}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi \cdot \text{id}}$  by constructing an environment  $\mathcal{E}_{IITM}^{\prime \text{single}}$  that internally simulates  $\mathcal{E}_{IITM}^{\text{single}}$  plus the additional import generated by the wrapper portion of  $\mathcal{S}_{IITM}$ .

Indistinguishability of multiple sessions. Since the protocols have disjoint sessions and  $\mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$  and  $\mathcal{S}_{IITM} | \phi_{IITM}^{\xi\text{-id}}$  are indistinguishable for any environment  $\mathcal{E}_{IITM}^{\text{single}}$  interacting with just a single session, the second composition theorem of the IITM model (cf. Theorem 3) immediately implies that  $\pi_{IITM}^{\xi\text{-id}} \leq_{IITM} \phi_{IITM}^{\xi\text{-id}}$ , i.e., there also exists a simulator for arbitrary environments  $\mathcal{E}_{IITM}$  interacting with multiple sessions.

The construction of the simulator  $S_{IITM}^{\text{UC-bounded}}$  in the above proof bridges Diff. 11: Since the IITM model does not require that environments provide a certain minimal amount of import to the adversary (the IITM model does not even require the concept of import), the simulator instead enforces this property itself by manually adding the difference between its own import and the import received by the protocol. The above proof also bridges Diff. 12 by showing that the UC security notion implies the single session IITM security notion. The second composition theorem of the IITM model (cf. Theorem 3) then directly implies security for multiple sessions.

The other implication of Theorem 4 is more involved since the IITM model considers a larger class of adversaries, including simulators, than the UC model. Specifically, the runtime of UC simulators is required to be bounded by a fixed polynomial (in their current import) independently of the environment. An IITM simulator does not need to adhere to any import mechanism. Its polynomial runtime bound is rather taken over  $\eta$  and the length of the external input *a* and may even depend on the environment. In fact, the following lemma shows that the reverse implication of Theorem 4 does not hold true in general:

**Lemma 3.** If time-lock puzzles exist, then there exist protocols  $\pi_{UC}$  and  $\phi_{UC}$  such that for the mapped protocols we have  $\pi_{IITM}^{\xi \cdot id} \leq_{IITM} \phi_{IITM}^{\xi \cdot id}$  but  $\pi_{UC} \leq_{UC} \phi_{UC}$  does not hold true. (These protocols are pretty simple, and hence, the result works for all mappings that preserve the protocols behaviors.)

We recall the definition of time-lock puzzles and formally prove this result in Appendix C, along with a discussion on the implications for security results. If we consider only the subclass of IITM simulators that corresponds to the class of UC simulators that adhere to the UC runtime notion, such as  $S_{IITM}^{\text{UC-bounded}}$  constructed in the proof of Theorem 4, we have the following reverse implication:

**Theorem 5.** Let  $\mathcal{A}_{Dum,IITM}^{UC\text{-bounded}}$  be the IITM dummy adversary that enforces balanced environments and adheres to the UC runtime notion as defined in the proof of Theorem 4. Let  $\mathcal{S}_{IITM}^{UC\text{-bounded}}$  be an IITM simulator that is of the form as the one described in the proof of Theorem 4.

one described in the proof of Theorem 4. If  $\mathcal{A}_{Dum,IITM}^{UC-bounded} | \pi_{IITM}^{\xi-id}$  and  $\mathcal{S}_{IITM}^{UC-bounded} | \phi_{IITM}^{\xi-id}$  are indistinguishable for all IITM environments interacting with a single session of the protocol, then we have  $\pi_{IITM}^{\xi-id} \leq_{IITM} \phi_{IITM}^{\xi-id}$  (multi session IITM security) as well as  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$ .

We provide the proof in Appendix D. Theorem 5 shows that the implication of Theorem 4 is non-trivial and non-degenerate since our mapping not only preserves security results but also distinguishing attacks. That is, if for all UC simulators there is a  $\xi$ -identity bounded UC environment that distinguishes  $\pi_{UC}$  and  $\phi_{UC}$ , then Theorem 5 implies that for all IITM simulators in the UC runtime class there is an IITM environment distinguishing  $\pi_{ITM}^{\xi-\mathrm{id}}$  and  $\phi_{UTM}^{\xi-\mathrm{id}}$ .

#### 3.4 UC Composition Implies IITM Composition

In this section, we investigate in how far composition results carry over from UC to IITM. We first observe the following direct corollary of Theorem 4:

Corollary 1 (Composition from the UC theorem). Let  $\pi_{UC}$ ,  $\phi_{UC}$ ,  $\rho_{UC}$  be UC protocols such that  $\pi_{UC} \leq {\xi \atop UC} \phi_{UC}$  and the UC composition theorem can be applied to  $\rho_{UC}$  to obtain  $\rho_{UC}^{\phi \to \pi} \leq_{UC} \rho_{UC}$ . Let  $\rho_{IITM}$  and  $\rho_{IITM}^{\phi \to \pi}$  be the IITM protocols obtained by applying the mapping from Section 3.2 to  $\rho_{UC}$  and  $\rho_{UC}^{\phi \to \pi}$ .<sup>16</sup> Then  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$ .

*Proof.* Directly follows from Theorem 4 by observing that the simulator constructed by UC composition has the property required by Theorem 4 (we discuss this in detail in Appendix E). Alternatively, one can use the extended mapping from Appendix H which does not require a specific UC simulator.  $\Box$ 

<sup>&</sup>lt;sup>16</sup> Note that  $\rho _{UC}^{\phi \to \pi}$  also contains some UC composition shell code introduced by the UC composition theorem to replace the code  $c_{\phi}$  with  $c_{\pi}$ .  $\rho _{IITM}^{\phi \to \pi}$  is thus obtained by mapping the overall machine codes, including the UC composition shell code.

While this corollary shows that security results obtained via the UC composition theorem carry over, it does not actually provide insights into how the UC and IITM composition theorems relate. To answer this question, we next show that the same composition statement can be obtained directly from the IITM composition theorem without relying on the UC theorem.

Obtaining Corollary 1 from the IITM composition theorem. We start by observing that the IITM theorem requires that higher-level protocols access the subroutine  $\pi_{IITM}/\phi_{IITM}$  only via its external I/O interface, i.e., the external I/O tapes that the environment had access to in absense of the higher-level protocol. In the special case of our mapped protocols  $\pi_{IITM}/\phi_{IITM}$ , which offer only a single external I/O tape to/from the machine with code  $c_{\pi}/c_{\phi}$ , this syntactical requirement of the IITM theorem actually corresponds to the "subroutine respecting" requirement for  $\pi_{UC}/\phi_{UC}$  in the UC theorem.<sup>17</sup> That is, subroutine respecting protocols are required to reject and drop all messages from and never send messages to instances outside of their session of  $\pi_{UC}/\phi_{UC}$ , except for inputs to and outputs from highest-level instances running code  $c_{\pi}/c_{\phi}$ . The only difference is that in UC "subroutine respecting" is a semantic requirement imposed on the behavior of machines whereas the IITM requirement enforces the same property on the syntactical level of interfaces by removing any unintended communication channels/tapes. Hence, to be able to apply the IITM composition theorem and conclude  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$  we have to make some slight syntactical adjustments to  $\rho_{\mathit{IITM}}$  such that the semantic "subroutine respecting" property is also reflected by the tape connections.

So let  $\rho_{IITM}$  be the protocol mapped according to Section 3.3. Then, since  $\rho_{IITM}$  uses code  $c_{\phi} \in \mathsf{Codes}$  and possibly other codes,  $\rho_{IITM}$  looks like depicted in Figure 4 (left-hand side); we refer to machines of the system  $\rho_{IITM}$  by  $M_i^{\rho}$ . The middle picture of Figure 4 illustrates the idea of our syntactical changes to  $\rho_{IITM}$ : We extend the protocol  $\rho_{IITM}$  by including the full set of machines of  $\phi_{IITM}$ , as obtained by mapping from  $\phi_{UC}$  according to Section 3.3. We now change all machines in  $\rho_{IITM}$ , i.e., all  $M_i^{\rho}$ , to send inputs/receive outputs to/from  $\phi_{IITM}$  instead of  $M^{\rho}_{c_{\phi}}.$  Since multiple machines need to connect to  $\phi_{IITM}$  but  $\phi_{IITM}$  provides only a single external I/O tape, we introduce a straightforward multiplexer  $M_{\text{multiplex}}$  that forwards messages between  $\phi_{IITM}$  and machines  $M_i^{\rho}$ . Since inputs to and outputs from  $M_{c_{\phi}}^{\rho}$  are the only way for higher-level instances in  $\rho$  to interact with instances in any session of the subroutine  $\phi$  (by the subroutine respecting property), this syntactic modification of  $\rho_{IITM}$  does not actually change its behavior. It, however, consistently moves all sessions of  $\phi$  to now be instances of the set of machines  $\phi_{IITM}$ . Note that when  $\phi_{IITM}$ calls subroutines (with code in Codes), then  $\phi_{IITM}$  now uses its own subrountine machines, instead of those of  $\rho_{IITM}$ . The composed protocol  $\rho_{IITM}^{\phi \to \pi}$  is then

<sup>&</sup>lt;sup>17</sup> The IITM composition theorem also supports IITM protocols that offer several external I/O tapes, even to subroutines, which gives the environment and higher-level protocols direct access to those subroutines. Such IITM protocols are more general. They do not and do not have to meet the "subroutine respecting" property.



Fig. 4. Overview of the static structures of the protocols in this section. Left:  $\rho_{IITM}$  mapped as per Section 3.2. Middle:  $\rho_{IITM}$  after redirecting all inputs/outputs from  $M_{c_{\phi}}^{\rho}$  to  $\phi_{IITM}$ . The machine  $M_{c_{\phi}}^{\rho}$  is formally still present but not used in a run. Right: The composed protocol  $\rho_{IITM}^{\phi \to \pi}$  after applying the IITM composition theorem, which replaces the protocol (and hence all sessions of)  $\phi_{IITM}$  with the protocol  $\pi_{IITM}$ .

defined by simply replacing the set of machines  $\phi_{IITM}$  with the set of machines  $\pi_{IITM}$  (right hand-side of Figure 4). This is as simple as reconnecting the single I/O tape between the multiplexer  $M_{\text{multiplex}}$  and  $\phi_{IITM}$  to instead connect to the external I/O tape of  $\pi_{IITM}$ . As a result, in  $\rho_{IITM}^{\phi \to \pi}$  all inputs to and outputs from sessions of  $\phi_{IITM}$  are now instead handled by sessions of  $\pi_{IITM}$ , which is just as in  $\rho_{UC}^{\phi \to \pi}$ . In other words, reconnecting this tape has the same effect as adding the UC composition shell code, which internally changes the code  $c_{\phi}$  to instead be  $c_{\pi}$  for such inputs/outputs. So, unlike in Corollary 1, when we use the IITM composition theorem we actually do not need to include this shell code in  $\rho_{IITM}^{\phi \to \pi}$ . The IITM composition theorem then implies the following:

**Corollary 2** (Composition from the IITM theorem). Let  $\pi_{UC}$ ,  $\phi_{UC}$ ,  $\rho_{UC}$  be UC protocols such that  $\pi_{UC} \leq \xi_{UC} \phi_{UC}$  and  $\rho_{UC}$  meets the requirements of the UC composition theorem. Let  $\rho_{IITM}$  and  $\rho_{IITM}^{\phi \to \pi}$  be the IITM protocols from above. Then immediately by the IITM composition theorem,  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$ .

We provide full details, including the formal definitions of  $\rho_{IITM}$ ,  $\rho_{IITM}^{\phi \to \pi}$  and the proof of Corollary 2 in Appendix F. In the process of showing this result, we also found and fixed an issue that formally invalidates the UC theorem, namely, additional assumptions on non-forced writes used within  $\rho$  are actually necessary (cf. Appendix F for details). Altogether, the construction shown in Figure 4 and Corollary 2 illustrate how the additional requirements of the UC theorem from Diff. 13 and Diff. 14 are reflected in the mapped IITM protocols when the IITM theorem is used to obtain the same composition result.

Novel Composition Operation. Recall that the UC theorem applied to a protocol  $\rho$  replaces *all* sessions of subroutines running code  $\phi$  with sessions

running code  $\pi$ . Similarly, the IITM theorem applied to a protocol  $\rho$  replaces *all* sessions of a set of machines  $\phi$  with sessions of a set of machines  $\pi$ . Observe that we can use the above modeling technique not just to move *all* sessions of  $\phi$  to a new set of machines. Under certain conditions, we can rather more generally move a proper subset of the sessions of  $\phi$  to a new set of machines, say  $\phi'$ , while moving the other sessions to a different set, say  $\phi''$ , where  $\phi'$  and  $\phi''$  still run the same code  $c_{\phi}$ . We then obtain a simple corollary of the UC and IITM composition theorems (also for similar models), where we can replace  $\phi'$  with a realization  $\pi'$  but replace  $\phi''$  with a different realization  $\pi''$ . In other words, our technique allows for *replacing subsets of sessions*. This can be useful, e.g., if  $\phi$  is an ideal signature functionality, where each session models one key pair. Then we might want to implement certain keys with a signature scheme  $\pi'$  but others with a different signature scheme  $\pi''$  but others with a higher-level protocol  $\rho$ . We explain this in more detail, including requirements on  $\rho$ , in Appendix G.

#### 3.5 Capturing Dynamically Generated Machine Code

We now explain how our constructions from the previous sections can be extended to also support an unbounded number of dynamically generated machine codes. This bridges Diff. 1 and thus completes our mapping.

We start by observing that the UC model can be interpreted to be defined on a single universal Turing machine which is instantiated arbitrarily often during a run. Whenever a new instance receives its first input message, which contains the extended identity (pid, sid, c) of that instance, it stores this identity and from then on runs the code c given in its identity. This mechanism, whichs allows the UC model to seamlessly support arbitrary dynamically generated machine codes, can be transferred to an IITM protocol as follows.

Whenever a protocol  $\pi_{IITM}$  requires an unbounded number of different dynamically generated codes, potentially in addition to a finite number of static machine codes **Codes** as above, then we first map the fixed number of static codes of  $\pi_{IITM}$  as described in Section 3.2. We then add a universal Turing machine  $M_{\rm UT}$  that all other machines  $M_{c_i}$  connect to via pairs of I/O tapes. Each instance of  $M_{\rm UT}$  is identified by an ID (pid, sid, c) (instead of (pid, sid) as for machines  $M_{c_i}$  with fixed code  $c_i$ ), where  $c \notin \mathsf{Codes}$ , and internally runs code c specified by its ID. Whenever an instance of any machine in  $\pi_{IITM}$  wants to send a message to an instance with ID (*pid*, *sid*) and code  $c \notin Codes$ , i.e., where  $M_c$  does not exist in  $\pi_{IITM}$ , then it sends the message to the instance (*pid*, *sid*, *c*) of  $M_{UT}$ instead (this is easily done by choosing the appropriate tape; the actual message format, including the headers, does not change). The resulting protocol  $\pi_{IITM}$ behaves just as  $\pi_{UC}$  with dynamically generated codes. Hence, by the same reasoning as for Theorem 4, all realization results carry over for this construction, including results obtained via the UC composition theorem (i.e., Corollary 1). In Appendix  $\mathbf{F}$  we argue that also Corollary 2 carries over since the same modeling technique from Section 3.4 still applies independently of whether or not there is a universal Turing machine.

29

This bridges Diff. 1 by showing that the IITM model with its composition theorem also fully supports protocols with an unbounded number of dynamically generated machine codes, including all results available in the UC model.

#### 3.6 Discussion: Beyond UC Protocols

Above, we have considered only IITM protocols that are obtained by mapping some UC protocols. Of course, once we have mapped a UC protocol  $\phi_{UC}$ , including any security and composability results, into the IITM model, we are no longer limited to only considering combinations of  $\phi_{IITM}$  with such mapped protocols. We can rather consider any combination of  $\phi_{IITM}$  with arbitrary other IITM protocols. This includes cases where a higher-level IITM protocol  $\mathcal{P}$ is designed based on top of  $\phi_{IITM}$ , which can then, by the IITM composition theorem, be composed with any existing UC realization  $\pi_{IITM}$  of  $\phi_{IITM}$ . One can also consider novel realizations of  $\phi_{IITM}$  via an IITM protocol  $\mathcal{P}$ .

Such IITM protocols, which are combined with the mapped UC protocols, can then make full use of the features of the IITM model, including seamless support for joint state, global state, arbitrarily shared state, protocols without pre-established SIDs, and arbitrary combinations thereof. For example, a higherlevel IITM protocol  $\mathcal{P}$  can be defined in such a way that different sessions of  $\mathcal{P}$ share the same instance of  $\phi_{IITM}$  and  $\mathcal{P}$  could also work without pre-established SIDs etc. We refer the reader to [4, 17, 19, 20] for in-depth overviews, including examples, of IITM protocols with these features which can now be combined with existing UC results. Our mapping thus opens entirely new options for protocol designers so far working in the UC model by allowing them to combine their UC results with these IITM features, including IITM protocols that would require extensions of or are not yet supported by the UC model.

## 4 Impossibility of Embedding the IITM Model into the UC Model

Having mostly focused on the direction from UC to IITM, we now briefly discuss the other direction. In [20], it has been shown that the IITM runtime notion permits IITM protocols which cannot be expressed in the UC model as they do not meet the UC runtime notion. This includes protocols often encountered in practice, such as protocols that have to deal with ill-formed network messages. Combined with our results, this shows that the class of IITM protocols is strictly larger than the class of UC protocols. Another difference in protocol classes is due to so-called directory machines as required by the UC model for composition. These directory machines provide an oracle to the adversary to test whether a certain extended ID exists and is part of a specific UC protocol session. IITM protocols need not provide such a side channel, i.e., they are able to keep the IDs of internal subroutines secret from the adversary. This is not merely a cosmetic difference. Such an oracle rather changes security properties and might not be simulatable when (the existence of) extended IDs depend on some information

that is supposed to remain secret. Finally, in this paper we provide an impossibility result which shows that also the class of IITM adversaries and hence simulators is strictly larger than the class of UC adversaries/simulators (cf. Lemma 3 and Appendix C).

So at best one can hope for an embedding of the IITM model into the UC model for a restricted class of IITM protocols that follow the UC runtime notion and provide the same side channel as the directory machine. Realization relations carry over only for simulators that meet the UC runtime notion. Another obstacle to an embedding are IITM protocols that share state between protocol sessions, which includes joint state realizations as a special case. This is because the UC model mandates that UC protocols are subroutine respecting, i.e., have disjoint sessions that do not interact with each other. It might be possible to overcome this mismatch by using an idea briefly mentioned in [4], namely, modeling all sessions of an IITM protocol within a *single* session of a UC protocol by introducing a second layer of "internal" SIDs identifying the actual sessions of the protocol. The downside of this idea is that a protocol mapped in such a way cannot be composed with UC protocols modeled in the "standard" way. This is because all sessions of such UC protocols would have to access the same UC session (but potentially different "internal" sessions) of the mapped protocol. Resolving this mismatch might be possible but probably requires a new variant or perhaps a corollary of the UC composition theorem. We leave exploring the details of this direction for future work.

## 5 Conclusion

In this work, we have initiated a line of research that investigates the so far unexplored relationships of models for universal composability. Towards this goal, we have answered the question whether and in how far protocols, security, and composability results from the UC model carry over to the IITM model. Our main finding is that despite the many conceptually differences in the models, *all* of this actually carries over in a natural way. As an immediate practical benefit, this allows protocol designers coming from the UC model to compose their results with and leverage the features of the IITM model, such as seamless support for protocols with joint, global, shared state, protocols without pre-established SIDs, including protocols that are not supported by the UC model, as well as combinations of all of this.

Through the embedding we developed a modeling technique that allows for obtaining a more general type of composition from existing composition theorems. Secondly, we also identified and fixed several issues that, among others, invalidate the UC composition theorem. These fixes should be compatible with and hence retroactively apply to existing UC protocols in the literature. These points of independent interest further highlight the new insights a formal embedding brings to the table.

While in this work we have mostly focused on the embedding of the UC model into the IITM model, we have also discussed and proved obstacles for the other direction, and hence, set the boundaries for such an embedding, leaving a formal embedding of (a subset of) the IITM model into the UC model to future work.

Acknowledgements. We thank Ran Canetti and Björn Tackmann for helpful discussions on an early draft of this paper.

## References

- Backes, M., Dürmuth, M., Hofheinz, D., Küsters, R.: Conditional Reactive Simulatability. International Journal of Information Security (IJIS) 7(2), 155–169 (4 2008)
- Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 1–30. Springer, Heidelberg (Nov 2020).
- Camenisch, J., Enderlein, R.R., Krenn, S., Küsters, R., Rausch, D.: Universal composition with responsive environments. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 807–840. Springer, Heidelberg (Dec 2016).
- Camenisch, J., Krenn, S., Küsters, R., Rausch, D.: iUC: Flexible universal composability made simple. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 191–221. Springer, Heidelberg (Dec 2019).
- Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. Tech. Rep. 2000/067, ePrint (2000), available at http://eprint.iacr. org/2000/067, new versions: Dec 2005, Jul 2013, Dec 2018, Aug 2019, Feb 2020
- Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001. pp. 136–145. IEEE Computer Society Press (Oct 2001).
- 7. Canetti, R.: Universally Composable Security. J. ACM 67(5), 28:1–28:94 (2020)
- Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 3–22. Springer, Heidelberg (Aug 2015).
- Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (Feb 2007).
- Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (Aug 2003).
- Graf, M., Rausch, D., Ronge, V., Egger, C., Küsters, R., Schröder, D.: A Security Framework for Distributed Ledgers. In: ACM CCS 2021, Nov 14–19, 2021, Seoul. ACM (2021)
- Hofheinz, D., Shoup, V.: GNUC: A new universal composability framework. Journal of Cryptology 28(3), 423–508 (Jul 2015).
- Hofheinz, D., Unruh, D., Müller-Quade, J.: Polynomial runtime and composability. Journal of Cryptology 26(3), 375–441 (Jul 2013).
- Küsters, R.: Simulation-Based Security with Inexhaustible Interactive Turing Machines. In: Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006). pp. 309–320. IEEE Computer Society (2006), see [20] for a full and revised version.

