Embedding the UC Model into the IITM Model

Daniel Rausch$^{1\,[0000-0002-1901-3659]}$, Ralf Küsters$^{1\,[0000-0002-9071-9312]}$, and Céline Chevalier^{2}

^{1} University of Stuttgart
{daniel.rausch, ralf.kuesters}@sec.uni-stuttgart.de
^{2} 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 π secure w.r.t. an ideal functionality ϕ one shows that for all network adversaries $\mathcal{A}$ attacking π there is an ideal adversary, called simulator, that interacts with ϕ 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 ϕ used by an arbitrary higher-level protocol ρ with their realization π such that $\rho^{\phi \to \pi} \leq \rho$, where in $\rho^{\phi \to \pi}$ the protocol ρ uses (possibly multiple instances of) π instead of ϕ.

The UC model by Canetti [5, 6] and the reactive simulatability model by Pfitzmann and Waidner [20] 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 [8] and GUC [7] 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 GNUC model [10] 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 [12, 18] 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 [19] 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 [13] the relationship between security notions employed in various models has been studied, although the study was carried out in one model, and [11, 12] discuss runtime notions employed in different models.

On the UC and IITM models. We choose the UC model [5] since it is currently the most widely used model in the literature on universal composability. The IITM model [12] has also already been used intensively to analyze a wide variety of protocols, including cryptographic protocols (e.g., [14, 15]) and also more generally security protocols such as blockchains and distributed ledgers (e.g., [9]). 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 [15], i.e., parties in one session are not required to share the same SID or fix it upfront (see [4, 15, 17, 18] 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 [6] and the IITM model in the Journal of Cryptology [18]. 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 faithful, 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, [18] 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 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 [6,18] 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 identity 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 c specified 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 η 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 and n_O is the sum of all imports sent by (that instance of) M to other machines, 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. Furthermore, all machines are parameterized with a security parameter η. All machine instances are required to run only when they hold at least η import, i.e., $n_I - n_O \geq \eta$.

[3] A function $f\colon \mathbb{N} \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$ 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 $a \in \bigcup_{\eta' \leq \eta^d} \{0,1\}^{\eta'}$: $f(\eta, a) < \eta^{-c}$.

Simulation-Based Security. Security of a protocol π 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 π, $\mathcal{A}$, 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 $(\mathit{pid}, \mathit{sid})$ 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 π. So $\mathcal{E}$ can talk to $\mathcal{A}$ or to π (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 $(\diamond, \mathit{sid}_c)$) 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).[4]

– All other machine instances (which are part of the protocol stack of π, 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 $(M, (\mathit{pid}, \mathit{sid}))$ has code $M = \pi$, $\mathit{sid} = \mathit{sid}_c$, 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 η (as per the 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 π, i.e., all instance with extended IDs of the form $(\pi, (\mathit{pid}, \mathit{sid}_c))$, where sid_c is the fixed challenge SID. Given a set of extended identities ξ, an environment is called ξ*-identity-bounded* if it claims only sender extended identities from ξ. The set ξ 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:

[4] The journal version of the UC model [6] 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. the full version [21] for details). 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 1. *Let π and ϕ be ppt protocols. Then π realizes ϕ w.r.t. ξ-identity-bounded environments ($\pi \leq^{\xi}_{UC} \phi$) 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 ξ-identity-bounded environments $\mathcal{E}$ it holds true that $(\mathcal{E}, C^{\pi,\mathcal{A}}_{EXEC}) \equiv (\mathcal{E}, C^{\phi,\mathcal{S}}_{EXEC})$.*[5]

Composition Theorem. To state the composition theorem, a bit more terminology is needed. A *session* (with SID *sid*) of a protocol π consists of all instances running code π 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 π 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 π).