- 32 Daniel Rausch, Ralf Küsters, and Céline Chevalier
- Küsters, R., Datta, A., Mitchell, J.C., Ramanathan, A.: On the relationships between notions of simulation-based security. Journal of Cryptology 21(4), 492–546 (Oct 2008).
- Küsters, R., Rausch, D.: A framework for universally composable Diffie-Hellman key exchange. In: 2017 IEEE Symposium on Security and Privacy. pp. 881–900. IEEE Computer Society Press (May 2017).
- Küsters, R., Tuengerthal, M.: Composition theorems without pre-established session identifiers. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011. pp. 41–50. ACM Press (Oct 2011).
- Küsters, R., Tuengerthal, M.: Ideal key derivation and encryption in simulationbased security. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 161–179. Springer, Heidelberg (Feb 2011).
- Küsters, R., Tuengerthal, M., Rausch, D.: Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. J. Cryptol. 33(4), 1585–1658 (2020)
- Küsters, R., Tuengerthal, M., Rausch, D.: The IITM Model: a Simple and Expressive Model for Universal Composability. J. Cryptol. 33(4), 1461–1584 (2020)
- Maurer, U.: Constructive cryptography a primer (invited paper). In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, p. 1. Springer, Heidelberg (Jan 2010)
- Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: Gritzalis, D., Jajodia, S., Samarati, P. (eds.) ACM CCS 2000. pp. 245–254. ACM Press (Nov 2000).
- 23. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-Lock Puzzles and Timed-Release Crypto. Tech. Rep. MIT/LCS/TR-684, MIT (March 1996), available at http: //theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps

# Appendix

In the following appendices we first give full details of the results described in the main body of the paper:

- Appendix A: proof of Lemma 2
- Appendix B: proof of Theorem 4
- Appendix C: proof of Lemma 3
- Appendix D: proof of Theorem 5
- Appendix E: proof of Corollary 1
- Appendix F: proof of Corollary 2

Next, we describe in Appendix G our technique for obtaining a new type of composition as a corollary of both UC and IITM composition theorems.

In Appendix H, we describe an extended variant of our mapping. The most notable effect of this more complex variant is that we can prove Theorem 4 without any assumptions on the simulator.

In Appendix I, we recall the formal definition of the second composition theorem of the IITM model [20], including the corresponding security definition, which enable a single session security analysis of protocols with disjoint sessions. We then formally show that this theorem is indeed applicable in the last step of the proof of Theorem 4.

Finally, we provide in Appendix J more in depth discussion of certain technical details of the UC model [7]. Most notably, this appendix provides full descriptions of the technical issues that we found while constructing our mapping and which, among others, cause the UC composition theorem to fail. For each such issue, we also construct a counter example that contradicts the composition theorem. We finally give fixes for all of these issues which should be compatible with existing UC protocols.

## A Proof of Lemma 2

Let  $\mathcal{E}$  be an environment. Since  $\mathcal{E}$  is universally bounded, there exists a polynomial p such that  $p(\eta + |a|)$  bounds the runtime of  $\mathcal{E}$ , where  $\eta$  is the security parameter and a the external input. Observe that p is an upper bound for the total number of bits sent from the environment to the protocol (via an I/O or network tape) and thus, by unary encoding of import, also for the total import sent to the protocol. Next, recall that the UC model shows (e.g., as part of the proof of the composition theorem) that multiple instances of a ppt protocol  $\pi_{UC}$  can be simulated within a single ppt instance of a machine. Since  $\pi_{IITM}$  behaves just as  $\pi_{UC}$ , we thus have that there exists a single polynomial q (in the total import

received by the protocol) that bounds the combined runtime of emulating all codes  $c_i$  in all instances of  $M_{c_i}$ . Combining both polynomials gives the runtime bound  $q(p(\eta + |a|))$ . This bound, however, only applies to the total runtime of all the codes  $c_i$ . We still need to upper bound the runtime of the surrounding logic and the additional machines that were introduced in  $\pi_{IITM}$  to run the codes  $c_i$  properly.

Observe that new instances of some machine  $M_{c_i}$  are only ever created by messages from the environment or by messages sent by some code  $c_j$  within a machine  $M_{c_j}$ . Hence,  $q(p(\eta + |a|))$  also upper bounds the combined number of instances of all machines  $M_{c_i}$ . Regarding  $M_{bc}$  and  $M_{msg}$ , there is at most one instance per session, where the number of sessions is bounded by the number of instances of  $M_{c_i}$ . Hence,  $q(p(\eta + |a|))$  also upper bounds each the number of instances of  $M_{bc}$  and instances of  $M_{msg}$ .

Furthermore,  $q(p(\eta + |a|)) + c$  for some constant c upper bounds the length of each individual message received by any machine instance. For messages sent by the environment or the code  $c_i$  in a machine  $M_{c_i}$ , we already know from above that they are bounded by  $q(p(\eta + |a|))$  plus potentially a constant c' for messages sent to  $M_{\rm msg}$ , which accounts for the additional strings nonForcedWrite, existsInstance?, and the code of the machine  $M_{\rm msg}$ . For messages sent by  $M_{\rm bc}$ , this follows from the fact that the very same message was previously generated by some code  $c_i$ . Similarly for nonForcedWrite messages sent by  $M_{msg}$ . For messages to register instances at  $M_{\rm msg}$  respectively the responses to these messages, these messages add at most a fixed constant c'' (accounting for the string register) to the length of the original input of the instance of  $M_{c_i}$ , which we know to be bounded by  $q(p(\eta + |a|))$ . Similarly for notifyImport messages and their response. Finally, there are the responses to totalImport? messages, which are also bounded by  $q(p(\eta + |a|)) + c'''$  for some constant c''' (note in particular that the total amount of import i contained in that message is upper bounded by  $p(\eta + |a|)$  since that is the maximal amount of import that an environment can provide).

Finally, observe that the runtime of an instance of a machine  $M_{c_i}$ , excluding the runtime needed for running code  $c_i$ , is linear in the input length, which is at most  $q(p(\eta + |a|))$ . Hence, the total runtime of each individual instance of  $M_{c_i}$  is bounded by a polynomial. For  $M_{bc}$ , we also directly see that the runtime of each instance is linear in the current message length  $\leq q(p(\eta + |a|))$ . For  $M_{msg}$ , each instance runs polynomial in the combined length of all inputs so far (since **register** inputs extend the internal state by at most its own length, the size of the internal state determines how often a predicate P is run, and the predicates P used in the UC model run in fixed linear time in their input<sup>18</sup>),

<sup>&</sup>lt;sup>18</sup> We discuss this property of predicates in detail in Appendix J.2. We note that the proof would still hold if one were to consider a more general class of predicates as long as the runtime of all predicates is still bounded by the same polynomial. Conversely, if the runtime of predicates were not restricted at all, then not only would this proof fail but it would also not be possible to show the UC composition theorem (cf. Appendix J.2).

which is at most  $(q(p(\eta + |a|)) + c)^2$ . Altogether, we have that there are at most polynomially many machine instances each of which is bounded by one of finitely many polynomials in  $\eta$  and |a|. Hence, the combined runtime of all instances is bounded by a polynomial in  $\eta$  and |a|, which gives the claim.

## B Proof of Theorem 4

As per Footnote 12, we do not show  $\leq_{IITM}$  as defined in Definition 2 but rather show the equivalent notion that considers a dummy adversary  $\mathcal{A}_{Dum,IITM}$  in the real world as that notion is closer to the definition of  $\leq_{UC}$ .

Let  $\mathcal{A}_{Dum, UC}$  be the UC dummy adversary with runtime bounded by a polynomial in its import. Specifically, recall that the dummy is defined to use one import for every bit that he forwards and stop once he has less than  $\eta$ import left. Recall that the dummy adversary  $\mathcal{A}_{Dum, UC}$  provides an operation op(i) that allows the environment to send import *i* to  $\mathcal{A}_{Dum, UC}$ , who internally adds *i* to the total import received so far and then returns control back to the environment. Let  $\mathcal{S}_{UC}$  be the corresponding simulator for the ideal world such that the protocols  $\pi_{UC}$  and  $\phi_{UC}$  are indistinguishable for any  $\xi$ -identity-bounded environment. As mentioned, we here show the theorem using the assumption that  $\mathcal{S}_{UC}$  implements op(i) without sending any messages to the ideal protocol before returning control to the environment, which is a natural property that should be trivially met by simulators for reasonable protocol definitions including existing simulators proposed in the literature. See Appendix H for the general case, which requires a somewhat more involved mapping.

On an intuitive level, the following proof proceeds in four major game hops, which iteratively modify both real and ideal worlds (see Figures 5 and 6 for an illustration): (1) Starting from the UC model, we switch into a special case of the IITM model where we consider UC runtime bounded adversaries and environments that are  $\xi$ -identity bounded while using only a single protocol session. The main challenge here is to ensure that IITM environments still respect the balanced property required for UC environments, which is necessary for a successful reduction. We do so by adding a wrapper to each adversary that manually generates additional import such that the internal adversary always has at least the same import as the protocol. (2) Next, we consider IITM environments that are not necessarily  $\xi$ -identity bounded (but still single session) and show that such an environment cannot distinguish protocols which manually enforce  $\xi$  themselves via the machine  $M_{\text{identity}}^{\xi}$ . (3) In the third step, we consider the general IITM dummy adversary instead of the specific dummy that additionally adheres to the UC runtime notion. Importantly, the general IITM dummy never stops delivering messages. The key idea of this step is to add a wrapper to the simulator which generates additional import such that, if the UC bounded dummy adversary were to receive the same import, then he will also never stop delivering messages und thus behave just as the general IITM dummy. (4) Finally, we consider general IITM environments that can access multiple concurrent protocol sessions. Security of this setting follows directly from the second IITM



Fig. 5. Game hops for the proof of Theorem 4 (Part I). Left sides show the real worlds, right sides show the ideal worlds.
# $\Rightarrow$ third proof step by reduction (continuation of Figure 5)



Fig. 6. Game hops for the proof of Theorem 4 (Part II). Left sides show the real worlds, right sides show the ideal worlds.

composition theorem for composing an unbounded number of concurrent (disjoint) protocol sessions. In what follows, we formalize each of these intuitive steps.

Indistinguishability for UC runtime bounded adversaries. We first show the following statement. If we consider the dummy adversary  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  in the IITM model that is also bounded by a fixed polynomial in its import, then we find a simulator  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  that is bounded by a fixed polynomial in its import such that no environment  $\mathcal{E}_{IITM}^{\text{single},\xi}$  that interacts with only a single protocol session and uses only sender identities permitted by  $\xi$  can distinguish real and ideal world. That is,  $\mathcal{E}_{IITM}^{\text{single},\xi} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM} \equiv \mathcal{E}_{IITM}^{\text{single},\xi} | \mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}$  for all such environments.

More formally, we define  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  to connect to all network tapes of machines of the real protocol  $\pi_{IITM}$ , which we will call internal network tapes in what follows.  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  offers two external network tapes connected to the environment. The first external network tape serves as the "main" connection to the environment where  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  receives and processes all messages from the environment intended for instances of any machine  $M_c \in \pi_{IITM}$ . That is, this main external tape directly corresponds to and implements the connection between UC environment and UC adversary in the UC model. The second external tape corresponds to the internal network tape of the machine  $M_{\rm msg}$  added by our mapping and allows the environment to issue totalImport? messages to that machine.  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  then internally simulates  $\mathcal{A}_{Dum,UC}$  but with the following additional wrapper. Each time  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  receives an incoming network message, it first sends a totalImport? request to the instance of  $M_{\rm msg}$  in the challenge session  $sid_c$ . If the total protocol import contained in the response is larger than the total import provided to  $\mathcal{A}_{Dum, UC}$  so far, then the difference is manually provided by  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  to the internal import budged of the simulated  $\mathcal{A}_{Dum,UC}$ via calls to op (if the protocol has received several new imports that exceed the import of the adversary, then op is called once for each of them). Afterwards, i.e., once the simulated  $\mathcal{A}_{Dum,UC}$  has processed *op* and wants to return control to the environment,  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  instead proceeds by processing the original incoming message: if the incoming message is a totalImport? request received on the second external tape corresponding to  $M_{\rm msg}$ , then the wrapper forwards this message to  $M_{\rm msg}$  and returns the response without invoking  $\mathcal{A}_{Dum,UC}$  (and hence without using the import budget of the simulated  $\mathcal{A}_{Dum,UC}$  for these messages). For all other messages received on the main external tape or any internal network tape of some machine  $M_c$ ,  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  continues just as  $\mathcal{A}_{Dum,UC}$  and thus adheres to the UC runtime notion.<sup>19</sup> Observe that the wrapper transfers the balanced requirement for environments to the IITM model: it is impossible for

<sup>&</sup>lt;sup>19</sup> Note that simulating  $\mathcal{A}_{Dum,UC}$  is indeed possible. In particular, whenever  $\mathcal{A}_{Dum,UC}$  wants to send a message to some protocol machine instance with extended ID (pid, sid, c), where c is the machine code of the recipient, then  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  can forward that message on the internal network tape  $t_{M_c}$  connected to the uniquely identified machine  $M_c$  running code c in  $\pi_{IITM}$ . If no protocol machine with code c and hence no such network tape exists, then the message is simply dropped. This is just as in the UC model, where the message would formally be sent but, since no

the IITM dummy adversary to have less import than the protocol, just as is the case in the UC model. We define the simulator  $S_{IITM}^{\text{UC-bounded}}$  in the same way. That is,  $S_{IITM}^{\text{UC-bounded}}$  internally simulates  $S_{UC}$  within the same wrapper, which adds additional import if the protocol has received more import than  $S_{UC}$  so far while directly forwarding totalImport? requests without involving  $S_{UC}$ . Observe that these adversaries meet the IITM runtime notion since  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$ and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}$  run in polynomial time in the combined length of all inputs received by the environment. This follows by analogous reasoning as for Lemma 2 and using the observation that the additional import that might be generated by the wrapper is upper bounded by the overall import provided by the environment to the protocol.

Let  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$  be an environment in the IITM model that sends inputs and network messages (via the dummy adversary) only to a single session of the protocol  $\pi_{IITM}/\phi_{IITM}$ . That is,  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$  outputs only inputs and network messages for the same challenge session  $sid_c$ , which the environment is free to choose in its first input/network message (cf. Appendix I for a formal definition of single session environments in the IITM model). We also assume that  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$  specifies only sender identities for inputs that are permitted by  $\xi$ . We show that such an environment cannot distinguish  $\mathcal{A}_{Dum,IITM}^{\operatorname{UC-bounded}} | \pi_{IITM}$  and  $\mathcal{S}_{IITM}^{\operatorname{UC-bounded}} | \phi_{IITM}$  by reducing this to the UC case.

We define  $\mathcal{E}_{UC}$  to be the environment in the UC model that runs the same logic as  $\mathcal{E}_{IITM}^{single,\xi}$  but stops the run with empty overall output upon reaching a fixed runtime bound q, where q is a polynomial in the number of import tokens currently held by  $\mathcal{E}_{UC}$ . Every time  $\mathcal{E}_{IITM}^{single,\xi}$  wants to provide import to the protocol such that the total import received by the protocol would be larger than what the adversary has received so far,  $\mathcal{E}_{UC}$  first provides the difference to the adversary via a call to op and then, after being activated again, sends the input of  $\mathcal{E}_{IITM}^{single,\xi}$  to the protocol. If  $\mathcal{E}_{IITM}^{single,\xi}$  wants to use the second external network tape of the IITM adversary to send totalImport? to  $M_{msg}$ , then  $\mathcal{E}_{UC}$  does not actually forward that request but rather continues to behave as if  $\mathcal{E}_{IITM}^{single,\xi}$  receives a response containing the amount of import sent as input to the protocol so far. Network messages sent by  $\mathcal{E}_{IITM}^{single,\xi}$  via the main tape of the IITM adversary are forwarded by  $\mathcal{E}_{UC}$  to its UC adversary. Responses by the UC adversary are processed as if they were received by  $\mathcal{E}_{IITM}^{single,\xi}$  on the main network tape of the IITM adversary. Observe that  $\mathcal{E}_{UC}$  is indeed a ( $\xi$ -identity-bounded) environment in the UC model as it is ppt in the UC sense by construction and as it is balanced due to providing the adversary with at least the same import as the protocol.

recipient instance with that code is ever created by  $\pi$ , the message is then dropped by definition of the computational model. Further note that this simulation is possible even if the protocol  $\pi_{IITM}$  is extended to not just handle a fixed number of codes  $c \in \mathsf{Codes}$  but also includes a universal Turing machine  $M_{\mathrm{UT}}$  that handles arbitrary dynamically generated codes  $c \notin \mathsf{Codes}$  as described in Section 3.5. In this case, all messages for a machine instance with code  $c \notin \mathsf{Codes}$  are forwarded on the internal network tape t of  $M_{\mathrm{UT}}$ .

Now compare the IITM real world  $\mathcal{E}_{IITM}^{\text{single},\xi} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$  with the UC real world where  $\mathcal{E}_{UC}$  runs with  $\mathcal{A}_{Dum,UC}$  and  $\pi_{UC}$ . Observe that until  $\mathcal{E}_{UC}$ reaches its runtime bound q it perfectly simulates not just  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$  but also the behavior of the wrapper of  $\mathcal{A}_{Dum,IITM}^{\mathrm{UC-bounded}}$ . In particular, if the wrapper  $\mathcal{A}_{Dum,IITM}^{\mathrm{UC-bounded}}$  in the UTM actions generated as a distinguished of the second in the IITM setting generates and provides additional import to the internally simulated  $\mathcal{A}_{Dum,UC}$  beyond the import provided by  $\mathcal{E}_{IITM}^{\text{single},\xi}$ , then  $\mathcal{E}_{UC}$  in the UC setting adds the same amount of import via the same calls to op. Note that the exact point in the run where import is provided via a call to op is slightly different in both worlds but still results in the same behavior. Specifically, in the IITM setting  $\mathcal{E}_{IITM}^{\text{single},\xi}$  first provides some import to the protocol (that raises the total over the import of the adversary), which then performs some computation. At some point the  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  is activated with a message m, who then first runs op and processes m afterwards. In the UC world,  $\mathcal{E}_{UC}$  first calls op, then provides the import to the protocol, which performs some computations, and then  $\mathcal{A}_{Dum, UC}$  is activated with and directly processes the message m. So in the IITM setting the protocol is activated before a call to op and in the UC setting the protocol is activated after a call to op. But since the dummy adversary processes op without any interaction with the protocol and therefore this step does not depend on the state of the protocol, the resulting behavior is still the same. In particular, we have that as soon as the message m is processed by (the internally simulated)  $\mathcal{A}_{Dum, UC}$ , which might require some interaction with the real protocol, the state of the real protocol is identical in both worlds. Altogether we have that the real worlds behave identical with the same overall outputs, if any, until  $\mathcal{E}_{UC}$  reaches the runtime bound q.

Now compare the IITM ideal world  $\mathcal{E}_{IITM}^{\text{single},\xi} | \mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}$  with the UC ideal world where  $\mathcal{E}_{UC}$  runs with  $\mathcal{S}_{UC}$  and  $\phi_{UC}$ . Since the simulator  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  uses the same wrapper as  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  and, by assumption, implements *op* without interacting with the protocol, the same reasoning as above also applies here. Hence we have that the ideal worlds behave identical with the same overall outputs, if any, until  $\mathcal{E}_{UC}$  reaches the runtime bound q.

The only thing left is to define the runtime bound q in a suitable way that allows for mapping all runs in the IITM setting to runs in the UC setting with the same overall output distribution. Let p be the polynomial in the security parameter  $\eta$  and length of the external input |a| that bounds the runtime of  $\mathcal{E}_{IITM}^{single,\xi}$  by the universally bounded property of IITM environments. Observe that the additional runtime needed by  $\mathcal{E}_{UC}$  for simulating responses to totalImport? requests and providing additional import can be bounded by a polynomial r in the total length of all outputs generated by  $\mathcal{E}_{IITM}^{single,\xi}$ , which is at most  $p(\eta + |a|)$ . We set  $q(\cdot) := p(\cdot) + r(p(\cdot))$  w.l.o.g. monotonically increasing. Now, consider a run in the IITM setting with security parameter  $\eta$  and external input a, where |a| is bounded by a polynomial in the security parameter (the security definition respectively the definition of negligible functions considers only external inputs of bounded length). Consider the corresponding run with the same randomness in the UC setting but with security parameter  $\eta$  and external input  $(a, 1^{\eta+2\cdot p(\eta+|a|)})$ . Note that the overall length of this input and hence the import provided to the environment is polynomial in  $\eta$ . Observe that, given this input, the runtime bound q is never exceeded: Since the overall runtime of  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$  is bounded by  $p(\eta + |a|)$  and import was encoded in unary in the IITM setting, we have that  $\mathcal{E}_{UC}$  forwards at most  $2 \cdot p(\eta + |a|)$  of its import, leaving it with at least  $\eta + |a|$ . Hence, the overall runtime bound of  $\mathcal{E}_{UC}^{\operatorname{single},\xi}$  is always at least  $q(\eta + |a|) = p(\eta + |a|) + r(p(\eta + |a|))$ , which is never reached by definition of  $\mathcal{E}_{IITM}^{\operatorname{single},\xi}$ . Thus, the distinguishing advantage in the IITM model is the same as in the mapped runs in the UC model, which is negligible by assumption.

Indistinguishability for arbitrary single session environments. We know that the systems  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi |_{IITM}$  are indistinguishable for single session  $\xi$ -identity bounded environments  $\mathcal{E}_{IITM}^{\text{single},\xi}$ . This is equivalent to  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi\text{-id}}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi\text{-id}}$  being indistinguishable for all single session environments  $\mathcal{E}_{IITM}^{\text{single},\xi}$ . In particular, a single session environment running with  $\mathcal{M}_{\text{identity}}^{\xi}$  constitutes a  $\xi$ -identity bounded single session environment.

**Indistinguishability for arbitrary adversaries.** So far, we have only constructed a simulator for the dummy adversary that has a fixed runtime bound in its import and which stops once the runtime/import granted by the environment is exhausted. However, the IITM model also quantifies over adversaries whose runtime is not externally determined by the environment. In particular, the dummy adversary  $\mathcal{A}_{Dum,IITM}$  in the IITM model does not expect adversarial import in its network messages, does not provide the operation op for adding import, and also does not adhere to a runtime bound determined by externally provided import. The IITM dummy rather always forwards whatever network messages it receives and hence cannot be stopped by the environment.<sup>20</sup> Observe that the simulator from above does not directly work for  $\mathcal{A}_{Dum,IITM}$  since he also simulates the case where the UC dummy adversary stops delivering messages due to reaching its runtime budget. We slightly modify the simulator via a wrapper to prevent this case from occurring, which then gives the desired result. The main idea to achieve this is to let the wrapper itself generate a sufficiently large amount of import via calls to op such that we can argue that, if one were to run  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  with that same import in the real world, then  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  never reaches its runtime bound.

There is also a minor syntactical difference between the adversaries  $\mathcal{A}_{Dum,IITM}$ and  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ . Namely, while  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  offers just one main external network tape to the environment for messages to all machines  $M_c \in \pi_{IITM}^{\xi\text{-id}}$  (plus a second external network tape exclusively for network messages to  $M_{\text{msg}}$ ) and then internally directs messages to the correct internal network tapes connected to each individual machine  $M_c \in \pi_{IITM}^{\xi\text{-id}}$  (which can be identified from the code c of the recipient),  $\mathcal{A}_{Dum,IITM}$  rather offers one external network tape corresponding

<sup>&</sup>lt;sup>20</sup> Note that this dummy  $\mathcal{A}_{Dum,IITM}$  running with an IITM protocol still runs in polynomial time in the length of all inputs provided by the environment and hence meets the IITM runtime notion.

to each of its internal network tapes and then forwards all messages between matching external and internal tapes. This mismatch is essentially just a cosmetic difference. In the simulation, we can simply bundle messages from all external tapes of machines  $M_c \in \pi_{IITM}$  on the single main tape of the previous simulation.

Before we formally define the simulator, we first fix a bit of terminology to simplify the following presentation. As explained above, the main idea of the simulation will be to generate additional import i via calls to op(x) in such a way that, if the same calls were received by  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ , then this dummy will have *i* additional runtime remaining after processing op(x). This additional runtime i will then be chosen in such a way that it is sufficient for forwarding all messages that might have to be forwarded by the dummy. However, processing op(x) already uses  $log_2(x) + c$  runtime of the runtime budget of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ (specifically the runtime budget of  $\mathcal{A}_{Dum,UC}$  simulated within), where c is a small constant accounting for the message format of op and the logarithm is because the import x contained in the op(x) message is interpreted in a binary encoding by the simulated  $\mathcal{A}_{Dum, UC}$ .<sup>21</sup> Hence, we cannot simply call op(i) as this provides exactly i import, which was sufficient to guarantee the balanced requirement in the first step of the proof, but then leaves the dummy with less than i import remaining after processing the operation. Instead, to ensure that iimport remains, we rather provide double that import by calling  $op(2 \cdot \max(i, c'))$ where c' is a fixed constant such that  $c' - log_2(2 \cdot c') - c \ge 0$ . Doubling the import ensures that, even after subtracting runtime cost  $log_2(x) + c$  for running op(x), the remaining import added by this operation is still at least as large as i. The value c' covers cases where i is too small such that doubling might not be sufficient to retain i import. Note that such c' indeed exists and every value  $i \geq c'$  then also has the desired property. In order to simplify the following presentation and highlight the core ideas more clearly, instead of always writing out this formula explicitly we will rather say "provide/add/generate i import via op" or, if clear from the context, just "provide/add/generate i import".

Given the above intuition and terminology, we now define  $S_{IITM}$  to offer the same set of external network tapes connected to the environment as  $\mathcal{A}_{Dum,IITM}$ , i.e., one external network tape corresponding to each internal network tape of each machine in  $\pi_{IITM}^{\xi\text{-id}}$ .  $S_{IITM}$  internally simulates  $S_{IITM}^{\text{UC-bounded}}$  but with the following additions.<sup>22</sup> Let p be the polynomial in the import currently held by the

<sup>&</sup>lt;sup>21</sup> As part of the simulation, the wrapper of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  implicitly maps the import fields in the header of messages between unary encoding used by protocol/environment and binary encoding expected by the simulated internal adversary  $\mathcal{A}_{Dum,UC}$ .

<sup>&</sup>lt;sup>22</sup> As mentioned above, on a technical level during this simulation  $S_{IITM}$  also maps the larger set of its own external network tapes to the two external network tapes of  $S_{IITM}^{\text{UC-bounded}}$ . Specifically,  $S_{IITM}$  forwards messages from the environment received on the external tape corresponding to  $M_{\text{msg}}$  to the second external tape of  $S_{IITM}^{\text{UC-bounded}}$ , i.e., the tape that also corresponds to  $M_{\text{msg}}$ , and vice versa. Messages received on all other external network tapes, each of which corresponds to some machine  $M_c \in \pi_{IITM}$ , are sent via the single main external network tape of  $S_{IITM}^{\text{UC-bounded}}$ , i.e., the tape where the internal simulator expects all incoming network messages for protocol machines  $M_c$ . Conversely, when  $S_{IITM}^{\text{UC-bounded}}$  wants to output a message in the name of some

real protocol that upper bounds the total runtime of the codes of  $\pi_{UC}$  and hence the length of all network messages sent by the real protocol  $\pi_{IITM}^{\xi\text{-id}}$  (except for responses to totalImport? requests, which do not exist in  $\pi_{UC}$  and thus also do not count towards the runtime budget of the dummy). This polynomial exists as argued as part of the proof of the UC composition theorem (cf. also the proof of Lemma 2). W.l.o.g., choose  $p(x) \ge x$  monotonically increasing. Now, upon its first activation  $S_{IITM}$  adds  $\eta$  additional import to the internally simulated  $S_{IITM}^{\text{UC-bounded}}$ via a call of *op*. Further, each time  $S_{IITM}$  is activated it keeps track of the sum of all imports received by the real protocol (by sending totalImport? requests to  $M_{\rm msg}$  in the ideal protocol as well as tracking the overall import forwarded to the protocol as part of network messages m from the environment. This includes the current network message received from the environment by  $S_{IITM}$ , if any). If the total amount of import sent to the real protocol in the last activation of the simulator was i and now the protocol has received some new import i', then  $S_{IITM}$  provides additional import p(i + i') - p(i) to  $S_{IITM}^{\text{UC-bounded}}$  (if the protocol has received multiple new imports i' since the last activation of the simulator, then this step is repeated for each of them separately, with the import from the current network message, if any, processed last). Then, if the current incoming message m is a network message received from the environment,  $S_{IITM}$  adds an additional  $2 \cdot |m|$  import via  $op.^{23}$  Afterwards,  $S_{IITM}$  lets  $S_{IITM}^{UC-bounded}$  process the incoming message. During all of this,  $S_{IITM}$  always shifts the "balanced environment" check of the inner wrapper contained within  $S_{IITM}^{UC-bounded}$  (see first step of this proof), including the message totalImport? sent to the ideal protocol. Thus, the simulated inner wrapper does not generate any additional calls to opitself. Note that since we have  $p(x) \geq x$  the outer wrapper of  $\mathcal{S}_{IITM}$  already generates sufficient import to guarantee the balanced requirement, which is why we can skip the simulation of the balanced check of the inner wrapper and still be able to perform a reduction to the previous case. Further observe that  $S_{IITM}$ still meets the runtime notion of the IITM model by the same reasoning as for  $\mathcal{S}_{UTM}^{\mathrm{UC-bounded}}$ . In particular, the additional import generated by the wrapper of

instance of a machine  $M_c \in \pi_{IITM}$  via its main external network tape, then  $S_{IITM}$  forwards this on its own external tape corresponding to  $M_c$ . This tape mapping also includes a straightforward conversion between the UC and IITM dummy formats for messages to/from  $M_c$ . In particular, the (unary) import in the header of an incoming network message from the environment for the IITM dummy does not contain import for the adversary but rather contains import *i* directly provided to the protocol (since the message is copied as is to the protocol by the IITM dummy). This can easily be converted to a message for the UC dummy by adding an empty adversarial import header and re-encoding the import *i* in binary to be an instruction for the UC adversary to forward import *i* to the protocol.

<sup>&</sup>lt;sup>23</sup> Technically, after converting m from the IITM to the UC dummy format the resulting message might be longer by some small constant c due to the addition of an adversarial import field. In such a case, the simulator rather adds  $2 \cdot (|m| + c)$  import, which ensures that the UC dummy can process also such a slightly longer UC message format. We leave this detail implicit in what follows.

 $S_{IITM}$  is upper bounded by a polynomial in the length of all messages sent by the environment plus  $\eta$ .

Now let  $\mathcal{E}_{IITM}^{single}$  be a single session environment that tries to distinguish  $\mathcal{A}_{Dum,IITM} \mid \pi_{IITM}^{\xi \cdot id}$  and  $\mathcal{S}_{IITM} \mid \phi_{IITM}^{\xi \cdot id}$ . We reduce this to the previous case as follows. We define  $\mathcal{E}'_{IITM}^{single}$  to internally simulate  $\mathcal{E}_{IITM}^{single}$  as well as the wrapper portion of  $\mathcal{S}_{IITM}$  (minus the changes to the internal wrapper of  $\mathcal{S}_{IITM}^{UC-bounded}$ ). More specifically,  $\mathcal{E}'_{IITM}^{single}$  starts by providing  $\eta$  import to the adversary. During the run,  $\mathcal{E}'_{IITM}^{single}$  keeps track of the import received by the real protocol so far. Each time  $\mathcal{E}_{IITM}^{single}$  wants to provide more import to the real protocol (either directly via an input or as part of a forwarded network message),  $\mathcal{E}'_{IITM}^{single}$  first evaluates the polynomial p on that import and provides the difference as additional import via a call to op to the adversary. If  $\mathcal{E}_{IITM}^{single}$  wants to send a network message m to the adversary, after  $\mathcal{E}'_{IITM}^{single}$  has evaluated p and potentially provided additional import as a result,  $\mathcal{E}'_{IITM}^{single}$  provides a further  $2 \cdot |m|$  import to the adversary before finally sending the actual message m. Observe that the additional runtime necessary for simulating the wrapper, including sending more import, is polynomial in the runtime of  $\mathcal{E}_{IITM}^{single}$  plus  $\eta$ . Hence  $\mathcal{E}'_{IITM}^{single}$  is universally bounded and a valid single session environment in the IITM model.

Consider  $\mathcal{E}_{IITM}^{\text{single}} | \mathcal{S}_{IITM} | \phi_{IITM}^{\xi \text{-id}} \text{ and } \mathcal{E}_{IITM}^{\prime \text{ single}} | \mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi \text{-id}}$ . By construction tion, at each point in the run  $\mathcal{E}'_{IITM}^{\text{single}}$  has provided at least  $p(x) \ge x$  import to its simulator, where x is the total amount of import provided to the protocol so far, which includes all import directly provided via inputs to the protocol. Hence, whenever the inner wrapper of  $S_{IITM}^{\text{UC-bounded}}$  checks the balanced requirement, the requirement is meet and thus no additional runtime is generated by the wrapper. In other words, this is the same behavior as for  $\mathcal{E}_{IITM}^{\text{single}} | \mathcal{S}_{IITM} | \phi_{IITM}^{\xi \text{-id}}$ , where these balanced checks are skipped entirely. So analogous to the first step of this proof, the only formal difference between runs of  $\mathcal{E}_{IITM}^{\text{single}} | \mathcal{S}_{IITM} | \phi_{IITM}^{\xi \cdot \text{id}}$ and  $\mathcal{E}'_{IITM}^{\text{single}} | \mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi \cdot \text{id}}$  is therefore the timing of additional calls to op issued by the outer wrapper: In the former case, where the wrapper runs as part of  $S_{IITM}$ , the protocol might be activated with new import first before the call of *op*. In the latter case, where the wrapper is simulated within  $\mathcal{E}'_{IITM}^{\text{single}}$ , the protocol might be activated with new import only after the call to op. Here we use again that the inner simulator within  $S_{IITM}$  processes theses additional calls to *op* independently of the protocol, which is true since  $S_{IITM}$  skips the balanced checks by the wrapper of  $S_{IITM}^{UC-bounded}$  while the innermost simulator  $\mathcal{S}_{UC}$  has this property by assumption. Thus, the formal difference in timing does not actually alter the behavior of the run. In particular, as soon as any other network message (i.e., a message that is not an additional call to op generated by the wrapper) is processed, we have that the states of the ideal protocol and the (internal) simulation is the same in both worlds, including the amounts of imports received so far. So both ideal worlds behave identical with the same output distribution.

Now consider the real world  $\mathcal{E}_{IITM}^{\text{single}} | \mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$  compared to the real world  $\mathcal{E}'_{IITM}^{\text{single}} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi\text{-id}}$ . Both worlds behave identical with all network

messages being delivered as long as  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  does not exceed its runtime budget. We show in the following that this case indeed does not occur by construction. At every point in a run,  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  has received total import from  $\mathcal{E}'_{IITM}^{\text{single}}$  at least  $\eta + p(s) + 2 \cdot t + u$ , where s is the total amount of import that the real protocol has received so far, t is the length of all network messages sent by the environment  $\mathcal{E}_{IITM}^{\text{single}}$ , and u is sufficient import to account for processing the additional op messages sent by the wrapper simulated in  $\mathcal{E}'_{IITM}^{\text{single}}$ . The total amount of import forwarded by  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  to the protocol is upper bounded by t due to the unary encoding of import used by  $\mathcal{E}_{IITM}^{\text{single}}$ . Hence, the adversary always keeps at least  $\eta + p(s) + t + u$  import for itself. As p(s) upper bounds the length of all network messages sent by the real protocol by construction (except for totalImport? responses, which are not limited by the import budget of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  anyways) and by definition of t and u, which account for all network messages sent by  $\mathcal{E}'_{IITM}^{\text{single}}$  to the adversary, we conclude that  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ indeed does not reach its runtime bound. It rather always has sufficient import for processing and forwarding all incoming messages in both directions while retaining at least  $\eta$  unused import for itself, as required by the UC runtime notion. So  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  never stops, just as  $\mathcal{A}_{Dum,IITM}$ , and hence the real worlds also behave identical with the same overall output distribution.

Since ideal and real worlds have the same output distribution: Since ideal and real worlds have the same output distributions and  $\mathcal{E}'_{IITM}^{single}$ cannot distinguish its systems as shown above, we obtain that  $\mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$ and  $\mathcal{S}_{IITM} | \phi_{IITM}^{\xi\text{-id}}$  cannot be distinguished by any single session environment  $\mathcal{E}_{IITM}^{single}$ .

**Indistinguishability for multiple sessions.** Finally, note that protocol sessions are disjoint due to the subroutine respecting property. Hence the second composition theorem of the IITM model (cf. Theorem 3) immediately implies that there exists a simulator  $S_{multi,IITM}$  such that  $A_{Dum,IITM} | \pi_{IITM}^{\xi-\text{id}}$  and  $S_{multi,IITM} | \phi_{IITM}^{\xi-\text{id}}$  are indistinguishable for all IITM environments. The simulator  $S_{multi,IITM}$  simply runs one copy of  $S_{IITM}$  for each protocol session (for completeness, we recall the second IITM composition theorem and provide a formalization of this statement in Appendix I). This gives the claim.

# C Proof of Lemma 3

In this section we show Lemma 3, which states that the reverse implication of Theorem 4 does not hold true in general. That is, under a certain complexity assumption there exist protocols  $\pi_{UC}$ ,  $\phi_{UC}$  such that we have  $\pi_{IITM} \leq_{IITM} \phi_{IITM}$  for the mapped protocols but  $\pi_{UC} \leq_{UC} \phi_{UC}$  does not hold true. On an intuitive level, the underlying reason is that the UC model imposes additional requirements on the simulator. Namely, the runtime of the simulator has to be bounded in a fixed polynomial in its import, where the import is determined by the environment. A simulator in the IITM model does not need to adhere to this added requirement. Hence this result is not due to any specific choices of

our mapping. It rather holds true for any mapping that allows for capturing the same protocol behavior, specifically the one constructed below, in both the UC and IITM models.

In what follows, we first show show two results that prove Lemma 3. We then further discuss the different classes of simulators considered by the UC and IITM models, including their implications for security.

**Proving Lemma 3.** For the construction of the protocols  $\pi_{UC}/\phi_{UC}$  resp.  $\pi_{IITM}/\phi_{IITM}$  we need the existence of so-called time-lock puzzles, a complexity assumption introduced in [23]. We use the definition from [20], which is in turn based on [13].

**Definition 3.** A time-lock puzzle consists of an ITM V (the verifier) and an ITM P (the prover) such that the following conditions are satisfied, where by  $\langle P, V \rangle$  we denote the distribution of the output of V after an interaction with P:

- Given an argument of the form (1<sup>η</sup>, s), V runs in polynomial time in η. Given an argument of the form (1<sup>η</sup>, s), P runs in polynomial time in η + s.
- 2. Easiness. For every polynomial p we have that

$$\min_{s \le p(\eta)} \operatorname{Prob}[\langle P(1^{\eta}, s), V(1^{\eta}, s) \rangle = 1]$$

is overwhelming (as a function in  $\eta$ ). (We call s the hardness of the puzzle.)

3. Hardness. For any ITM B running in polynomial time in the length of its first two arguments (i.e., in  $\eta + |a|$ ) there exists a polynomial p such that

$$\sup_{s\geq p(\eta+|a|)}\mathsf{Prob}[\langle B(1^\eta,a,s),V(1^\eta,s)\rangle=1]$$

is negligible (as a function in  $\eta$  and a).

Given a time-lock puzzle (P, V), the construction of  $\pi/\phi$  then uses a similar idea as given in [13]. The authors of [13] showed that their proposed polynomial runtime notion allows for simulation of protocols that cannot be simulated by simulators using the UC runtime notion of the 2005 version of the UC model [5]. Here we show that their idea for proving this result carries over to the IITM runtime notion, which is based on the runtime notion proposed in [13], and the UC runtime notion of the current journal version of the UC model [7], which still uses the same underlying idea of runtime tokens/import as in the 2005 version but changed several technical details such as the handling of the security parameter and the new balanced requirement for environments. Furthermore, our construction of  $\pi/\phi$  is also somewhat simpler than the protocols constructed in [13] as we do not require any involvement of the adversary in the real world and hence also no direct interactions of the simulator with the environment.

We define the protocol  $\pi$  as follows. An instance (pid, sid) of  $\pi$  ignores all messages on the network and in particular is incorruptible.  $\pi$  expects its first message from the environment to contain import at least  $\eta$  as well as a difficulty

s of a puzzle that the environment wants to solve. If the first message is different, then  $\pi$  halts permanently and discards any future messages. After receiving the initial message,  $\pi$  runs the interactive puzzle verification protocol with the environment, where  $\pi$  runs V and the environment is supposed to run the role of the prover (but might, of course, choose to deviate from the algorithm P). At the end of the protocol, i.e., once the verifier V has accepted, rejected, or aborted,  $\pi$  returns **real** to the environment.

Now, the protocol  $\phi$  is defined just as  $\pi$  but with the following modification. If the verifier outputs 1, i.e., accepts the puzzle, then, instead of returning an output to the environment,  $\phi$  sends the difficulty s of that puzzle to the simulator.  $\phi$ then expects the simulator to solve a puzzle of the same difficulty, i.e.,  $\phi$  runs the verifier V for difficulty s in the interactive puzzle verification protocol with the simulator. If V outputs 1 and hence also accepts the run with the simulator, then  $\phi$  returns **real** to the environment. In any other case, including the case that  $\phi$ is activated by the environment while puzzle verification with the simulator is still in progress,  $\phi$  returns **ideal** instead.

Note that both  $\pi$  and  $\phi$  meet the UC runtime notion. This is because they do not start running unless the have  $\eta$  import and then run the verification algorithm V at most two times, which itself runs in polynomial time in  $\eta$ . Hence,  $\pi_{UC}$  and  $\phi_{UC}$  are valid UC protocols and, by our mapping, the mapped protocols  $\pi_{IITM}$ and  $\phi_{IITM}$  are valid IITM protocols. We note that the following argument does not actually depend on our specific mapping; it rather holds for any reasonable modeling  $\pi_{IITM}$  and  $\phi_{IITM}$  of  $\pi$  and  $\phi$  in the IITM model.