A protocol π is called *subroutine respecting* if a protocol session of π 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 π is used as a subroutine within a higher-level protocol ρ.[6] 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 π and its subroutines. A protocol π 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 ρ is called (π, ϕ, ξ)*-compliant* if *(i)* all instances of all sessions of ρ perform write requests to input tapes only via forced-write 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 ρ) for the same SID but one for code π while the other is for code ϕ, and *(iii)* the extended identities of all instances in all sessions of ρ that pass inputs to an instance with code π or ϕ satisfy the polytime predicate ξ. Given such a (π, ϕ, ξ)-compliant protocol ρ, the protocol $\rho^{\phi\to\pi}$ is defined just as ρ but replaces (input write requests to) subroutine instances of ϕ with (input write requests to) subroutine instances of π. In subroutines of ρ this replacement is done as well. [7] Now, the composition theorem is as follows:

[5] 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 definitio to be equivalent. Also, if ξ always permits all identities, then one simply writes $\leq_{UC}$ instead of $\leq^{\xi}_{UC}$.

[6] The subroutine respecting property ensures that π running within a larger protocol ρ still behaves as in the security experiment, where an environment can interact only with one session of π and only via the highest-level instances of that session.

[7] Formally, $\rho^{\phi\to\pi}$ contains an additional so-called UC composition shell code which acts as a wrapper that replaces these write requests.

Theorem 1 (UC Composition [6]). *Let ρ, π, ϕ be ppt protocols, let ξ be a ppt predicate, such that ρ is (π, ϕ, ξ)-compliant, π and ϕ are both subroutine respecting and subroutine exposing, and $\pi \leq^{\xi}_{UC} \phi$. 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.[8] The names are used for determining pairwise connections between machines in a system of machines.[9] 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* $\mathcal{Q}$ of IITMs is a set of IITMs of the form $\mathcal{Q} = \{M_1, \cdots, M_k, !\,M'_1, \cdots, !\,M'_{k'}\}$[10] 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 $\mathcal{Q}$ 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 $\mathcal{Q}$, at any time only one machine instance is active and all other instances wait for 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

[8] 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].

[9] 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.

[10] Also written $M_1 \mid \cdots \mid M_k \mid !\,M'_1 \mid \cdots \mid !\,M'_{k'}$.

in mode CheckAddress with input m on tape n. If it accepts, it gets to process m using 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 $\mathcal{Q}$ and $\mathcal{R}$ are called indistinguishable ($\mathcal{Q} \equiv \mathcal{R}$) if the difference between the probability that $\mathcal{Q}$ outputs 1 (on the decision tape) and the probability that $\mathcal{R}$ outputs 1 is negligible in the security parameter η and the external input a (see Footnote 3).

Types of Systems and Their Runtime. We need the following terminology. For a system $\mathcal{Q}$, the tapes of machines in $\mathcal{Q}$ that do not have a matching tape, i.e., there does not exist another machine in $\mathcal{Q}$ 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: [11]

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 $\mathcal{P}$*

[11] 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 [18].

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 [18]). *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 $\mathcal{S} \,|\, \mathcal{F}$, 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. 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 $\mathcal{Q}$ 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) ρ in Theorem 1, and $\mathcal{P}$ and $\mathcal{F}$ to the subroutines π and ϕ, 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 that $\pi_{UC} \leq^{\xi}_{UC} \phi_{UC}$ and the UC composition theorem can be applied to ρ_{UC} to obtain $\rho^{\phi \to \pi}_{UC} \leq_{UC} \rho_{UC}$. 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 ξ. 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).

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

Normalization. W.l.o.g., let us first bring the protocol π_{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 π_{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_π *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 π_{UC} is such that within each protocol session sid_c there are no instances running code c_π 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 π_{UC} that forwards messages between the environment and those instances running code c_π with SID sid_c. 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 π_{UC} so is indeed without loss of generality. We also assume that the protocol π_{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 π_{UC} given that π_{UC} is already assumed to be subroutine respecting.