Observe that the environment can distinguish both  $\pi$  and  $\phi$  only if the environment solves a time-lock puzzle of some difficulty *s* while the simulator does not. Hence, the natural definition for the simulator is to run *P* for difficulty *s* in order to solve the same puzzle as the environment. This is indeed possible in the IITM model; in particular, this simulator running with  $\phi$  fulfills the IITM runtime notion. We have the following:

# **Lemma 4.** Let $\pi_{IITM}$ and $\phi_{IITM}$ be as above. Then $\pi_{IITM} \leq_{IITM} \phi_{IITM}$ .

*Proof.* We define the simulator S to run the proving algorithm P on input s together with  $\phi$ . We have to argue two points. Firstly, the combination  $S | \phi_{IITM}$  meets the IITM runtime notion, i.e., is *environmentally bounded*. Secondly, there is only a negligible chance for the environment to solve a puzzle s successfully while the simulator running P on s does not.

Let  $\mathcal{E}$  by an arbitrary but fixed environment. Recall that, by the universally bounded property, the runtime of  $\mathcal{E}$  is upper bounded by a polynomial q in the security parameter and length of the external input. The hardness condition of time-lock puzzles hence implies that there is a polynomial p such that an environment has negligible probability of solving a puzzle with difficulty  $s \geq p(\eta + |a|)$ ; w.l.o.g we can assume that p is monotonically increasing. Hence, in an overwhelming set of runs we have that the environment cannot solve the puzzle or  $s < p(\eta + |a|)$ . In the first case, the simulator is never activated, so the combined runtime of  $\mathcal{S} \mid \phi_{IITM}$  is trivially bounded by the polynomial that bounds the runtime of  $\phi_{IITM}$ . In the second case, the simulator S additionally runs P on input s. By assumption, P is polynomial in  $\eta + s$ , i.e., there is a polynomial p'such that  $p'(\eta + s)$  upper bounds the runtime of P. W.l.o.g. we can assume that p' is monotonically increasing. Combined with the bound for s, we thus have that, except for negligible probability, the runtime of S is upper bounded by the polynomial  $p'(\eta + p(\eta + |a|))$ . Altogether we have that the runtime of  $S \mid \phi_{IITM}$ in a run with  $\mathcal{E}$  is bounded by a polynomial in  $\eta$  and |a| except for negligible probability, which is precisely the definition of *environmentally bounded*.

We still have to argue indistinguishability. Recall the definition of negligible function with external input, which considers inputs a bounded in length by an arbitrary but fixed polynomial in  $\eta$ . Let r be such an arbitrary but fixed polynomial. Given such r, we have to show that the probability of distinguishing both worlds is negligible in  $\eta$ . We already know that  $s < p(\eta + |a|)$  and thus  $s < p(\eta + r(\eta))$  with overwhelming probability in  $\eta$ . The easiness condition of time-lock puzzles implies that S running P solves the puzzle except for probability negligible in  $\eta$ . Hence  $\phi_{IITM}$  is indistinguishable from  $\pi_{IITM}$  except for probability negligible in  $\eta$  (depending on r), which was to show.

Observe that the polynomial that bounds the runtime of S and hence  $S | \phi_{IITM}$  actually depends on the environment  $\mathcal{E}$ . This is because the polynomial p in the proof depends on  $\mathcal{E}$ . Furthermore, the polynomial bounding the runtime of the simulator S actually is in the external input of the environment. Hence, if the runtime of a fixed environment is increased indirectly due to the external input being bounded by a larger polynomial, then the simulator is allowed to profit from the same increase in runtime. Such a dependence of the simulator runtime bound is possible under the IITM runtime notion. In contrast, the UC runtime notion requires a fixed polynomial for bounding the runtime of the simulator and also lets the environment decide upon the argument, i.e., the import provided to the simulator, of that polynomial. An environment in the UC model is free to provide only a minimal amount of import, so the UC simulator generally cannot adjust to computationally more powerful environments. This is the underlying reason for the impossibility result shown next, namely that there is no UC simulator such that  $\pi_{UC} \leq _{UC} \phi_{UC}$ .

As a warmup, observe that the simulator S from the above proof, which simply runs P on s to solve the puzzle, cannot be used for showing  $\pi_{UC} \leq_{UC} \phi_{UC}$ . This is because the environment can decide to provide just  $\eta$  import, in which case Shas to run in time polynomial in  $\eta$ . This is not sufficient for running P, which also depends on the difficulty s that is dynamically chosen by the environment. In other words, an environment can effectively "starve" the simulator. This additional class of attacks possible for UC simulators is exactly what we will use in the following proof.

**Lemma 5.** Let  $\pi_{UC}$  and  $\phi_{UC}$  be as above. Then there is no simulator such that  $\pi_{UC} \leq_{UC} \phi_{UC}$ .

*Proof.* Suppose there was such a simulator S. This simulator is required to meet the UC runtime notion, i.e., there is a polynomial p in the import of the simulator such that the runtime of S is bounded by that polynomial.

Now consider the class of environments that provide exactly  $\eta$  import to both the adversary/simulator and the protocol. Observe that the protocol, which is designed to run in polynomial time in  $\eta$ , is still able to perform all of its tasks. Hence, if the environment solves a puzzle of difficulty s, then the simulator is required to do the same (except for negligible probability) to be indistinguishable.

Since the simulator has fixed import  $\eta$ , he has runtime bounded by  $p(\eta)$ . By the hardness property of the puzzle, there exists a polynomial p' such that the simulator cannot solve puzzles of difficulty  $s \ge p'(\eta)$  (except for negligible probability). Conversely, by the easiness condition we have that P running on input  $s = p'(\eta)$  solves the puzzle in time polynomial in  $\eta + s$  (except for negligible probability). Let p'' be that polynomial such that  $p''(\eta + s) = p''(\eta + p'(\eta))$ bounds the runtime.

Now choose the environment  $\mathcal{E}$  from the aforementioned class (i.e., environments that provide only a minimal amount of import) that sets  $s = p'(\eta)$  and then runs P on s to solve the puzzle. Observe that  $\mathcal{E}$  is a valid UC environment since it is balanced and P runs in polynomial time in  $\eta$ , i.e.,  $\mathcal{E}$  meets the UC runtime notion. By the above arguments we have that there is an overwhelming probability for  $\mathcal{E}$  solving the puzzle for difficulty s while  $\mathcal{S}$  fails to do so, in which case both  $\pi$  and  $\phi$  are easily distinguishable.

The combination of Lemma 4 and Lemma 5 shows Lemma 3.

**On the difference of the simulator classes.** To summarize the above findings, both UC and IITM simulators run in polynomial time. In the IITM model, the polynomial runtime bound can depend on the environment as well as the length of the external input provided to the environment, which allows a simulator to compensate for environments that have additional computational resources. In contrast, the polynomial of UC simulators is both independent of the environment and independent of the external input (i.e., amount of import) available to the environment. So there are protocols where an environment can effectively overpower a UC simulator by using much more runtime itself. In other words, if a realization requires a simulator that can adjust to the computational resources available to an environment, then this realization is deemed safe and is composable in the IITM model but is considered unsafe and not composable in the UC model.

The natural question arising from this comparison is whether the UC notion of simulators is overly restrictive or whether the IITM notion of simulators is too lax and allows for proving security of realizations that should actually be deemed insecure. We argue for the former. The security of a realization  $\pi$  is defined via a suitable ideal protocol  $\phi$ . Note that ideal protocols such as  $\phi$  are not designed to exclusively run with a simulator in the ideal world of the security experiment. To facilitate composition, ideal protocols are rather also designed to be used as subroutines in hybrid protocols where they directly interact via the dummy adversary with the environment. Thus  $\phi$  must be defined to provide security guarantees even in the presence of environments whose runtime is not bounded

by a single fixed but rather any arbitrary polynomial that increases based on arbitrary external input. In other words, the security guarantees defined by  $\phi$  are actually independent of a specific class of simulators as long as that class is not more powerful than the class of environments (which is also a necessary requirement for obtaining a composition theorem).

So considering that the subroutine  $\phi$  is already defined to provide security while interacting over the network with environments bounded by arbitrary polynomials which also depend on the external input, allowing the simulator to use similarly flexible computational power, as is done in the IITM model, does not weaken the security statement for a realization  $\pi$  of  $\phi$ . Hence the class of IITM simulators appears to be a reasonable choice that allows for a wider variety of realizations compared to the UC simulator class while still being sufficiently restricted to obtain a composition theorem and hence enable protocol composition.

# D Proof of Theorem 5

Here we provide the proof of Theorem 5. Let us start by describing the class of simulators that Theorem 5 considers more precisely.  $S_{IITM}^{\text{UC-bounded}}$  must be built from some core logic and a wrapper such that the core logic adheres to the UC runtime notion and does not use the added totalImport? command.<sup>24</sup> The wrapper handles incoming totalImport? requests by forwarding them to  $M_{\text{msg}}$  and returning the responses. The wrapper also generates more import via calls to *op* if the IITM environment is not balanced, i.e., provides more import to the protocol than to the adversary, but otherwise does nothing.

protocol than to the adversary, but otherwise does nothing. First observe that the implication  $\pi_{IITM}^{\xi\text{-id}} \leq_{IITM} \phi_{IITM}^{\xi\text{-id}}$  has already been shown in the second half of the proof of Theorem 4 (cf. Figure 6). As for the other implication  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$ , this essentially reverses the argument of the first half of the proof of Theorem 4 (cf. Figure 5). More formally:

We define the UC simulator  $S_{UC}$  for the UC dummy adversary  $\mathcal{A}_{Dum,UC}$  to be the same as  $S_{IITM}^{\text{UC-bounded}}$  except for the wrapper. Observe that  $S_{UC}$  meets the UC runtime notion by the properties of  $S_{IITM}^{\text{UC-bounded}}$  and hence is a valid UC simulator. Now let  $\mathcal{E}_{UC}$  be a  $\xi$ -identity bounded UC environment that tries to distinguish  $\pi_{UC}$  running with  $\mathcal{A}_{Dum,UC}$  from  $\phi_{UC}$  running with  $\mathcal{S}_{UC}$ . We reduce this to the IITM case by constructing an IITM environment  $\mathcal{E}_{IITM}$  that internally simulates and hence behaves exactly as  $\mathcal{E}_{UC}$ . Note that  $\mathcal{E}_{UC}$  runs in polynomial time in its current import, where the total import in the whole system is the length of the external input |a|. Hence  $\mathcal{E}_{IITM}$  runs in polynomial time in the security parameter and external input, i.e., is universally bounded and thus a valid IITM environment.

<sup>&</sup>lt;sup>24</sup> The second requirement is actually not strictly necessary but simplifies both the protocol mapping and the proof, cf. Appendix H.3. It is also not a strong requirement since, as mentioned, the UC code of the ideal protocol must generally already provide at least the same information as totalImport? to the (core of) the simulator.

Now compare the real UC world with the real IITM world. By construction,  $\mathcal{E}_{UC}$  running with  $\mathcal{A}_{Dum,UC}$  and  $\pi_{UC}$  behaves identical to the system  $\mathcal{E}_{IITM} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$  with the same overall output as long as the wrapper of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  does not add any additional import. This only occurs if  $\mathcal{E}_{IITM}$  at some point in the run provides more total import to the protocol than it has provided to the adversary so far. However, since  $\mathcal{E}_{IITM}$  simulates  $\mathcal{E}_{UC}$  which is a balanced environment, this case never occurs and we have that both real worlds have the same output distributions. Furthermore, since  $\mathcal{E}_{IITM}$  simulates  $\mathcal{E}_{UC}$ ,  $\mathcal{E}_{IITM}$  is also  $\xi$ -identity bounded. This implies that  $\mathcal{E}_{IITM} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi-\text{id}}$  also behave identical with the same overall output distribution. The same argument shows that the ideal worlds, i.e.,  $\mathcal{E}_{UC}$  running with  $\mathcal{S}_{UC}$  and  $\phi_{UC}$  and  $\mathcal{E}_{IITM} | \mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi-\text{id}}$ , behave identical with the same overall output distribution.

Observe that  $\mathcal{E}_{IITM}$  only ever interacts with a single session of the protocol due to simulating  $\mathcal{E}_{UC}$ , which always has this property. Hence, by assumption  $\mathcal{E}_{IITM}$  cannot distinguish  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi\text{-id}}$  and  $\mathcal{S}_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi\text{-id}}$ . This implies that  $\mathcal{E}_{UC}$  also cannot distinguish the UC worlds, i.e.,  $\pi_{UC} \leq _{UC} \phi_{UC}$ , which was to show.

# E Proof of Corollary 1

Suppose we have  $\pi_{UC}$ ,  $\phi_{UC}$ ,  $\rho_{UC}$  UC protocols such that  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$  and the UC composition theorem can be applied to  $\rho_{UC}$  to obtain  $\rho_{UC}^{\phi \to \pi} \leq_{UC} \rho_{UC}$ . Let  $\rho_{IITM}^{\phi \to \pi}$  and  $\rho_{IITM}$  be the protocols obtained by applying our mapping to the IITM model, if necessary using the technique to model dynamically generated code from 3.5. To conclude that  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$  from Theorem 4, i.e., to show Corollary 1, we only have to check that the simulator constructed by the UC composition theorem is in the class considered for Theorem 4. That is, there must be a simulator for the composed protocol that implements the operation op(i) for providing import i to the dummy adversary without interacting with the ideal protocol. We argue that this is the case in what follows.

First, we need to recall the precise technical details of op(i) as implemented by the dummy adversary. By its definition (cf. Pages 42 and 43 of [7]), the dummy adversary expects inputs from the environment to be of the form (i, (m, id, c, i')), where *i* is the import provided to the adversary. The dummy then interprets this message by sending *m* with import *i'* to the machine with ID *id* and code *c*. If *id* is the unique ID of the environment, i.e., id = 0, then the dummy adversary instead sends the message *m* to the environment without any import. Note that id = 0indeed uniquely identifies the environment since all other machine instances, both from the protocol and the adversary, have IDs of the form id = (pid, sid). So formally op(i) is the input  $(i, (0, 0, \epsilon, 0))$  from the environment, where  $\epsilon$  is the empty word. This input provides import *i* to the dummy, who then returns the message 0 to the environment, i.e., the dummy behaves precisely as claimed for the operation op(i) (any message other than 0 could also be used and results in the same behavior).

Let us now take a look at the construction of the simulator in the proof of the composition theorem (cf. Figure 10 on Page 63 of [7]). Specifically, consider the implementation of op(i), i.e., the behavior of the simulator upon receiving in input from the environment of the form  $(i, (0, 0, \epsilon, 0))$  where id = 0. It turns out that the definition of the simulator is actually incomplete since it does not handle messages of this format at all, which, however, is required to be indistinguishable from the dummy adversary. The definition rather only specifies the behavior for cases where *id* has the form (pid, sid), i.e., for messages that are to be sent to the protocol (or adversary).

Looking at the proof and underlying arguments, one observes that the definition of the simulator can be completed by defining it to handle incoming messages from the environment for id = 0 in exactly the same manner as the dummy adversary. Indeed, this definition is consistent with the main idea for the simulator. Namely, upon receiving an incoming message for an ID id = (pid, sid)and code c, the composition simulator first checks whether the intended recipient is part of a subroutine session of  $\pi/\phi$ . If so, then the incoming message is handled by an internal copy of the simulator that shows  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$ . Otherwise, the composition simulator acts exactly as the dummy adversary, i.e., forwarding the message to the corresponding higher-level instance within  $\rho$  while observing the runtime bounds of the dummy. So if id = 0, which always belongs to the environment and is never used by any instance within any protocol, the simulator does not have to check whether this ID belongs to a session of  $\pi/\phi$  since he already knows that it does not. He can rather immediately act just as the dummy would, namely by returning the message to the environment. One easily verifies that this way of completing the definition of the simulator is consistent with the rest of the proof of the UC composition theorem. In particular, this definition of the behavior for target ID id = 0 is easily simulatable by the hybrid environments constructed in the proof.

So altogether we have that the operation op(i) can be implemented in the composition simulator in the same manner as in the dummy adversary. Hence, that simulator is in the class of simulators that are required by Theorem 4.

# F Proof of Corollary 2

In this section we provide full formal details and the proof for Corollary 2. We then show how and why this result extends to the case of protocols with dynamic code (Section 3.5), where some machine codes are handled by instances of a universal Turing machine  $M_{\rm UT}$ .

Showing Corollary 2. Let us first briefly discuss a technical detail concerning  $\rho_{UC}/\rho_{IITM}$ . In mapping  $\rho_{UC}$  to  $\rho_{IITM}$  and hence to obtain Corollary 1, we formally considered  $\rho_{UC}$  with the standard subroutine respecting wrapper code  $cs_{\rho}$  added to all of its machines, which prefixes the SIDs of all subroutines of a session of  $\rho$  by the SID  $sid_{\rho}$  of that session but otherwise preserves the behavior of each individual session. This ensures that  $\rho_{UC}$ , which was constructed and proven secure only in a single session setting, remains secure in a multi session

setting due to disjoint subroutines. Note that the realization  $\rho_{UC}^{\phi \to \pi}$  of  $\rho_{UC}$ carries over after adding such wrappers to both protocols since the resulting protocols still behave identical when used in a single session, i.e., one can use the simulator that essentially works as before while internally adding/removing the wrapper code and SID prefixes for network messages. This is a standard technique and argument that is implicitly used throughout the UC literature, where protocols  $\rho_{UC}$  are generally defined with a single session in mind. Without this technique and argument it would hence be virtually impossible to re-use and compose such a  $\rho_{UC}$  as a subroutine of another protocol, where  $\rho_{UC}$  has to be subroutine respecting, i.e., remain secure even when multiple sessions of  $\rho_{UC}$ run concurrently. As a result of the addition of  $cs_{\rho}$  to all machines in  $\rho_{UC}$ , the code of the subroutine  $\phi_{UC}$  has now formally changed to additionally include the wrapper code  $cs_{\rho}$  which prefixes all SIDs of instances of  $\phi_{UC}$  with the SID  $sid_{\rho}$ of the current session of  $\rho_{\mathit{UC}}$  but hides that prefix from the inner logic. Just as for the overall  $\rho_{UC}$ , adding this wrapper does not actually change the behavior of  $\phi_{UC}$  (or really any other subroutine) while running within a single session of  $\rho_{UC}$ . Hence, if we consider the same wrapper  $cs_{\rho}$  added to  $\pi_{UC}$ , then we still have  $\pi_{UC} \leq \xi_{UC} \phi_{UC}$  for a modified  $\xi$  that additionally requires the environment to call  $\pi_{UC}/\phi_{UC}$  as a subroutine of some instance within a single session of  $\rho_{UC}$ , i.e., using the same wrapper  $cs_{\rho}$  and SIDs that are consistent with the SID prefix used by the wrapper  $c_{s_{\rho}}$  of  $\pi_{UC}/\phi_{UC}$  (again, the simulator is essentially the same as before but internally adds/removes the wrapper code and SID prefixes). Finally observe that our mapping from UC to IITM protocols and all corresponding results still apply essentially unchanged to these updated  $\pi_{UC}/\phi_{UC}$  that include the wrapper  $cs_{\rho}$  on top of their own wrapper  $cs_{\pi}/cs_{\phi}$ . The only formal difference is that the challenge session ID  $sid_c$  now consists of two elements, namely the prefixed SID used by  $cs_{\rho}$  and the inner SID identifying a session of  $\pi/\phi$  (within that session of  $\rho$ ). Hence we can still obtain mapped protocols  $\pi_{IITM}$  and  $\phi_{IITM}$ such that  $\pi_{IITM}^{\xi \text{-id}} \leq_{IITM} \phi_{IITM}^{\xi \text{-id}}$  by the same reasoning as for Theorem 4. In the rest of this section, we therefore leave this technical detail of the additional wrapper  $cs_{\rho}$  implicit as it does not affect any of our results but would distract from the core ideas of the following construction.

Next, we describe the the syntactic changes to  $\rho_{IITM}$  for obtaining Corollary 2, i.e., the transition from the left side of Figure 4 to the protocol in the middle. Let  $\rho_{IITM}$  be the protocol mapped according to Section 3.3, where we denote the resulting IITMs by  $M_i^{\rho}$ . One of these machines is running code  $c_{\phi}$ , say,  $M_{c_{\phi}}^{\rho}$ . The left and middle of Figure 4 illustrate the idea of our syntactical changes to  $\rho_{IITM}$ : We extend the protocol  $\rho_{IITM}$  by additionally including the full set of machines of  $\phi_{IITM}$ , as mapped by Section 3.3, and then reroute all inputs to and outputs from  $M_{c_{\phi}}^{\rho}$  to instead be sent to  $\phi_{IITM}$  via its single external I/O tape. Since such inputs and outputs are the only way for  $\rho$  to interact with instances in any session of the subroutine  $\phi$  (by the subroutine respecting property of  $\phi$ ), this does not actually change the behavior of the modified  $\rho_{IITM}$ . It, however, consistently moves all sessions of  $\phi$  to be instances of the set of machines  $\phi_{IITM}$  (instead of instances of machines  $M_i^{\rho}$ ). The set of machines  $\phi_{IITM}$  can then be replaced by the set of machines  $\pi_{IITM}$  using the IITM composition theorem (right side of Figure 4), thereby replacing all sessions of  $\phi$  with sessions of  $\pi$ , yielding a protocol  $\rho \phi^{\to \pi}_{IITM}$  that behaves just as  $\rho \phi^{\to \pi}_{UC}$ .