IITMs and tapes. We model the protocol π_{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 π_{IITM}. Just as π_{UC}, π_{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.[12] 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 π_{UC} we can also extend the protocol π_{IITM} to allow for sending messages between different instances of the *same* machine. This is done by adding a special bouncer machine M_{bc} to the system π_{IITM}. M_{bc} connects to all machines in the system via a pair of I/O tapes each.

[12] 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.

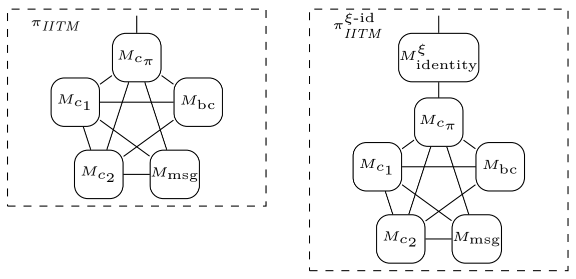

Fig. 1. Left: Static structure of a protocol π_{IITM} using three $\mathsf{Codes} = \{c_\pi, c_1, c_2\}$ constructed by our mapping. Right: Static structure of the modified protocol $\pi^{\xi\text{-id}}_{IITM}$ that enforces ξ-identity bounded environments (cf. Section 3.3). Lines denote internal connections 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 $(\mathit{pid}, \mathit{sid})$.

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_{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_π} 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_{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_{msg} along with the description of machines M_c below.

Addressing of instances. In π_{UC}, an instance of a machine running code c is uniquely addressed by an ID of the form $(\mathit{pid}, \mathit{sid})$ and learns the ID $(\mathit{pid}_s, \mathit{sid}_s)$ and code c_s of senders who provide input or subroutine output. Furthermore, by our initial normalization of π_{UC}, we have that $\mathit{sid} = (\mathit{sid}_c, \mathit{sid}')$ for internal instances, i.e., instances running code $c \neq c_\pi$, and $\mathit{sid} = \mathit{sid}_c$ for instances running c_π. To capture these unique IDs for instances in π_{IITM}, we use a suitable instantiation of the $\mathsf{CheckAddress}$ mode, cf. Figure 2. That is, instances of M_c expect incoming inputs/outputs m to be of the form $((\mathit{pid}, \mathit{sid}, c), (\mathit{pid}_s, \mathit{sid}_s, c_s), m')$,

```
Mode CheckAddress:
    Let $m$ be the message received on 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
    on the network tape. Furthermore, if $c \neq c_\pi$, also try to parse $sid$ as $(sid', sid'')$
    If parsing fails, return reject.
    if $id = \perp \vee id = (pid, sid)$ then     { id is a global variable that is ⊥ iff this instance is
        return accept.                            { fresh. It is set to be the ID of the current instance
                                                  { in mode Compute upon accepting the first message.
    else
        return reject.
    end if
```

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

where m' is the actual message body.[13] Network messages from the adversary are expected to be of the form $((pid, sid, c), m')$.[14] 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 (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

[13] 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.

[14] Network messages do not contain a sender identity since the sender is always know to be the network adversary.

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 π_{IITM} as in π_{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 π_{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 π_{UC} is created), the instance registers itself with the instance (ϵ, sid_c) of M_{msg} by sending $((\epsilon, sid_c, c_{M_{\text{msg}}}), (pid, sid, c), \texttt{register})$ 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_π} and it receives some import $i > 0$ in a message from the environment, then it sends $((\epsilon, sid_c, c_{M_{\text{msg}}}), (pid, sid, c), (\texttt{notifyImport}, i))$ to notify the session specific instance M_{msg} about this amount. M_{msg} stores i and returns an acknowledgement; we describe the purpose of registrations and import notifications later on.

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 π_{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

```
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
      $id \leftarrow (pid, sid)$          {Store the ID of this instance such that the
  end if                                   CheckAddress mode can use this information.
  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
  if this is the first message received on an I/O tape then
      Send $((\epsilon, sid_c), (pid, sid, c), \texttt{register})$ on the tape connected to $M_{\text{msg}}$, 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 $((\epsilon, sid_c), (pid, sid, c), (\texttt{notifyImport}, i))$ on the tape connected to $M_{\text{msg}}$, 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 $tt \in$
  $\{\texttt{input}, \texttt{output}, \texttt{backdoor}\}$ 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 17. 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 $c \in$ Codes

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_{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_{msg}. The machine M_{msg} offers a "`nonForcedWrite`" command to the machines M_c that, given message body m', message type $mt \in \{\texttt{input}, \texttt{output}\}$, 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_{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 highest-level protocol machine M_{c_π} 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 π_{UC} all instances of M_{c_π} 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_π} wants to send a subroutine output m' to an extended receiver identity $eid_r = (pid_r, sid_r, c_r)$, M_{c_π} first asks M_{msg} whether eid_r already exists in the system (via a special `existsInstance`? request). If so, the message is sent by M_{c_π} 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_π} that is connected to the environment. Note that, unlike for other messages, the sender machine code c_π 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_{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 the full version [21] 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 π_{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 $\pi_{IITM} := \,!M_{c_\pi} \mid \,!M_{c_1} \mid \ldots \mid \,!M_{c_{n-1}} \mid \,!M_{\text{msg}} \mid \,!M_{\text{bc}}$. Based on the construction and the discussion above, we can easily check that π_{UC} and π_{IITM} behave the same:

Lemma 1. *For all unbounded (including runtime) environments interacting with π_{UC}/π_{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 π_{UC} in the UC model and π_{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 π_{IITM}. In the UC setting with π_{UC} this request can instead be internally simulated by the environment. □

We show in the next lemma (proven in the full version [21]) that π_{IITM} is a well-defined 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 π_{IITM} is environmentally bounded in the IITM model if π_{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 $\pi_{UC} \leq^{\xi}_{UC} \phi_{UC}$, then $\pi^{\xi\text{-id}}_{IITM} \leq_{IITM} \phi^{\xi\text{-id}}_{IITM}$ for protocols mapped as described in Section 3.2 plus an additional mechanism to capture ξ-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 ξ-identity bounded behavior can be enforced within protocols themselves, thereby bridging Diff. 10.

While designing this mechanism, we found that the definition of ξ-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 the full version [21]. In a nutshell, the issue is that the UC model allows for defining the identity set ξ 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 π_{UC} respectively ϕ_{UC}, compared to the composition theorem, where there are additional instances of a higher-level protocol ρ as well as potentially multiple sessions of π_{UC}/ϕ_{UC}. Hence, even if ρ is ξ-compliant in the setting where instances of ρ and multiple sessions of π_{UC}/ϕ_{UC} exist, this does not imply that an environment internally simulating ρ while running only with π_{UC}/ϕ_{UC} (but with no actual instances of ρ and only a single session of π_{UC}/ϕ_{UC} being present in the system) also is ξ-identity-bounded. Based on this observation, in the full version 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 ξ 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/ρ to/from one session of the subroutine π/ϕ. This fix, which follows a similar idea as [1], indeed solves the problem: The sequence of messages between ρ and one of its subroutine sessions remains the same (for each respective subroutine session) even if we only simulate ρ within an environment. Hence, such an environment running directly with a single session of the subroutine π/ϕ is indeed ξ-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 ξ that falls outside this class. We provide an extended discussion in the full version [21].