More formally, first map the protocol  $\phi_{IITM}$  according to Section 3.3 with machines  $M_j^{\phi}$ , where  $M_{c_{\phi}}^{\phi}$  is the highest-level machine running code  $c_{\phi}$  and offering a single external I/O tape for others to connect to. We start modifying  $\rho_{IITM}$  by adding the static set of machines  $M_i^{\phi}$  to the system  $\rho_{IITM}$ . We then connect all machines  $M_i^{\rho}$  to the single external I/O tape of  $M_{c_{\phi}}^{\phi}$ . Since there is only a single I/O tape available but multiple different (instances of) machines  $M_i^{\rho}$  have to be able to send inputs to and receive outputs from  $\phi_{IITM}$ , we introduce a straightforward multiplexer  $M_{\text{multiplex}}$ . We connect every machine  $M_i^{\phi}$  via one I/O tape used to  $M_{\text{multiplex}}$  for providing inputs to and receiving outputs. We further connect  $M_{\text{multiplex}}$  to the single external tape of  $\phi_{IITM}$ . In a run of the protocol, when an instance of  $M_{\text{multiplex}}$  receives a message  $((pid, sid, c_{\phi}), (pid_s, sid_s, c_s), m')$  on an I/O tape connected to machine  $M_{c_s}^{\rho}$ then  $M_{\text{multiplex}}$  forwards the message  $((pid, sid), (pid_s, sid_s, c_s), m')$  on the single I/O tape connected to the subroutine machine  $M_{c\phi}^{\phi}$ . If  $M_{\text{multiplex}}$  receives output  $((pid, sid, c), (pid_s, sid_s), m')$  on that tape, then the instance of  $M_{\text{multiplex}}$  sends the message  $((pid, sid, c), (pid_s, sid_s, c_{\phi}), m')$  on the unique tape connected to  $M_c^{\rho}$ . This tape exists since, by assumption,  $\rho_{UC}$  uses only finitely many machine codes, which includes the codes used in outputs of the subroutine  $\phi_{UC}$ .

Finally, we modify the machines  $M_i^{\rho}$  to reroute all inputs to and outputs from  $M^{\rho}_{c_{\phi}}$  to instead be sent to/received from  $\phi_{IITM}$ , specifically  $M^{\phi}_{c_{\phi}}$ , via the multiplexer  $M_{\text{multiplex}}$ . As explained above, the subroutine respecting property of  $\phi_{UC}$  should imply that this does not change the behavior of  $\rho_{IITM}$ . However, it turns out that we need another property. Specifically, we require that higher-level instances of  $\rho_{UC}$  do not use predicates for sending outputs via a non-forced write commands<sup>25</sup> that match an instance in a session of  $\phi_{UC}$  (respectively  $\pi_{UC}$  in the composed protocol). Similarly, a session of  $\phi_{UC}$  may also not use a predicate for non-forced write outputs that, in a run within  $\rho_{UC}$ , matches instances of  $\rho_{UC}$  that are not part of the same session of  $\phi_{UC}$  (analogously for  $\pi_{UC}$  in the composed protocol). This is because such predicates create a side channel that is not available to an environment and thus prevents composition. Indeed, after observing that this second requirement is necessary for being able to modify  $\rho_{IITM}$  such that it supports composition via the IITM theorem, we found that the same issue also occurs for the UC theorem, i.e., the UC theorem is currently false but can be fixed by adding the same requirement (cf. Appendix J.4 for full details). Since this is necessary for composition in both the IITM and UC models, we assume in what follows that  $\rho_{UC}$  already has this property. Analogous to Lemma 1, we then have:

**Lemma 6.** Let  $\rho_{IITM} = ! M_{c_{\rho}}^{\rho} | \dots | ! M_{n}^{\rho} | ! M_{multiplex} | \phi_{IITM}$  be as described above. For all unbounded environments interacting with  $\rho_{UC}/\rho_{IITM}$ , also directly

 $<sup>^{25}</sup>$  Recall that the UC model already requires  $\rho_{\mathit{UC}}$  to use forced writes for inputs.

# via the network, there is a bijective mapping between runs of both protocols such that they behave identically.

*Proof.* Before the modification of  $\rho_{IITM}$ , this followed by construction of our protocol mapping. The only change in the redefined  $\rho_{IITM}$  is that we have moved all calls providing input to the subroutine  $\phi_{UC}$  from  $M_{c_{\phi}}^{\phi}$  to a new set of machines  $\phi_{IITM}$ . This does not alter the behavior of the protocol:

Observe that the codes of the protocol  $\phi$ , including all subroutine codes, contain the standard subroutine respecting shell code  $sc_{\phi}$  (this code is specific to  $\phi$ ). This code, among others, ensures that the only messages sent by any instance in a session of  $\phi$  to an instance that is not in that session are outputs by instances running  $c_{\phi}$ . Now consider messages sent by a higher-level instance of  $\rho$ (that is not part of a session of  $\phi$ ) addressed to one of the codes c used by  $\phi$  and which might thus be sent to an instance in a session of  $\phi$ . If this message is an input to  $c_{\phi}$ , then this message is consistently redirected to  $M_{c_{\phi}}^{\phi}$  in our modified  $\rho_{IITM}$ . In all other cases, the message is sent to an instance of  $M_c^{\rho}$  (both before and after the modification), which runs code c. The subroutine respecting shell code  $sc_{\phi}$  contained in c then immediately drops the message, which, again, is the same before and after the modification. Altogether, the only way for higher-level instances of  $\rho$  and instances in sessions of  $\phi$  to communicate without dropping messages is via inputs received and outputs sent by  $c_{\phi}$ , which are consistently rerouted via the multiplexer to  $M_{c_{\phi}}^{\phi}$  in our modification.

Next observe that the introduction of a second network interface (namely the network tapes of the additional machines  $M_c^{\phi}$ ), where now all network messages from and to sessions of the subroutine  $\phi$  are sent/received, also does not provide any additional knowledge to the environment. This is because the environment can already determine whether network messages are sent from/to a session of  $\phi$ from the extended ID of the sender/receiver. Specifically, all the machine codes of all instances of any session of  $\phi$  contain the subroutine respecting shell code  $sc_{\phi}$ . Conversely, by definition of this shell code if some instance running with that shell code is not part of a session of  $\phi$  then that instance drops all incoming network messages and never sends an outgoing network message. Hence, by using the existence of the shell code  $sc_{\phi}$  in the sender/receiver extended identity as an indicator, an environment (or adversary/simulator on the network) can easily map between the network behavior of the original protocol with only a single network interface and the modified  $\rho_{IITM}$  where there are two separate network interfaces that are used depending on whether the sender/receiver is part of a session of  $\phi$  (this mapping works in both directions).<sup>26</sup>

<sup>&</sup>lt;sup>26</sup> Alternatively, instead of checking for the existence of this shellcode in the extended sender/receiver ID, the same information can also be obtained by querying the directory machine of  $\phi$  to determine whether the sender/receiver is part of a session of  $\phi$ . This is the same idea that is used to construct the simulator in the proof of the UC composition theorem. Indeed, the purpose of the directory machine is to allow a simulator to determine whether network messages belong to a higher-level instance or an instance in a session of  $\phi$ , i.e., obtain the same information as provided by the two separated network interfaces of our modified  $\rho_{IITM}$ .

So the only formal difference between the original and the modified  $\rho_{IITM}$  is how non-forced writes are handled. Observe that in the modified  $\rho_{IITM}$  there are now two machines executing such writes: The machine  $M_{\rm msg}^{\rho}$  belonging to  $\rho_{IITM}$  handles only higher-level instances of  $\rho_{IITM}$  (since the new subroutine instances of  $\phi_{IITM}$  do not register themselves in that machine). On the other hand  $M_{\text{msg}}^{\phi}$  within  $\phi_{IITM}$  handles only instances of a session of  $\phi_{IITM}$  but not any other instances of  $\rho$ . Here we use the second requirement discussed above, namely, predicates of a session of  $\phi_{IITM}$  never match instances of  $\rho$  that are outside of the session of  $\phi_{IITM}$  and vice versa. By this, both separated machines still behave in the same way as the unmodified protocol  $\rho_{IITM}$  where a single machine  $M_{\rm msg}^{\rho}$  handles all instances.

Altogether the behavior of the modified  $\rho_{IITM}$  is still the same and in particular still just as for  $\rho_{UC}$ . 

Next, we define the composed protocol  $\rho_{IITM}^{\phi \to \pi}$  (right side of Figure 4) by replacing the set of machines  $\phi_{IITM}$  with the set of machines  $\pi_{IITM}$ . That is,  $\rho_{IITM}^{\phi \to \pi} = ! M_{c_{\rho}}^{\rho} | \dots | ! M_{n}^{\rho} | ! M_{\text{multiplex}} | \pi_{IITM}$ . In particular, the single I/O tape from the multiplexer  $M_{\text{multiplex}}$  to  $\phi_{IITM}$  is simply reconnected to the single external I/O tape of  $\pi_{IITM}$ . As a result, all inputs to and outputs from sessions of  $\phi_{IITM}$  are now instead handled by sessions of  $\pi_{IITM}$ , which is just as in  $\rho_{UC}^{\phi \to \pi}$ . In other words, reconnecting this tape has the same effect as adding the UC composition shell code, which internally changes the code  $c_{\phi}$  to instead be  $c_{\pi}$  for such inputs/outputs. So, unlike in Corollary 1, when we use the IITM composition theorem we actually do not need to include this shell code in  $\rho_{UTM}^{\phi \to \pi}$ . Altogether we have:

**Lemma 7.** For all unbounded environments interacting with  $\rho \frac{\phi \to \pi}{UC} / \rho \frac{\phi \to \pi}{IITM}$ , also directly on the network, there is a bijective mapping between runs of both protocols such that they behave identically.

*Proof.* Directly follows from the above observation.

We can now prove Corollary 2:

Proof (Corollary 2). We have  $\pi_{IITM}^{\xi \text{-id}} \leq_{IITM} \phi_{IITM}^{\xi \text{-id}}$  by Theorem 4. We modify  $\rho_{IITM}$  by replacing  $\phi_{IITM}$  with  $\phi_{IITM}^{\xi \text{-id}}$ , which essentially adds the identity machine  $M_{\text{identity}}^{\xi}$  between  $M_{\text{multiplex}}$  and  $M_{c_{\phi}}^{\phi}$ , to obtain a protocol  $\rho_{IITM,id}$ . Since  $\rho_{UC}$  respects the predicate  $\xi$  while sending inputs to the subroutine  $\phi$ , the same holds for  $\rho_{IITM}$ . Hence  $\rho_{IITM}$  and  $\rho_{IITM,id}$  behave identical, i.e.,  $\rho_{IITM,id} \leq_{IITM} \rho_{IITM}$  for the simulator that is the dummy adversary.

Since  $\pi_{IITM}^{\xi,id} \leq_{IITM} \phi_{IITM}^{\xi,id}$ , the composition theorem of the IITM model (Theorem 2) implies  $\rho_{IITM}^{\phi \to \pi}, id \leq_{IITM} \rho_{IITM,id}$ . We can finally repeat the same reasoning as in the first step to obtain  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM,id}^{\phi \to \pi}$ . Note that here we use that if  $\rho_{IITM}$  respects  $\xi$ , then  $\rho_{IITM,id}^{\phi \to \pi}$  also does the same (except up to a negligible set of runs) for our proposed definition of  $\xi$ -identity boundedness. If this were not the case, an

environment could distinguish  $\pi_{IITM}^{\xi\text{-id}}$  and  $\phi_{IITM}^{\xi\text{-id}}$  by internally simulating higher level instances of  $\rho$  and checking whether  $\xi$  is met for the sequence of messages sent by those instances to the external subroutine  $\pi_{IITM}^{\xi\text{-id}}/\phi_{IITM}^{\xi\text{-id}}$ , which contradicts  $\pi_{IITM}^{\xi\text{-id}} \leq_{IITM} \phi_{IITM}^{\xi\text{-id}}$ .

By transitivity of the  $\leq_{IITM}$  relation we obtain  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$ .  $\Box$ 

Corollary 2 for protocols with dynamic code (Section 3.5). As for whether the IITM composition theorem allows for obtaining the same kinds of composition results also for protocols with dynamic code (i.e., Corollary 2), there are essentially two cases to consider. Firstly, a protocol  $\phi_{IITM}$  with dynamic code is a fixed subroutine of a more complex protocol  $\rho_{IITM}$ , i.e., the code  $c_{\phi}$  is not dynamically generated by  $\rho$  but rather part of Codes<sub> $\rho$ </sub> ( $\rho$  might still choose to dynamically generate other machines codes). The IITM composition theorem directly implies that we can replace  $\phi_{IITM}$  with a realization  $\pi_{IITM}$  by the same reasoning as for Corollary 2. Note in particular that the IITM composition theorem is agnostic of the internal behavior of the machines of  $\pi_{IITM}$ ,  $\phi_{IITM}$ , and  $\rho_{IITM}$ , i.e., composition is independent of whether one of the internal machines is a universal Turing machine.

The second case is more interesting: Suppose that we have a protocol  $\rho_{IITM}$  that uses dynamically generated code, i.e., has a universal Turing machine. The UC composition theorem allows for replacing some arbitrary code  $c_{\phi}$ , including all of its subroutines, with the code of some realization  $c_{\pi}$  even if the code  $c_{\phi}$  is dynamically generated by  $\rho$  (and hence not in Codes<sub> $\rho$ </sub>). In  $\rho_{IITM}$ , where dynamically generated code is represented via instances of  $M_{\rm UT}$ , this composition operation corresponds to replacing some but not all instances of  $M_{\rm UT}$ , namely exactly those instances that run code  $c_{\phi}$  (as well as their subroutines), with instances running code  $c_{\pi}$ . However, the IITM composition theorem does not directly allow for replacing subsets of instances of the same machine; only the full machine with all of its instances can be replaced by a realization.

This mismatch can be resolved via some simple purely syntactical changes using the same modeling technique from Section 3.4 and illustrated in Figure 4. Namely, we can first add the machines  $M_i^{\phi}$  of the subroutine  $\phi_{IITM}$  to  $\rho_{IITM}$ . We then reroute all messages from/to instances of  $M_{\rm UT}$  that run code  $c_{\phi}$  (and that hence should be replaced via composition) via a multiplexer  $M_{\text{multiplex}}$  to now instead be handled by an instance of  $M_{c_{*}}^{\phi}$ . By the same reasoning as for Lemma 6, the modified protocol still behaves identical. Note in particular that this reasoning is actually independent of whether  $c_{\phi}$  is dynamically generated, i.e., whether instances running code  $c_{\phi}$  in the unmodified protocol  $\rho_{IITM}$  are instances of some fixed machine  $M^{\rho}_{c_{\phi}}$  or a subset of the instances of the universal Turing machine  $M_{\rm UT}$ . In both cases, since inputs to and outputs from  $c_{\phi}$  are consistently rerouted to the new machine  $M_{c_{\phi}}^{\phi}$  in  $\phi_{IITM}$  and such inputs/outputs are the only way for a higher-level protocol to interact with any instance in any session of  $\phi$ , we have that this change consistently moves all instances in all sessions of  $\phi$  to be instances of machines in  $\phi_{IITM}$  and hence does not alter the behavior.

The IITM composition theorem then directly implies that we can replace the machines  $\phi_{IITM}$  with their realization  $\pi_{IITM}$  by the same argument as for Corollary 2. Observe that, since the above rerouting of messages ensures that all sessions of  $\phi$ , which were previously instances of  $M_{\rm UT}$ , are now exactly the instances of the machines of  $\phi_{IITM}$ , this composition operation precisely replaces all sessions of  $\phi$  with sessions of  $\pi$ . This is exactly as for the UC composition theorem.

# G A New Type of Composition

Recall that the UC theorem applied to a protocol  $\rho$  replaces all sessions of subroutines running code  $\phi$  with sessions running code  $\pi$ . Similarly, the IITM theorem applied to a protocol  $\rho$  replaces all sessions of a set of machines  $\phi$  with sessions of a set of machines  $\pi$ . Using our modeling technique from Sections 3.4 and 3.5, where we syntactically move sessions of a protocol  $\phi$  to a different set of machines without changing the semantic behavior of the overall protocol  $\rho$ , it is actually possible to obtain a more general type of composition as a simple corollary of the existing composition theorems of both models (and other similar models): under certain conditions, we can implement a proper subset of the sessions of  $\phi$  with a realization  $\pi$  while the remaining sessions can be implemented using a different realization, say,  $\pi'$ . This can be useful, e.g., if  $\phi$  is an ideal signature functionality, where each session models one key pair belonging to a certain party, then we might want to implement certain keys with a signature scheme  $\pi$  but others with a second signature scheme  $\pi'$ . In what follows, we explain the underlying idea as well as necessary requirements which allow for using our modeling technique to perform this type of composition. The precise technical details then depend on the underlying universal composability model and protocol  $\rho$  under consideration and hence need to be filled in on a case by case basis.

More specifically, suppose that we want to replace a proper subset of the sessions of  $\phi$  (in runs of a protocol  $\rho$ ) with a realization  $\pi$  while leaving all remaining sessions of  $\phi$  alone. We need that  $\pi/\phi$  have disjoint sessions that do not interact with each other. In the UC model, this is guaranteed due to the subroutine respecting property. In the IITM model, this is a special case of protocols (cf. Appendix I). We further need that higher-level instances of  $\rho$  as well as the adversary on the network are able to compute which sessions of  $\phi$  are to be replaced by  $\pi$  and which are to be left alone. The probably most common way to achieve this property is by specifying the set of sessions that are to be replaced via a polynomial time indicator function on a (prefix of a) SID shared by all instances of a session of  $\phi$ . In our above example, where  $\phi$  is an ideal signature functionality and the SID models a signing key pair belonging to a party, the indicator function could be such that it decides based on the owner of the key whether that session is implemented via  $\pi$ . Alternatively,  $\rho$  might be such that the same party uses multiple keys for different purposes in the protocol, where

each of these keys has its own SID. In that case the indicator function can decide the implementation depending on where/how the key is used in  $\rho$ .

Given protocols with the above properties, we can use our modeling technique from Sections 3.4 and 3.5 to move all sessions of  $\phi$  which should *not* be realized by  $\pi$  to instead be sessions of a new separate set of machines/codes  $\phi'$  that behaves identical to  $\phi$ . In the IITM model,  $\phi'$  is simply a second set of IITMs with the same code as  $\phi$  as shown in Section 3.4. In the UC model, the code  $\phi'$  formally needs to be different from  $\phi$  (otherwise, the composition theorem would not just replace sessions of  $\phi$  but also replace all sessions of  $\phi'$ ). This is easily achieved by adding some additional bit to the code of  $\phi$  that is never used or by adding a dummy machine on top of  $\phi$ , both of which formally change the code but do not affect the behavior. Note that all realizations of  $\phi$  are also realizations of  $\phi'$ . In the IITM model, this follows from the fact that we simply copied the set of machines  $\phi$ , which is possible in the IITM model. In the UC model, we formally changed the code of  $\phi$  to obtain  $\phi'$ , but since this is done in such a way that the behavior is not changed, one can verify that existing realizations of  $\phi$  carry over. We finally alter all higher-level instances of  $\rho$  as follows: whenever they are about to send a message to a session of  $\phi$ , they first check whether the recipient session is one that shall be realized by  $\pi$ . If not, then the message is instead sent to  $\phi'$ . When a higher-level instance of  $\rho$  receives a message from such a session of  $\phi'$ , it acts as if this message was received by  $\phi$ .

Due to session disjointness, separating sessions of  $\phi$  to be run on two different sets of machines/codes  $\phi$  and  $\phi'$  is a purely syntactical modification that does not actually alter the semantics/behavior of  $\rho$ . More formally, one can show that the new  $\rho$  realizes the original one: since a simulator is also able to determine which sessions of  $\phi$  are to be modified and which ones are to be left alone, he can simply reroute network messages accordingly. Since sessions of  $\phi$  from the original  $\rho$  are now split to be sessions of two separate machines/codes  $\phi$  and  $\phi'$  in the modified  $\rho$ , we can use the UC/IITM composition theorem to replace all sessions of  $\phi$  with sessions of  $\pi$  while leaving the sessions of  $\phi'$  as is. This is precisely what we wanted to achieve, namely, replace a subset of all sessions!

One can apply the composition theorem a second time to replace also the remaining sessions, i.e., the sessions of  $\phi'$ , but with a different realization  $\pi'$ . Alternatively, one can iterate the above technique to further split the remaining sessions into separate machines/codes  $\phi'$  and  $\phi''$  that can again be realized independently.

# H A More Complex Variant of our Protocol Mapping

In what follows, we describe an extended variant of our mapping which models more closely the precise technical details of the UC runtime notion for adversaries. The main difference is the following. In the mapping described in the main body, the adversary gets access to an totalImport? command that reveals the protocol import of a session (as mentioned, the code of the UC protocols generally must already provide at least the same information to allow for a simulation in the

first place. So we do not actually change the security properties modeled by the protocol by making this information given to the adversary explicit). The idea for the IITM adversary/simulator then is that he can manually enforce the balanced requirement on environments by checking the amount of import received by the protocol and, if this is more than he has received so far, simply add the missing difference to his own import budget. Note that there formally is a slight mismatch in this construction between the UC and IITM worlds: In the IITM world, the totalImport? mechanism allows the simulator to add additional import after the protocol has already received it. In the UC model, due to the balanced property of environments the simulator is guaranteed to obtain additional import before the protocol is provided with the same. As we show in Theorem 4, however, this slight mismatch is not an issue as long as the UC simulator allows for adding runtime without interacting with and hence influencing the state of the ideal protocol. As explained in Section 3.3, this includes all existing UC protocols from the literature. Indeed, it seems that no reasonable protocol definition should require some interaction by the network adversary/simulator whenever the network adversary/simulator increases its own runtime, also considering that this information is not available to protocols in reality. However, formally one might be able to build artificial protocols with realizations such that simulation requires the simulator to contact the protocol whenever the simulator receives additional import.

To cover arbitrary simulators, including those not meeting the requirement of Theorem 4, the extended variant of our mapping guarantees that the IITM adversary is activated and obtains import before the protocol import exceeds the import of the adversary, just as in the UC model. This resembles the precise behavior of the UC model and hence resolves the above slight mismatch, thereby removing the need for a specific class of simulators in Theorem 4. As an interesting side note, this mapping also does not require the explicit totalImport? request; the IITM adversary instead just learns some upper bound of the import received by the ideal protocol, which is the exact guarantee that a balanced UC environment provides to a UC adversary (again, we emphasize that the code of the UC protocol generally already provides at least the same information as totalImport? to the UC adversary and hence, after applying our mapping, also to the IITM adversary. We just do not have to make this explicit in this variant of our mapping). On the downside, this extended mapping leads to protocols which, while still modeling the intended original protocol behavior, use more complex and less natural definitions, in particular when they are composed using the IITM composition theorem. We therefore chose to present the simpler mapping in the main body, which should already cover all protocols of practical interest.

Next, we describe how our mapping is changed for the following variant. We then explain how the theorems and proofs from Section 3.3 are changed. Most notably, we no longer require a specific type of simulator for the updated version of Theorem 4. We then also explain how this change in the mapping affects higher-level protocols that are modeled to be compatible with the IITM composition theorem such as the one described in Section 3.4.

## H.1 Updates to the mapping

The variant of our mapping is mostly the same as the one described in 2.2 but changes how import related to the adversary is handled. Firstly, the totalImport? command is removed. Instead, the highest-level machine  $M_{c_{\pi}}$  now provides an additional provideAdversarialImport request to the environment that, by suitable encoding, is distinguishable from regular inputs intended for the code  $c_{\pi}$ . The intention of this request is to give the environment the option to provide some import i to the adversary via the protocol. The protocol itself can then check that the balanced requirement is met at all times and block messages otherwise. More specifically, the IITM environment can send the input (provideAdversarialImport, i), where i is an import encoded in unary. This input is then forwarded to  $M_{\rm msg}$ , which keeps track of all adversarial imports provided in this manner so far (in addition to the protocol imports that are already tracked).  $M_{\rm msg}$  then forwards this message to the IITM adversary on the network who will, jumping slightly ahead, interpret this to be the same as a direct network message from a UC environment that provides additional import *i*. He will then perform the same actions as in this case, potentially including any interactions with the ideal protocol. This then has the exact same timing as in the UC case, where an environment directly provides this import, and thereby allows us to show Theorem 4 for arbitrary simulators. To enforce the balanced requirement, the machine  $M_{c_{\pi}}$  is changed as follows. Each time a regular input (i.e.,  $\neq$  provideAdversarialImport) carrying some import > 0 arrives from the environment on the single external I/O tape,  $M_{c_{\pi}}$  first checks with  $M_{msg}$ whether accepting this message would increase the total import obtained by the protocol from the environment to be larger than the total import provided by provideAdversarialImport requests. If so, then  $M_{c_{\pi}}$  drops this message. In particular, the import contained in that message is not added to the list of protocol imports received so far. If not, i.e., the environment is still balanced, then the incoming message is processed as usual, which includes storing the new import amount in  $M_{\rm msg}$ .

This already allows for showing a version of Theorem 4 for this variant of our mapping but without assumptions on simulator. To also show the corresponding version of Theorem 5, we additionally need the following mechanism. We extend  $M_{msg}$  to allow the adversary to send messages of the form (registerAdversarialImport, i) on the network, where i is an import encoded in unary. These messages allow the adversary to notify the protocol in case the adversary has received some import i by the environment directly, say, as part of some regular network messages. More formally,  $M_{msg}$  treats these messages in the same way as (provideAdversarialImport, i) by adding i to the list of adversarial imports so far.  $M_{msg}$  then returns an acknowledgement to the adversary, say (registerAdversarialImport, ok).

One easily verifies that the above modified mapping yields IITM protocols that, while running with an IITM environment, behave just as the original UC protocols running with a balanced UC environment. Specifically, the added command (provideAdversarialImport, i) available to IITM environ-

ments corresponds to the UC environment calling op(i) on the network, where op is the operation for providing the dummy adversary with import, while the **registerAdversarialImport** command on the network corresponds to the UC environment providing some import for the adversary as part of a network message  $\neq op$ . Indeed, this correspondence in both directions is what we show and use in the following proofs. Hence, the mapped IITM protocol is still faithful to the intended protocol behavior.

# H.2 Adjusting Theorem 4 and its proof

As mentioned, for this variant of our mapping we can show a stronger version of Theorem 4 that does not require a simulator who can receive import without having to interact with the ideal protocol. In what follows, we explain how the proof of Theorem 4 needs to be adjusted.

Indistinguishability for UC runtime bounded adversaries. We define the IITM dummy adversary  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  that adheres to the UC runtime notion as follows.  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  still consists of a wrapper that internally runs  $\mathcal{A}_{Dum,UC}$ . If the wrapper of  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  receives (provideAdversarialImport, i) from the protocol, then it internally calls op(i) of  $\mathcal{A}_{Dum,UC}$ , where op is the operation that can be used by the UC environment to provide the UC dummy with additional import. If the wrapper receives a registerAdversarialImport request from the environment on the network for  $M_{\rm msg}$ , then this request is dropped (we will handle such requests in a later step when we build a simulator for the regular IITM dummy adversary that is not UC bounded and simply forwards all requests, including registerAdversarialImport). If the wrapper of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  receives some import i from the environment as part of some network message (which might be a regular message that is intended to be forwarded to the protocol but might also be a call to op), then the wrapper first registers this import in the ideal protocol by sending (registerAdversarialImport, i) and waiting for the response. The wrapper then forwards the network message from the environment to  $\mathcal{A}_{Dum,UC}$ . In all other cases, i.e., when the wrapper receives some message from the protocol, environment, or  $\mathcal{A}_{Dum,UC}$  (including notifications by  $\mathcal{A}_{Dum,UC}$  to the environment for calls to op), it forwards the message accordingly between protocol/environment and  $\mathcal{A}_{Dum, UC}$ .

Note that a major difference between the wrapper of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  defined here and the original wrapper defined in the proof of Theorem 4 is that this wrapper is not responsible for enforcing the balanced requirement of the environment manually. He can rather rely on the protocol, which already enforces this requirement in our modified mapping. Hence,  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  also never has to generate any additional import himself but can rather take exactly the amount it gets from the environment.

The corresponding IITM simulator  $S_{IITM}^{\text{UC-bounded}}$  is again defined to internally simulate  $S_{UC}$  within the same wrapper as describe above for  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ . Observe that, just as  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ , the simulator  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  does not need to generate import itself but rather relies only on the import provided by the environment. Furthermore, by definition of our protocol mapping, the simulator is always activated immediately at the exact point in time when the environment provides new adversarial import, even when done via the message (provideAdversarialImport, i) to the protocol, which is precisely as in the UC model. Hence we can map a run in the IITM model to a run in the UC model (see below) without additionally requiring that op does not interact with the protocol.

Given a single session environment  $\mathcal{E}_{IITM}^{\text{single},\xi}$  in the IITM model that sends only inputs permitted by  $\xi$  and tries to distinguish  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}$  and  $\mathcal{S}_{UTM}^{\text{UC-bounded}} | \phi_{IITM}$ , we construct the corresponding UC environment  $\mathcal{E}_{UC}$  as follows.  $\mathcal{E}_{UC}$  runs the same logic as  $\mathcal{E}_{IITM}^{\text{single},\xi}$  but stops the run with empty overall output upon reaching a fixed runtime bound q in its currently held import. If  $\mathcal{E}_{IITM}$  wants to send the network message (registerAdversarialImport, i) via the adversary to the protocol, then  $\mathcal{E}_{UC}$  drops this message, which then activates  $\mathcal{E}_{IITM}$ . If  $\mathcal{E}_{IITM}$  wants to send the input (provideAdversarialImport, i) to the protocol, then  $\mathcal{E}_{UC}$  instead sends op(i) to the adversary on the network. Furthermore,  $\mathcal{E}_{UC}$  keeps track of the total amount of import *i* (successfully) provided by  $\mathcal{E}_{IITM}$  directly via inputs to the protocol and the total amount of import j provided by  $\mathcal{E}_{IITM}$  to the adversary (either directly via a network message or via a provideAdversarialImport message to the protocol). If  $\mathcal{E}_{IITM}$ wants to send an input containing import i' to the protocol, then  $\mathcal{E}_{UC}$  checks whether i+i' > j. That is, whether  $\mathcal{E}_{IITM}$  would violate the balanced requirement by sending that import. If so, then the message is instead dropped. Otherwise, i' is added to the total import *i* tracked by  $\mathcal{E}_{UC}$  and the input is delivered to the protocol. All other actions of  $\mathcal{E}_{UC}$  are just as for  $\mathcal{E}_{IITM}$ . We have that  $\mathcal{E}_{UC}$ meets the UC runtime notion and is balanced by construction.

Now compare runs of the real IITM and real UC worlds using the same randomness. As long as the runtime bound q of  $\mathcal{E}_{UC}$  is not met, both worlds behave perfectly identical with the same overall output, if any. In particular, whenever  $\mathcal{E}_{IITM}$ in the IITM world provides adversarial import *i* via provideAdversarialImport, this is then forwarded to  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  who internally calls op(i) of  $\mathcal{A}_{Dum,UC}$ . This is exactly the same as for  $\mathcal{E}_{UC}$  in the UC world. Similarly, both the IITM and UC worlds ignore and drop (registerAdversarialImport, i) messages from  $\mathcal{E}_{IITM}$ . While in the IITM world the IITM protocol internally tracks the adversarial import and drops inputs if they would violate the balanced requirement, in the UC world the same is instead done by  $\mathcal{E}_{UC}$ . Note in particular that the protocol in the IITM world and the environment  $\mathcal{E}_{UC}$  in the UC world have the same view on the amounts of protocol and adversarial imports. Hence they drop inputs at the same points of a run. Finally observe that, even though  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  additionally registers some adversarial import in the IITM protocol but the same operation is not performed (and not even available) in the UC world, this registration does not affect or change the behavior of the IITM protocol compared to the UC protocol except for changing the set of inputs from the IITM environment that are dropped due to violating the balanced requirement. In the UC world, this

aspect is already perfectly simulated by  $\mathcal{E}_{UC}$ . Hence, the protocols in both worlds also behave the same.

The ideal worlds can be compared in the same way to obtain a mapping of runs (using the same randomness) with the same behavior as long as the runtime bound q of  $\mathcal{E}_{UC}$  is not met. Observe in particular that adversarial import provided via provideAdversarialImport in the IITM world and the corresponding calls of op in the UC world occur at the same points in time in the run. More specifically, when comparing the states of the IITM and UC protocols the only difference in the IITM world is that the protocol additionally extends the list of adversarial imports. However, as argued above, this does not actually affect or change the behavior of the IITM protocol except for dropping a different set of inputs from the environment, which is simulated correctly by  $\mathcal{E}_{UC}$ . In particular, the behavior of the IITM protocol while receiving messages from the simulator following the call of op, if any such messages are sent, is identical to the UC world. So again both the UC and IITM worlds behave the same, even if the simulator decides to interact with the protocol upon receiving import via op.

The previous observation is actually the key difference between this variant of our protocol mapping and the simpler mapping presented in the main body of this paper. In this construction, we can argue that calls to op occur at the same points of a run in both the UC and IITM worlds such that the behavior of the respective protocols, if they were to receive a message from the simulator, is also the same. In contrast, in the proof of Theorem 4 for the simpler mapping we had a situation where calls to op in the UC and IITM worlds did not occur at precisely the same points in time. Instead, in the IITM world such a call would occur after a protocol has already received a message with import, thereby updating its state and potentially changing its behavior upon receiving a network message. In the UC world, the call to op would occur before the protocol receives the message with import. This is why we needed a specific class of simulators that do not interact with the protocol while receiving import via op. For such simulators, it does not matter whether the protocol receives its import before or after the call of op. We only had to ensure that the overall amount of import added via op is consistent in both the UC and IITM worlds between any pair of messages  $m, m' \neq op$  received by the simulator.

To complete this step of the proof, we only have to define the runtime bound q of  $\mathcal{E}_{UC}$  in such a way that, given sufficiently long external inputs,  $\mathcal{E}_{UC}$  does not actually reach the runtime bound. This can be done in the same manner as in the proof of Theorem 4.

**Indistinguishability for arbitrary single session environments.** This step does not change.

Indistinguishability for arbitrary adversaries. We have to construct a simulator  $S_{IITM}$  for the regular IITM dummy adversary  $\mathcal{A}_{Dum,IITM}$ , who does not receive any import himself. This dummy rather always forwards all messages independently of any import mechanism. The idea for constructing the corresponding simulator is still the same as in the proof of Theorem 4, but the technical details are slightly different. Namely, the simulator still internally generates its own

import in such a way that, if one were to run  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  with the import generated by the simulator instead of  $\mathcal{A}_{Dum,IITM}$  in the real world, then  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$ never reaches its runtime bound. As we will see below, due to the definition of **provideAdversarialImport**, we are able to construct the simulator in such a way that we can again perform a reduction without requiring any additional assumptions on the implementation of op in the simulator. In particular, the environment constructed for the reduction is able to provide import at the exact same points in time in the run as  $\mathcal{S}_{IITM}$  generates fresh import. In what follows, to improve readability we use the same terminology as in the third step of Appendix B, i.e., when we say that additional import *i* is generated by the wrapper of the simulator (via a call to **provideAdversarialImport**), then this implicitly also includes sufficient import to account for the logarithmic runtime reduction due to a call of op such that at least import *i* remains unused.

Let p be the polynomial in the import currently held by the real protocol that upper bounds the total runtime of the codes of  $\pi_{UC}$ ; w.l.o.g. choose p monotonically increasing such that  $p(x) \geq x$ . We define  $S_{IITM}$  to internally simulate  $S_{IITM}^{UC-bounded}$  within a wrapper layer. Upon its first activation with some message m, the wrapper of  $S_{IITM}$  provides  $\eta$  import to  $S_{IITM}^{\text{UC-bounded}}$  via provideAdversarialImport as if received from the ideal protocol. Since this internally calls op,  $S_{IITM}^{UC-bounded}$  should at some point output an acknowledgement for the environment that op was executed. As soon as the wrapper of  $\mathcal{S}_{IITM}$  receives the next such notification, it drops it and rather continues processing the initial message m as it would for any other messages. When  $S_{IITM}$  receives a network message (registerAdversarialImport, i) for  $M_{msg}$ , then the wrapper of  $S_{IITM}$  first forwards this message to the ideal protocol to register import i, waits for the response, and then provides d import to  $\mathcal{S}_{IITM}^{\text{UC-bounded}}$  via provideAdversarialImport, where d := p(j+i) - p(j) and j is the total amount of import provided by the environment so far in network messages registerAdversarialImport, inputs provideAdversarialImport, and forwarded via regular network messages. Again, this internally calls op. Hence, when the wrapper of  $S_{IITM}$  receives the next acknowledgement for op from  $S_{IITM}^{\text{UC-bounded}}$ , the wrapper instead sends (registerAdversarialImport,ok) to the environment. That is, the environment receives an acknowledgement to the original (registerAdversarialImport, i) request, just as for the dummy adversary  $\mathcal{A}_{Dum,IITM}$  in the real world. Whenever  $\mathcal{S}_{IITM}$  receives any other network message m from the environment,  $S_{IITM}$  first computes d := p(j+i) - p(j) with j as above and where i is the protocol import that would be forwarded as part of m.  $\mathcal{S}_{IITM}$  then provides  $2 \cdot |m| + d$  import to the internal simulator via a message provideAdversarialImport and afterwards, as soon as he receives the next notification for op from the internal simulator, forwards m to  $\mathcal{S}_{UTM}^{\text{UC-bounded}}$ . Finally, whenever the wrapper of  $S_{IITM}$  receives a (provideAdversarialImport, i) message from the ideal protocol, it internally provides d import to  $S_{IITM}^{\text{UC-bounded}}$ via provideAdversarialImport where again d := p(j + i) - p(j) and j are as defined above. Afterwards, as soon as  $S_{IITM}^{\text{UC-bounded}}$  wants to send the next acknowledgement to a call of op to the environment, the wrapper of  $S_{IITM}$  instead

sends (provideAdversarialImport, i). In other words, the simulator forwards the original provideAdversarialImport message just as the dummy  $\mathcal{A}_{Dum,IITM}$  does in the real world. In all other cases,  $\mathcal{S}_{IITM}$  simply forwards the messages between  $\mathcal{S}_{IITM}^{UC-bounded}$  and the protocol/environment.

Let  $\mathcal{E}_{IITM}^{\text{single}}$  be a single session environment that tries to distinguish the systems  $\mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$  and  $\mathcal{S}_{IITM} | \phi_{IITM}^{\xi\text{-id}}$ . The environment  $\mathcal{E}'_{IITM}^{\text{single}}$  for the reduction (to the case of indistinguishability of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \pi_{IITM}^{\xi\text{-id}}$  and  $S_{IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi \cdot \text{id}} )$  is defined similarly to the one constructed in the proof of Theorem 4. That is,  $\mathcal{E}'_{IITM}^{\text{single}}$  internally simulates  $\mathcal{E}_{IITM}^{\text{single}}$  and the wrapper portion of  $S_{IITM}$ . Additionally,  $\mathcal{E}'_{IITM}^{\text{single}}$  keeps track of protocol and adversarial imports (via provideAdversarialImport and registerAdversarialImport) and drops inputs if  $\mathcal{E}_{IITM}^{\text{single}}$  did not provide sufficient adversarial import. More formally,  $\mathcal{E}'_{IITM}^{\text{single}}$  internally runs  $\mathcal{E}_{IITM}^{\text{single}}$  but with the following modifications. As soon as  $\mathcal{E}_{IITM}^{\text{single}}$  wants to send its first network message or send an input of type provideAdversarialImport to the protocol,  $\mathcal{E}'_{IITM}^{\text{single}}$  first provides  $\eta$ import to the adversary via an input provideAdversarialImport to the protocol. Afterwards,  $\mathcal{E}'_{IITM}^{\text{single}}$  drops the next notification for an op operation from the adversary and instead processes the original message sent by  $\mathcal{E}_{IITM}^{\text{single}}$  as for any other messages from  $\mathcal{E}_{IITM}^{\text{single}}$ . During the run, if  $\mathcal{E}_{IITM}^{\text{single}}$  wants to send the network message (registerAdversarialImport, i) to  $M_{msg}$ ,  $\mathcal{E}'_{IITM}^{single}$  instead provides d import via an input provideAdversarialImport to the protocol, where d := p(j+i) - p(j) and j is the total import that  $\mathcal{E}_{IITM}^{\text{single}}$  has previously provided via registerAdversarialImport network messages, provideAdversarialImport inputs, and forwarded as part of network messages m. Then, once  $\mathcal{E}'_{IITM}^{\text{single}}$  receives the next notification for an *op* operation,  $\mathcal{E}'_{IITM}^{single}$  activates  $\mathcal{E}_{IITM}^{single}$  with network message (registerAdversarialImport, ok). If  $\mathcal{E}_{IITM}^{single}$  wants to send any other network message m, then  $\mathcal{E}'_{IITM}^{\text{single}}$  first provides  $2 \cdot |m| + d$  (with d computed as above) adversarial import via an input provideAdversarialImport and then, upon receiving the next notification for op, sends m to the adversary. Whenever  $\mathcal{E}_{IITM}^{\text{single}}$  wants to send the input (provideAdversarialImport, i) to the proto- $\operatorname{col}, \mathcal{E}'_{IITM}^{\text{single}}$  instead provides import d via an input provideAdversarialImport where d := p(j + i) - p(j) and j are as above. Then, once  $\mathcal{E}'_{IITM}^{\text{single}}$  receives the next *op* notification,  $\mathcal{E}'_{IITM}^{\text{single}}$  activates  $\mathcal{E}_{IITM}^{\text{single}}$  with the network message (provideAdversarialImport, i). Finally,  $\mathcal{E}'_{IITM}^{\text{single}}$  tracks the total amount of adversarial import  $i_{adv}$  provided by  $\mathcal{E}_{IITM}^{single}$  via provideAdversarialImport and registerAdversarialImport as well as the total protocol import  $i_{\text{prot}}$  provided directly to the protocol in inputs from  $\mathcal{E}_{IITM}^{\text{single}}$ . If  $\mathcal{E}_{IITM}^{\text{single}}$  wants to send an input containing protocol import such that  $i_{\text{prot}} > i_{\text{adv}}$ , then  $\mathcal{E}'_{IITM}^{\text{single}}$  instead drops this message (without increasing the counter  $i_{\text{prot}}$ ), which in turn activates  $\mathcal{E}_{IITM}^{\text{single}}$ with empty input by definition of the IITM model.

Now consider runs of the real world  $\mathcal{E}_{IITM}^{\text{single}} | \mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$  and the real world  $\mathcal{E}'_{IITM}^{\text{single}} | \mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} | \phi_{IITM}^{\xi\text{-id}}$  with the same randomness. Observe that these

runs behave identical and have the same overall output as long as the runtime bound of  $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}}$  is not reached. In particular, the additional  $\eta$  adversarial import provided by  $\mathcal{E}'_{IITM}^{\text{single}}$  at the beginning does not change the behavior of the protocol except for increasing the total amount of adversarial import stored in the protocol. This in turn only increases the set of inputs from the environment that the protocol running with  $\mathcal{E}'_{IITM}^{\text{single}}$  accepts. As  $\mathcal{E}'_{IITM}^{\text{single}}$  already drops inputs itself if they would not be accepted by the protocol running with  $\mathcal{E}_{IITM}$ , this aspect is simulated correctly.  $\mathcal{E}'_{IITM}^{single}$  further simulates the effect of (registerAdversarialImport, i) correctly by instead providing d import using an input provideAdversarialImport, which also increases the amount of adversarial import stored in the protocol. The only difference here is the exact amount of adversarial import registered in the protocol. By definition of d and since  $p(x) \ge x$ , we have that  $\mathcal{E}'_{IITM}^{\text{single}}$  provides at least the same amount of import as  $\mathcal{E}_{IITM}$ , i.e., at most increases the set of inputs accepted by the protocol. As argued above, since  $\mathcal{E}'_{IITM}^{\text{single}}$  itself already correctly simulates dropped inputs, this increase in adversarial imports stored in the protocol does not change the behavior of the run. The same argument holds for (provideAdversarialImport, i) inputs by  $\mathcal{E}_{IITM}^{\text{single}}$ , where, again, the overall import provided by  $\mathcal{E}'_{IITM}^{\text{single}}$  might be larger but is at least the same, thereby not changing the overall behavior of the run. Finally, just as in the proof of Theorem 4, the construction of  $\mathcal{E}'_{IITM}^{single}$  is such that  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  never reaches its runtime bound.<sup>27</sup> Hence, both  $\mathcal{E}_{IITM}^{single} | \mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi \cdot id}$  and  $\mathcal{E}'_{IITM}^{single} | \mathcal{A}_{Dum,IITM}^{UC-bounded} | \phi_{IITM}^{\xi \cdot id}$  have the same overall output distribution.

Now consider runs of the ideal world  $\mathcal{E}_{IITM}^{single} | \mathcal{S}_{IITM} | \phi_{IITM}^{\xi\text{-id}}$  and the ideal world  $\mathcal{E}'_{IITM}^{single} | \mathcal{S}_{IITM}^{UC-bounded} | \phi_{IITM}^{\xi\text{-id}}$  with the same randomness. First observe that the initial import  $\eta$  is generated by  $\mathcal{S}_{IITM}$  at the same point in the run as  $\mathcal{E}'_{IITM}^{single}$  calls (provideAdversarialImport,  $\eta$ ). This is because the first activation of  $\mathcal{S}_{IITM}$  is either due to a network message by  $\mathcal{E}_{IITM}^{single}$  or due to an provideAdversarialImport message sent by  $\mathcal{E}_{IITM}^{single}$  via the protocol. Indeed, even if  $\mathcal{E}_{IITM}^{single}$  first sends several inputs  $m \neq \text{provideAdversarialImport}$  to the protocol, then these inputs are either dropped since no adversarialimport has been provided/registered yet or they do not contain any import. But by definition of the UC protocol, the machine codes c do not start running until they have received at least  $\eta$  import. Hence, the protocol will never be the one to send the first message to the simulator, except if activated by an provideAdversarialImport input. This is precisely as simulated by  $\mathcal{E}'_{IITM}^{single}$ . During the remainder of the run, both worlds behave identical. In particular, observe that  $\mathcal{S}_{IITM}$  registers exactly the amount of import provided by the environment via registerAdversarialImport network messages but not any more. Hence, the ideal protocol running with  $\mathcal{S}_{IITM}$  drops inputs iff  $\mathcal{E}'_{IITM}^{single}$  formally provides an overall larger amount of

<sup>&</sup>lt;sup>27</sup> Here we use that the total amount of adversarial imports provided by  $\mathcal{E}_{IITM}$  via inputs provideAdversarialImport and network messages registerAdversarialImport is an upper bound for the import directly provided via inputs to the real protocol.

adversarial import to  $\phi_{IITM}^{\xi\text{-id}}$  compared to  $\mathcal{E}_{IITM}^{\text{single}}$ . As already explained for the real worlds above, this only increases the set of inputs that are accepted by the protocol, which  $\mathcal{E}'_{IITM}^{\text{single}}$  already simulates correctly itself. So the ideal worlds also behave identical with the same overall outputs.

Indistinguishability for multiple sessions. This step does not change.

# H.3 Adjusting Theorem 5 and its proof

In the theorem we can drop the requirement that the core of the simulator does not call totalImport? since this request does not exist in our variant of the mapping. We do not need to instead require that the core does not call the new operation registerAdversarialImport. This can rather be concluded from the fact that real and ideal world are indistinguishable. Specifically, the wrapper of the simulator already keeps the amount of adversarial import stored in the ideal protocol identical to the amount of adversarial import stored in the real protocol. Hence, any simulator whose core calls registerAdversarialImport will cause the real and ideal world to be easily distinguishable: the ideal world then has more adversarial import, which the environment can check by providing a suitable amount of protocol import. Hence, we can conclude that the core does not use the new command registerAdversarialImport that is available only in the IITM protocol.

We can therefore directly use the core of the simulator in the UC model. The proof of the theorem then stays essentially the same. Note in particular that an IITM environment can simply simulate a distinguishing UC environment. Since the UC environment is balanced, it will always provide at least as much import directly to the adversary (via network messages) as it provides directly via inputs to the protocol. By definition of the wrappers of  $\mathcal{A}_{Dum,IITM}^{UC-bounded}$  and  $\mathcal{S}_{IITM}^{UC-bounded}$ , this adversarial import is registered in the real and ideal protocols, respectively. Hence, the real and ideal protocols in the IITM model always have at least as much adversarial import as protocol import, i.e., do not drop any messages. This precisely matches the behavior in the UC model.

# H.4 Composition

Suppose we have  $\pi_{UC}$ ,  $\phi_{UC}$ ,  $\rho_{UC}$  UC protocols such that  $\pi_{UC} \leq_{UC}^{\xi} \phi_{UC}$  and the UC composition theorem can be applied to  $\rho_{UC}$  to obtain  $\rho_{UC}^{\phi \to \pi} \leq_{UC} \rho_{UC}$ . Observe that, just as for Corollary 1, we can apply the variant of our protocol mapping directly to  $\rho_{UC}^{\phi \to \pi}$  and  $\rho_{UC}$  to obtain protocols  $\rho_{IITM}^{\phi \to \pi}$  and  $\rho_{IITM}$ . Note that this involves, among others, mapping the UC composition shell code added by the UC composition theorem to be able to express the meaning of  $\rho_{IITM}^{\phi \to \pi}$ . By the above result,  $\rho_{UC}^{\phi \to \pi} \leq_{UC} \rho_{UC}$  then already implies  $\rho_{IITM}^{\phi \to \pi} \leq_{IITM} \rho_{IITM}$ . In particular, we do not even have to check that the simulator constructed for the UC composition theorem meets certain properties since this variant of our mapping works for arbitrary simulators.

Now consider the case that we want to obtain the same statement directly via the composition theorem of the IITM model (which also does not require mapping the UC composition shell code) or want to compose some arbitrary higher-level IITM protocol  $\rho_{IITM}$  with  $\pi_{IITM}/\phi_{IITM}$ , where  $\rho_{IITM}$  does not have a counter part in the UC model. This is still possible for this variant of our mapping by modeling  $\rho_{IITM}$  in the same way as described in 3.4. That is, we first map  $\phi_{IITM}$  to the IITM model independently of any higher-level protocols  $\rho$ .<sup>28</sup> The higher-level machines of a protocol  $\rho_{IITM}$  then connect via a single tape to the machines of  $\phi_{IITM}$ , potentially using a multiplexer. Note that, since  $\phi_{IITM}$  is mapped independently of the definition of  $\rho_{IITM}$  (if  $\rho_{IITM}$ even has a UC counterpart), it includes the adversarial import mechanism of this variant of our mapping. Hence, the higher-level protocol  $\rho_{IITM}$  must be defined in a way that it first provides the subroutine sessions of  $\phi_{IITM}$  with sufficient adversarial import via provideAdversarialImport before  $\rho_{IITM}$  can provide protocol import and hence use these sessions. Defining a higher-level protocol  $\rho_{IITM}$  in this way should always be possible via syntactical changes that do not affect its semantics. For example,  $\rho_{IITM}$  can simply provide the same adversarial import that it has received also to its subroutine. Alternatively, whenever  $\rho_{IITM}$ wants to provide new protocol import i to subroutine  $\phi_{IITM}$ ,  $\rho_{IITM}$  can first call (provideAdversarialImport, j),  $j \ge i$ . The adversarial import j provided by  $\rho_{IITM}$  to the subroutine  $\phi_{IITM}$  is actually arbitrary, as long as it is larger than the protocol import provided (to ensure that the subroutine works as expected) and still of polynomial length (to ensure polynomial runtime of the overall protocol).

So it is possible to model higher-level protocols  $\rho$  in a manner such that they are compatible with the IITM composition theorem. However, this variant of our mapping still introduces quite a bit of additional technical overhead to the protocol definition of such higher-level protocols. This is one of the main reasons why we rather presented our simpler mapping in the main body, which should already cover basically all protocols of practical interest while yielding less technical, more natural definitions of higher-level protocols.

# I Single Session Security Analysis in the IITM model

In this section we first recall the security definition and second composition theorem of the IITM model [20] which enable a single session security analysis of protocols with disjoint sessions. We then provide full details of the last step of the proof of Theorem 4 by showing that the second composition theorem indeed applies.

<sup>&</sup>lt;sup>28</sup> Note that this is different from the above, where  $\rho_{IITM}^{\phi \to \pi}$  and  $\rho_{IITM}$  are obtained by mapping a specific higher-level protocol as well as all of its subroutines, including  $\phi_{IITM}$ .

# I.1 Recalling the Second IITM Composition Theorem

We first have to formalize the meaning of a "protocol session". We consider a (polynomially computable) protocol session identifier function  $\sigma$  which, given a message, a tape name, and the direction of the message on that tape, outputs a protocol session identifier PSID (a bit string) or  $\perp$ .<sup>29</sup> For example, the following function takes the prefix of a message as its PSID:  $\sigma_{\text{prefix}}(m, t, d) := s$  if m = (s, m') for some s, m' and  $\sigma_{\text{prefix}}(m, t, d) := \perp$  otherwise, for all m, t, d. The reason that  $\sigma$ , besides a message, also takes a tape name and direction as input is that the way PSIDs are extracted from messages may depend on the tape and the recipient machine of a message.<sup>30</sup>

Now, we say that an IITM M is a  $\sigma$ -session machine (or a  $\sigma$ -session version) if the following conditions are satisfied: (i) M rejects (in mode CheckAddress) a message m on tape t with direction d if  $\sigma(m, t, d) = \bot$ . (ii) If  $m_0$  is the first message that M accepted (in mode CheckAddress), say on tape  $t_0$  with direction  $d_0$ , in a run, then, M will reject all messages m received on some tape t and some direction d (in mode CheckAddress) with  $\sigma(m, t, d) \neq \sigma(m_0, t_0, d_0)$ . (iii) Whenever M outputs a messages m on tape t with direction d (in mode Compute), then  $\sigma(m, t, d) = \sigma(m_0, t_0, d_0)$ , with  $m_0, t_0, d_0$  as before. We say that a system Q is a  $\sigma$ -session system (or a  $\sigma$ -session version) if every IITM occurring in Q is a  $\sigma$ -session machine.

We call an environment  $\mathcal{E} \sigma$ -single session if it only outputs messages with the same SID according to  $\sigma$ . Hence, when interacting with a  $\sigma$ -session version, such an environmental system invokes at most one protocol session.

Let  $\mathcal{P}$  and  $\mathcal{F}$  be protocol systems, which in the setting considered here would typically describe multiple sessions of a protocol. Moreover, we assume that  $\mathcal{P}$ and  $\mathcal{F}$  are  $\sigma$ -session versions, i.e., both protocols have disjoint sessions that do not interact with each other according to  $\sigma$ . Now, we define what it means that a single session of  $\mathcal{P}$  realizes a single session of  $\mathcal{F}$ . This is defined just as  $\mathcal{P} \leq_{IITM} \mathcal{F}$ , with the difference that we consider only  $\sigma$ -single session environments, and hence, environments that invoke at most one session of  $\mathcal{P}$  and  $\mathcal{F}$ .

**Definition 4.** Let  $\mathcal{P}$ ,  $\mathcal{F}$  be  $\sigma$  be as above. Then,  $\mathcal{P}$  single-session realizes  $\mathcal{F}$  w.r.t.  $\sigma$  ( $\mathcal{P} \leq_{IITM}^{\sigma\text{-single}} \mathcal{F}$ ) if and only if there exists an adversary  $\mathcal{S}$  (a simulator or an ideal adversary) such that  $\mathcal{E} | \mathcal{P} \equiv \mathcal{E} | \mathcal{S} | \mathcal{F}$  for every  $\sigma$ -single session

<sup>&</sup>lt;sup>29</sup> In [20] a PSID was just called SID. Here we use a different term to avoid confusion with the SIDs used by UC and our mapped protocols. While both are related, as we will show later in the section, a PSID uniquely identifies the whole protocol session whereas there might be multiple different SIDs used by instances within that session.

<sup>&</sup>lt;sup>30</sup> The definition of protocol session functions given in [20] does not include a direction d. This is because the basic IITM model is defined on unidirectional tapes, so a tape name already uniquely identifies the message direction. Since here we consider named bidirectional tapes, which are implemented via pairs of uniquely named unidirectional tapes, we additionally add the direction d to the function input. This is easily seen to be a special case of the IITM definition due to the way bidirectional tapes are implemented.

environment  $\mathcal{E}$ . (the details regarding interfaces and runtime are analogous to Definition 2)

Now, the following second IITM composition theorem says that if  $\mathcal{P}$  realizes  $\mathcal{F}$  w.r.t. a single session, then  $\mathcal{P}$  realizes  $\mathcal{F}$  w.r.t. multiple sessions. As mentioned before, in the setting considered here  $\mathcal{P}$  and  $\mathcal{F}$  would typically model multi-session versions of a protocol/functionality.

**Theorem 6 (Unbounded Self Composition).** Let  $\sigma$ ,  $\mathcal{P}$ , and  $\mathcal{F}$  be as above. Then,  $\mathcal{P} \leq_{IITM}^{\sigma\text{-single}} \mathcal{F}$  implies  $\mathcal{P} \leq_{IITM} \mathcal{F}$ .

## I.2 Full Details of the Last Step of the Proof of Theorem 4

Recall that in the last step of Theorem 4 we claimed that the second composition theorem, i.e., Theorem 6 is applicable and implies the overall statement. We now formalize this statement using the terminology and definitions from above.

The previous steps of the proof showed that there is a simulator  $S_{IITM}$  such that  $\mathcal{A}_{Dum,IITM} | \pi_{IITM}^{\xi\text{-id}}$  and  $S_{IITM} | \phi_{IITM}^{\xi\text{-id}}$  cannot be distinguished by any single session environment  $\mathcal{E}_{IITM}^{\text{single}}$ . By Lemma 5 from [20] the IITM dummy adversary  $\mathcal{A}_{Dum,IITM}$  can be added or dropped from a system without changing its behavior. Hence, we also have that  $\pi_{IITM}^{\xi\text{-id}}$  and  $S_{IITM} | \phi_{IITM}^{\xi\text{-id}}$  cannot be distinguished by any single session environment  $\mathcal{E}_{IITM}^{\text{single}}$ . Next, we formally define the protocol session identifier function  $\sigma$  which in turn also formalizes the meaning of a single session environment.

Recall that, by our normalization from Section 3.2, the protocols  $\pi/\phi$  use the standard subroutine respecting wrapper mechanism of the UC model. Hence instances running the highest-level code  $c_{\pi}/c_{\phi}$  have an SID  $sid_c$  with instances running any other codes have SIDs of the form  $(sid_c, sid')$ , where  $sid_c$  is the SID of the highest-level instances of that protocol session. Furthermore, the subroutine respecting code guarantees that instances belonging to a session  $sid_c$  send messages only to other instances sharing this  $sid_c$ , except for outputs provided by the highest-level instances running  $c_{\pi}/c_{\phi}$  to the environment. So the idea for defining a protocol session identifier is to use  $sid_c$ , which is also included in the headers of all network messages.

More formally, we define  $\sigma(m, t, d)$  as follows. If m is an input from the environment (via  $M_{identity}^{\xi}$ ) to  $c_{\pi}/c_{\phi}$  (which can be determined from t and d), then extract and output  $sid_c$  from the receiver ID contained in the header of m. Conversely, if m is an output from  $c_{\pi}/c_{\phi}$  (via  $M_{identity}^{\xi}$ ) to the environment, then extract and output  $sid_c$  from the sender ID contained in the header of m. If m is received or sent by a protocol machine on the network, extract and output  $sid_c$  from the receiver or sender ID contained in the header of m. Similarly, if m is on an internal I/O tape between protocol machines, then extract and output  $sid_c$  from the receiver contained in the header of m. If m does not match the expected format of messages on a certain tape, e.g., does not contain a well-formed header or the header contains the wrong machine code, then we set  $\sigma(m, t, d) := \bot$ .

Observe that, given this definition of  $\sigma$ , the protocols  $\pi_{IITM}^{\xi\text{-id}}$  and  $\phi_{IITM}^{\xi\text{-id}}$  are indeed  $\sigma$  session versions. Condition (i) follows by the definition of CheckAddress which rejects messages that do not have a certain expected format, including header. This is exactly the type of messages where  $\sigma$  returns  $\perp$ . For (ii), observe that all instances of  $\pi_{IITM}^{\xi\text{-id}}$  and  $\phi_{IITM}^{\xi\text{-id}}$ , after accepting their first message, store their SID and then accept future messages only if they are for the same SID.<sup>31</sup> Since this SID contains, potentially as a prefix, the PSID returned by  $\sigma$ , this implies (ii). (iii) is obviously fulfilled for external tapes where the PSID is extracted from the sender ID. For internal tapes between machines  $M_c$  running some UC code c, where the PSID is extracted from the receiver, observe that the definition of the subroutine respecting shell code ensures that the  $sid_c$  portion of both the sender and receiver SIDs is the same (since messages on such tapes are sent between two instances within the same session). For internal tapes between a machine  $M_c$  and the machines  $M_{bc}$  and  $M_{identity}^{\xi}$ , this follows by the definition of their compute modes (as well as the subroutine respecting property of  $M_c$ ) which also always use the same  $sid_c$  for addressing. Hence (iii) is also fulfilled.

which also always use the same  $sid_c$  for addressing. Hence (iii) is also fulfilled. Altogether we have that  $\pi_{IITM}^{\xi \cdot id}$  and  $\phi_{IITM}^{\xi \cdot id}$  are  $\sigma$ -session versions and there exists a simulator  $S_{IITM}$  such that  $\pi_{IITM}^{\xi \cdot id}$  and  $S_{IITM} | \phi_{IITM}^{\xi \cdot id}$  cannot be distinguished by any  $\sigma$ -single session environment  $\mathcal{E}_{IITM}^{\text{single}}$ . This is exactly Definition 4, i.e., we have  $\pi_{IITM}^{\xi \cdot id} \leq_{IITM} \phi_{IITM}^{\xi \cdot id}$ . By Theorem 6, this implies  $\pi_{IITM}^{\xi \cdot id} \leq_{IITM} \phi_{IITM}^{\xi \cdot id}$  as claimed.

# J Discussion of Technical Details of the UC Model

In this section, we provide more in depth discussion of certain technical details of the UC model [7]. This includes full details for issues that we found while constructing our mapping and which, among others, cause the composition theorem to fail.

# J.1 Adversaries Revealing Their Identity

As highlighted in Footnote 5, while the adversary is restricted to use only nonforced write commands he is not restricted in whether he wants to reveal its extended identity, including its full code, to the receiver (cf. the definition of the security experiment given on Page 39 and summarized in Figure 6 of [7]). Observe that this code is "authenticated" by the computational framework of the UC model. That is, if a protocol receives such a revealed extended identity, then

<sup>&</sup>lt;sup>31</sup> A slight exception is  $M_{\rm identity}^{\xi}$ , which accepts messages from the subroutine protocol if they contain the same sender SID as used by that instance of  $M_{\rm identity}^{\xi}$ . For messages from the environment,  $M_{\rm identity}^{\xi}$  works just as other machines, i.e., accepts them only if the receiver SID is the same. Note that this is a technical detail that does not actually affect the following argument. In particular, this definition is consistent with the way  $\sigma$  extracts PSIDs from the messages in different directions between protocol and environment.

the computational framework guarantees that the code contained in that identity is the same as the code that the sender (i.e., attacker) is actually running. This appears to be unintended. Indeed, previous versions of the UC model, such as the one from 2013 [5], removed the code of the adversary from any messages sent to a backdoor tape.

As it turns out, preventing the adversary from revealing its own code in an authenticated fashion is actually essential to obtain a meaningful model that supports composition. Specifically, adversaries that can reveal their authenticated code cause the following three major issues in the current version of the UC model:

1. The dummy adversary is incomplete. The dummy adversary cannot be used to simulate arbitrary other adversaries, i.e., Claim 11 of [7] is false. This is because the dummy adversary is only ever able to reveal its own code to the protocol. However, an attacker revealing a different code might actually be able to cause the real world to behave in a manner that is distinguishable from the ideal world.

It is quite simple to come up with such protocols. For example, consider the ideal world protocol  $\phi$  that does nothing, i.e., drops all inputs and network messages. Define the real protocol  $\pi$  to behave as  $\phi$  but  $\pi$  actually processes network messages from the adversary. If the network message does not contain an authenticated identity or the authenticated identity contains the code of the dummy attacker, then  $\pi$  drops the network message just as  $\phi$  does. Otherwise,  $\pi$ sends output **real** directly to the environment.

It is easily seen that  $\pi$  running with the dummy attacker is indistinguishable from  $\phi$  running with the dummy attacker (which acts as the simulator for  $\phi$ ). But clearly  $\pi$  running with some other attacker in the real world can easily be distinguished from  $\phi$  running with any simulator in the ideal world.

Since completeness of the dummy adversary is used in an essential way in the proof of the UC composition theorem, this also invalidates the proof of that theorem.

2. The UC composition theorem does not hold true. Above we already showed that the proof the composition theorem (Theorem 22 in [7]) is false. But an even stronger statement is the case: the composition theorem itself is false and cannot be shown at all. The underlying reason is that the behavior of the ideal protocol  $\phi$  might depend on the authenticated code of the simulator. However, obtaining a composition theorem requires constructing a new simulator that not only handles network messages from  $\phi$  but also from a higher-level protocol  $\rho$ . Since this changes the code of the simulator, the behavior of  $\phi$  might also change, in which case it might no longer be indistinguishable from its realization.

Again, it is relatively simple to come up with such protocols. For example, consider the real protocol  $\pi$  that, upon receiving the first input from the environment, returns **real** and otherwise drops all other messages. We define  $\phi$  to forward the first input from the environment to the simulator. If the response contains the authenticated code of the simulator and that code equals a certain fixed code  $c_{\rm sim}$ , then  $\phi$  outputs **real** to the environment. In all other cases,  $\phi$ 

drops the message and stops. In particular, if  $\phi$  receives a second input by the environment, then just as  $\pi$  it drops that input. We define the simulator code  $c_{\rm sim}$  to internally simulate the dummy adversary towards the environment. If the dummy adversary wants to send a message to the real protocol, then the simulator simply drops the message. If the simulator receives the first message from an instance of the ideal protocol, then the simulator returns that message while revealing its own extended identity. All subsequent messages from any protocol instances are dropped.

By construction, we have that  $\pi$  running with the dummy adversary is indistinguishable from  $\phi$  running with the simulator with code  $c_{\rm sim}$ . Next, we construct a protocol  $\rho$  using  $\phi$  such that the composed protocol  $\rho^{\phi \to \pi}$  is distinguishable for all simulators. Upon receiving its first input from the environment,  $\rho$  forwards this input to the subroutine  $\phi$  and returns the output, if any. Afterwards, all further inputs from the environment are forwarded to the adversary on the network.

There is no simulator S such that  $\rho^{\phi \to \pi}$  running with the dummy adversary is indistinguishable from  $\rho$  running with S, which shows that the composition theorem is false. This can be seen as follows: observe that the simulator with code  $c_{\rm sim}$  is not suitable. While both worlds are indistinguishable as long as the environment sends only a single input, they behave differently in the case of a second input. Namely, in the real world the protocol  $\rho^{\phi \to \pi}$  forwards the input via the dummy adversary to the environment. In contrast, in the ideal world protocol  $\rho$  this input is forwarded to the simulator running code  $z_{\rm sim}$ , which then drops the message. Further observe that any simulator running code  $\neq c_{\rm sim}$ already causes distinguishable behavior upon the first input. Namely, the real world protocol  $\rho^{\phi \to \pi}$  outputs **real** upon receiving the first input. The same is impossible to accomplish in the ideal world protocol  $\rho$  if the simulator runs code  $\neq c_{\rm sim}$  by construction of  $\phi$ .

**3.** The realization relation can be shown only for few special cases. For practically all pairs of protocols  $\pi$ ,  $\phi$  it is impossible to find a simulator that proves the realization relation (cf. Definition 1). The only exception are protocols  $\pi$ ,  $\phi$  where the simulator does not need to perform any actions, i.e., is the same as the adversary in the real world.

The reason for this is that the adversary can not only reveal its code to the protocol but also to the environment. Since the realization relation quantifies over all adversaries in the real world, this includes adversaries  $\mathcal{A}$  that reveal their authenticated code to the environment. To be able to simulate such an attacker, the simulator needs to run the same code, i.e., be the same as  $\mathcal{A}$ .

**Our fix.** We can fix all of the above issues by disallowing the adversary from revealing its own identity, which, looking at previous versions of the UC model, appears to be the intended behavior. This also matches how existing protocols from the UC literature have so far been defined and analyzed, i.e., this fix also retroactively applies to existing works. Alternatively, one could allow an adversary to freely decide upon the code that is revealed as part of its sender extended identity, i.e., remove the authentication of the adversarial code just already done

for environments. All of our results, including our protocol mapping, carry over to this case.

# J.2 Predicates for Non-Forced Writes

Recall that the non-forced write mechanism defined by the UC model allows a machine to specify the receiver of a message via a predicate, where the message is then delivered to the first machine instance in chronological order that matches the predicate (cf. Pages 23f in [7]). The model states that M', which is supposed to be an extended recipient identity, is interpreted as a predicate P on extended identities. While the paper does not explicitly state how exactly this interpretation works, the statement "[for messages from the protocol to the adversary] the forced-write flag must be unset, and the recipient code must not be specified" (Page 40 of [7]) given later in the paper suggests that the predicates used by protocols are interpreted as follows: A sender specifies parts of the receiver extended identity. The corresponding predicate then is the predicate that matches an extended identity iff that identity contains the same parts. Observe that predicates of this form run in linear time in their input (for a single fixed polynomial) as claimed in the proof of Lemma 2.

We also note that the above interpretation is not only heavily implied. More general ways to interpret M' as a predicate P, say, by running the code contained in M' to compute an indicator function on extended identities, would not allow for showing the composition theorem without first imposing some additional requirements. In particular, a protocol must not be able to specify a predicate P that would allow the protocol to perform exponential work in the security parameter. Such predicates would not be simulatable within an environment and hence the composition theorem would break down.

As for adversaries, they must specify the full extended ID of the intended recipient for non-forced writes to the protocol, i.e., they may not use predicates which could match multiple different extended IDs for sending network messages to the protocol. While this natural property of adversaries is not explicitly highlighted in the definition of the model of protocol execution, it is used, among others, to prove the UC composition theorem. For example, the simulator constructed for the composition theorem (cf. Figure 10 in [7]) requires incoming network messages to specify the exact extended identity of the intended protocol recipient. This full extended ID then enables the composition simulator to identify the target protocol session of the network message via calls to the directory machines of each existing session and by checking whether the extended ID belongs to a main party of a new session. This would be impossible to figure out for the simulator if only some parts of the extended identity were given, i.e., if it were a predicate that might match multiple different extended IDs possibly belonging to different protocol sessions.

# J.3 *ξ*-Identity Bounded Environments

Recall that the UC model defines the set of permitted sender identities  $\xi$  via an arbitrary (polytime) predicate that can be taken over the entire configuration of the whole system (cf. Page 40 of [7]), where a configuration of a system consists of the environment, the adversary, and all protocol instances. Observe that (a configuration of) the system considered for showing that  $\pi$  realizes  $\phi$  is drastically different from the system for showing that  $\rho^{\phi \to \pi}$  realizes  $\rho$ . For example, in the latter case there are instances running code  $\rho$ , additional higher-level instances that are not part of  $\pi/\phi$ , possibly not just a single but rather multiple sessions of  $\pi/\phi$ , and even different adversaries and environments (e.g., the simulator constructed by the composition theorem has a different machine code). Hence, even if we know that in runs of  $\rho$  all inputs provided to  $\phi$  meet the predicate  $\xi$ , this does not imply that the same is still the case if  $\rho$  is internally simulated by an environment running with  $\phi$ . But this is crucial for obtaining the composition theorem (Theorem 22 in [7]) is false for this general definition of  $\xi$ .

Consider the following counter example for the composition theorem: Upon receiving the first input, the real protocol  $\pi$  outputs real. In the same situation,  $\phi$  instead outputs ideal. All other messages are ignored. Let  $\rho$  be the dummy protocol that forwards inputs from the environment to  $\phi$  and returns outputs from  $\phi$  to the environment. Let  $\xi$  be the predicate that forbids all sender identities if there is no protocol instance running code  $\rho$  in the current configuration of the system but otherwise allows all sender identities. Observe that  $\pi \leq_{UC}^{\xi} \phi$ for the simulator that is the dummy adversary. This is because a  $\xi$ -identity bounded environment may not provide any inputs to the protocol until there is an instance running code  $\rho$ . But providing an input is the only way for the environment to spawn any new instances. Note in particular that the adversary may not use forced-write and hence cannot spawn new protocol instances. So in a run of a  $\xi$ -identity bounded environment with  $\pi/\phi$  there will never be any protocol instances, making both worlds trivially indistinguishable. Conversely, observe that  $\rho^{\phi \to \pi}$  and  $\rho$  are easily distinguishable by any environment that sends a single input. This input spawns a new instance of  $\rho$ , which then in turn forwards that input to its subroutine. By definition of  $\xi$ , this is permitted since now an instance of  $\rho$  exists. Similar counter examples can also be obtained, e.g., by letting  $\xi$  depend on the existence of multiple sessions of  $\pi/\phi$ .

**Our fix.** Since the underlying issue is that  $\xi$  behaves differently depending on the setting,  $\xi$  needs to be restricted in such a way that it behaves identical independently of whether we are currently considering a protocol  $\rho$  or an environment internally simulating the behavior of  $\rho$ . Observe that in both cases the sequence of inputs and outputs between (the internally simulated)  $\rho$  and one of its subroutine sessions of  $\phi$  remains the same.

We hence propose, following a similar idea as [1], considering only predicates  $\xi$  that are defined over this sequence of inputs to/outputs from a session, excluding the code  $\pi/\phi$  contained in the receiver resp. sender extended identities of the subroutine, but nothing more. This then fixes the issue in the composition theorem.

In particular, if  $\rho$  is  $\xi$ -identity bounded w.r.t. to one of its subroutine sessions of  $\phi$ , then an environment internally simulating  $\rho$  while running with (a single session of)  $\phi$  also is  $\xi$ -identity bounded. Furthermore, the same assumption on  $\rho$ also implies that the environment is  $\xi$ -identity bounded even while running with  $\pi$ , except for negligible probability, as otherwise one could trivially distinguish  $\pi$  and  $\phi$  by building an environment that evaluates  $\xi$  itself. Here we use that  $\xi$ does not depend on the code  $\pi/\phi$  contained in extended identities, which the environment does not have access to. Both of these properties are necessary for showing the composition theorem, which has to construct an environment  $\mathcal{E}$  that is always  $\xi$ -identity bounded, not just in runs with  $\phi$  but also in runs with  $\pi$ .<sup>32</sup>

Discussion of an alternative attempt to fix the issue. One might consider defining the class of predicates  $\xi$  not only over the sequence of inputs/outputs to/from one session of  $\pi/\phi$  but, more generally, also over the internal states of that individual session. Intuitively, this state does not change depending on whether the session is currently running as a subroutine of a protocol  $\rho$  or with an environment that internally simulates the behavior of  $\rho$ . So the issue described above does not occur, i.e., one might hope that the composition theorem holds true for this definition.

Given this definition of a class of predicates  $\xi$ , we first observe that the behavior of such a  $\xi$  generally changes drastically depending on whether we consider the protocol  $\rho$  running with sessions of  $\phi$  or the composed protocol  $\rho^{\phi \to \pi}$  running with sessions of  $\pi$  instead. This because the internal state of  $\phi$  is drastically different than the internal state of  $\pi$  (unless we have  $\pi = \phi$ , in which case the composition theorem is not needed). For example,  $\xi$  might depend on the existence of some internal instances present in  $\pi$  but not  $\phi$ .  $\xi$  might also depend on some variables used only in one protocol but not the other, or  $\xi$  might simply depend on the code of  $\pi/\phi$ . Hence, even assuming that  $\rho$  respects  $\xi$  for inputs to a session of  $\phi$ ,  $\rho^{\phi \to \pi}$  might not at all respect  $\xi$  for any inputs to a session of  $\pi$ . As a result, an environment internally simulating  $\rho$  and running with either  $\phi$  or  $\pi$  would be  $\xi$ -identity bounded for  $\phi$  but generally not for  $\pi$ , i.e., would not be a valid  $\xi$ -identity bounded environment that we can use for a reduction. So the composition theorem cannot be shown without introducing any additional assumptions if we allow  $\xi$  to depend on the internal state of  $\pi/\phi$ .

One might consider adding the new requirement for the composition theorem that also  $\rho^{\phi \to \pi}$  must respect  $\xi$  for inputs provided to  $\pi$ . This requirement appears to at least be necessary and potentially also sufficient for showing the composition theorem for the class of predicates  $\xi$  that depend on the internal state of a session of  $\pi/\phi$ . Assuming that the composition theorem can indeed be proven, let us

<sup>&</sup>lt;sup>32</sup> Formally, to get rid of the negligible chance of the environment  $\mathcal{E}$  simulating  $\rho$  violating  $\xi$  in a run with  $\pi$ , one instead constructs an environment  $\mathcal{E}'$  that simulates  $\mathcal{E}$  but, for each input provided by  $\mathcal{E}$ , first evaluates  $\xi$  itself and aborts the simulation if  $\xi$  is not met. Note that evaluating  $\xi$  is indeed possible for  $\mathcal{E}'$  using our definition. The resulting environment is then  $\xi$  identity bounded both for  $\phi$  and  $\pi$  and still behaves just as  $\rho$  respectively  $\rho^{\phi \to \pi}$ , except for negligible probability. This is sufficient for showing the theorem.

take a closer look at the class of protocols  $\rho$  that meet this new requirement, i.e., that not only respect  $\xi$  for inputs to  $\phi$  but are also such that  $\rho^{\phi \to \pi}$  respects  $\xi$  for inputs to  $\pi$ .

Let us first consider the subclass of predicates  $\xi$  which change their behavior depending on secret internal state of  $\phi/\pi$  that higher-level instances in  $\rho$  do not have access to, such as local variables or IDs of internal instances present only in one protocol, the number of internal instances, or even just the code of the protocol  $\phi/\pi$ .<sup>33</sup> Essentially the only way for both  $\rho$  and  $\rho^{\phi\to\pi}$  to meet  $\xi$  is to not send any inputs to  $\pi/\phi$  at all. In other words, we are considering the trivial class of protocols  $\rho$  where the composition theorem does nothing. This is because such a  $\xi$  essentially requires  $\rho$  to figure out whether it is still running with  $\phi$ or has been modified to instead run with  $\pi$  and then adapt its input behavior such that  $\xi$  is still met. This is not possible for  $\rho$ . Indeed, the whole point of the composition theorem is to argue that the behavior of  $\rho$  does not change when we replace sessions of  $\phi$  with sessions of  $\pi$ .

So to be able to consider interesting protocols  $\rho$  that actually contain subroutine sessions  $\phi$  that the composition theorem can replace, we essentially need a class of predicates  $\xi$  that depends only on "publicly available" information of a session of  $\pi/\phi$  that both  $\rho$  and hence also an environment simulating  $\rho$ and interacting with  $\pi/\phi$  has access to. Such publicly available information is therefore also guaranteed to be indistinguishable between  $\pi$  and  $\phi$ . Formalizing the intuitive meaning of "publicily available" in the case where  $\xi$  can take into account the entire internal state of  $\pi/\phi$  is a non-trivial task.

To come up with a possible formalization of this class of predicates  $\xi$ , let us consider the behavior of any concrete protocol  $\rho$ . Higher-level instances in  $\rho$  can change their behavior only depending on the state of  $\pi/\phi$  that they have already seen and accessed, which are exactly the inputs sent to and outputs obtained from a session of  $\pi/\phi$  (except for the code of  $\pi/\phi$  contained in the extended receiver/sender identities). Hence, any construction of a higher-level protocol  $\rho$  can only ever meet predicates  $\xi$  that only depend on and can be computed based on that information. Conversely, if  $\xi$  changes its behavior depending on some publicly available information that  $\rho$  could theoretically retrieve but never actually accesses, then  $\rho$  will not meet  $\xi$ . But these observations lead us back to our previous proposed definition of a class of predicates  $\xi$ . Namely, to be able to construct protocols  $\rho$  that meet  $\xi$ , such predicates should depend on and be computable from only the sequence of inputs to and outputs from a session of  $\pi/\phi$ .

So altogether the attempt of letting  $\xi$  also depend on the internal state of a session of  $\pi/\phi$  not only requires at least an additional assumption on  $\rho^{\phi \to \pi}$ . It also does not appear to yield a more general composition theorem since any higher-level protocol  $\rho$  that one might consider can only meet predicates  $\xi$  that depend solely on the information about the state of  $\pi/\phi$  available to  $\rho$  so far, i.e., exactly the information that our proposed definition already takes into account.

<sup>&</sup>lt;sup>33</sup> Recall that the change in code from  $\phi$  to  $\pi$  is hidden from higher-level instances within  $\rho^{\phi \to \pi}$ .

## J.4 Non-forced Write for Outputs Within Higher-Level Protocols

Recall that protocols in the UC model are allowed to use non-forced-writes for inputs and outputs sent to other protocol machines (cf. Figure 6 and Page 40 of [7]). That is, they may specify the receiving protocol instance of an input/output message via a predicate over extended IDs, where the first extended ID (in the order of their first creation) that matches then receives the message. In the setting of the composition theorem, this is more restricted. Specifically, neither  $\rho$  nor any of its subroutine instances, including  $\phi$  and its subroutine instances, may use non-forced-writes for providing inputs. It turns out, however, that this is not sufficient. Without also imposing restrictions on non-forced write outputs, the composition theorem is actually false (Theorem 22 in [7]).

Intuitively, the reason is the following: suppose some higher-level instance within  $\rho/\rho^{\phi\to\pi}$  generates an output via non-forced write. Then the behavior of this write command generally depends also on the existence and time of creation of internal subroutine instances within  $\pi/\phi$ . In other words, this creates a sidechannel that allows the behavior of  $\rho$  to depend on internal subroutine instances of  $\pi/\phi$ . However, an environment does not have access to this side channel and hence cannot internally simulate  $\rho$  while running with a session of  $\pi/\phi$ . But this is needed for showing the composition theorem. Similar issues can occur if the subroutine  $\pi/\phi$  uses the non-forced write mode for providing outputs. In the stand alone setting where  $\pi/\phi$  directly runs with the environment, the predicate used by  $\pi/\phi$  might match some instance *i* that is an internal subroutine of  $\pi/\phi$ . i.e., the message would be delivered to that internal instance i. In contrast, while running within a protocol  $\rho$ , there might be some other protocol instance j within  $\rho$  that also matches the predicate but was created earlier than *i*. Hence, when the subroutine  $\pi/\phi$  uses non-forced write to deliver the output, the instance j instead of i gets to process the message. Again, this behavior of the protocol  $\rho$ cannot be simulated by an environment directly running with a session of  $\pi/\phi$ since the environment never gets to see the message.

Consider the following concrete counter example to the composition theorem. The real protocol  $\pi$ , upon receiving its first input, forwards that input to an internal subroutine  $\pi'$  with PID 0. This subroutine does nothing, i.e., it simply drops all messages, which then activates the environment. After receiving its first input,  $\pi$  also drops all future messages. We define the ideal protocol  $\phi$  in the same way, but  $\phi$  uses PID 1 for its subroutine. We have that  $\pi \leq_{UC} \phi$  for the simulator that behaves as the dummy adversary but additionally translates network messages for the directory machine by switching from PID 0 to PID 1 and vice versa if the request is about the subroutine instance. Observe in particular that the environment has no way to check whether the internal instance has PID 0 or 1, which is exactly as it should be in a reasonable model.

The protocol  $\rho$ , upon receiving its first input, sends an input to  $\phi$  (with PID 2), which then activates the environment. Upon receiving its second input,  $\rho$  sends an input to a new subroutine  $\rho'$  with PID 1.  $\rho'$  is defined to take all inputs and outputs from  $\rho$  and return them as outputs to  $\rho$ . As soon as  $\rho$  regains control,  $\rho$  sends the output **real** via a non-forced write using the predicate that matches

machine instances with PID 1. Afterwards,  $\rho$  forwards outputs from  $\rho'$  to the environment but drops all other messages. Observe that  $\rho$  and  $\rho^{\phi \to \pi}$  are easily distinguishable by an environment that sends two inputs: Upon receiving the second input,  $\rho$  does not output anything in the ideal world. This is because the predicate will match the subroutine  $\phi'$ , which was created before the instance of  $\rho'$  with the same PID, and  $\phi'$  simply drops that message, thereby activating the environment. In contrast,  $\rho^{\phi \to \pi}$  in the real world will output **real** to the environment. This is because the only instance with PID 1 in  $\rho^{\phi \to \pi}$  is  $\rho'$ , which then returns the message **real** to  $\rho^{\phi \to \pi}$  and  $\rho^{\phi \to \pi}$  forwards that message to the environment. A similar counterexample can be constructed by defining  $\pi/\phi$  in such a way that the behavior of their predicates depends on other instances (outside of their own sessions) within  $\rho$ .

**Our fix.** We require that, in runs of  $\rho/\rho^{\phi\to\pi}$  within an arbitrary environment/context, predicates used by higher-level instances of  $\rho/\rho^{\phi\to\pi}$  for non-forced writes never match instances of any session of  $\pi/\phi$ . The same must also hold in the other direction for predicates used by instances of sessions of  $\pi/\phi$  (in runs of  $\rho/\rho^{\phi\to\pi}$ ), i.e., they may not match any instances in  $\rho/\rho^{\phi\to\pi}$  not belonging to the same session of  $\pi/\phi$ . This is sufficient to fix the above issues since, using these requirements, the behavior of a session of  $\pi/\phi$  and the other instances of  $\rho/\rho^{\phi\to\pi}$  does not change when an environment interacting with one session of  $\pi/\phi$  only internally simulates the remainder of  $\rho/\rho^{\phi\to\pi}$ . Alternatively, one could use the stronger and simpler requirement that forbids non-forced writes not just for inputs but also for outputs. This is still reasonable since the main purpose of non-forced writes in protocols appears to be to allow the protocol to send a network message to the adversary without being aware of the full code of the adversary.

# J.5 Simulator Definition for the Composition Theorem

The simulator defined in the proof of the composition theorem is formally incomplete since it does not specify how a certain type of network messages from the environment to the dummy adversary are handled. We observe that this appears to be an oversight that can easily be fixed by adding the same code as used by the dummy adversary to process these messages. See E for full details.