We now embed this (fixed) definition of ξ-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 π_{IITM} and ϕ_{IITM} in a generic way to manually enforce the ξ-identity-bounded property for all environments. This is a technique that is commonly used in the IITM model, see for example [14,16]. Formally, we add to each protocol an additional dummy forwarder machine $M^{\xi}_{\text{identity}}$ between the environment and the highest-level machine M_{c_π} respectively M_{c_ϕ}, creating new protocols $\pi^{\xi\text{-id}}_{IITM}$ and $\phi^{\xi\text{-id}}_{IITM}$ (cf. right hand-side of Figure 1). In a run, (a session specific instance of) $M^{\xi}_{\text{identity}}$ checks for every input whether ξ is met and, if not, drops the input, thereby activating the environment as a fallback. This achieves the desired goal: On the one hand, environments that are already ξ-identity-bounded are not restricted since for such environments the original protocols π_{IITM}/ϕ_{IITM} and the modified protocols $\pi^{\xi\text{-id}}_{IITM}/\phi^{\xi\text{-id}}_{IITM}$ behave identically. For any other environment $\mathcal{E}$, the combination of $\mathcal{E}$ and $M^{\xi}_{\text{identity}}$ constitutes a ξ-identity-bounded environment for the original protocol. Note that the extended protocols are still environmentally bounded as $M^{\xi}_{\text{identity}}$ adds only a polynomial number of steps; in particular, ξ 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 the full version [21], 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 π_{UC}, ϕ_{UC} be such that $\pi_{UC} \leq^{\xi}_{UC} \phi_{UC}$. Then it holds true that $\pi^{\xi\text{-}id}_{IITM} \leq_{IITM} \phi^{\xi\text{-}id}_{IITM}$.*

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. The proof proceeds in 4 steps (see the full version [21] for details and the general case):

Reduction to UC. We first define an IITM dummy adversary $\mathcal{A}^{\text{UC-bounded}}_{Dum,IITM}$ and an IITM simulator $\mathcal{S}^{\text{UC-bounded}}_{IITM}$ that adhere to the UC runtime notion and 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 $\mathcal{E}^{\text{single},\xi}_{IITM}$ that sends inputs and network messages (via the dummy adversary) only to a single session of the protocol π_{IITM}/ϕ_{IITM}, adheres to the ξ-identity bound, and tries to distinguish the worlds $\mathcal{A}^{\text{UC-bounded}}_{Dum,IITM} \,|\, \pi_{IITM}$ and $\mathcal{S}^{\text{UC-bounded}}_{IITM} \,|\, \phi_{IITM}$. 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 $\mathcal{E}^{\text{single},\xi}_{IITM}$. $\mathcal{E}_{UC}$ 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 *op* with 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 ξ-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 $\mathcal{A}_{Dum,IITM} \,|\, \pi_{IITM}^{\xi\text{-id}}$ and $\mathcal{S}_{IITM} \,|\, \phi_{IITM}^{\xi\text{-id}}$ to indistinguishability of the worlds $\mathcal{A}_{Dum,IITM}^{\text{UC-bounded}} \,|\, \pi_{IITM}^{\xi\text{-id}}$ and $\mathcal{S}_{IITM}^{\text{UC-bounded}} \,|\, \phi_{IITM}^{\xi\text{-id}}$ by constructing an environment ${\mathcal{E}'}_{IITM}^{\text{single}}$ 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 $\mathcal{A}_{Dum,IITM} \,|\, \pi^{\xi\text{-id}}_{IITM}$ and $\mathcal{S}_{IITM} \,|\, \phi^{\xi\text{-id}}_{IITM}$ are indistinguishable for any environment $\mathcal{E}^{\text{single}}_{IITM}$ interacting with just a single session, the second composition theorem of the IITM model (cf. Theorem 3) immediately implies that $\pi^{\xi\text{-id}}_{IITM} \leq_{IITM} \phi^{\xi\text{-id}}_{IITM}$, i.e., there also exists a simulator for arbitrary environments $\mathcal{E}_{IITM}$ interacting with multiple sessions. $\square$

The construction of the simulator $\mathcal{S}^{\text{UC-bounded}}_{IITM}$ 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 η 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 π_{UC} and ϕ_{UC} such that for the mapped protocols we have $\pi^{\xi\text{-id}}_{IITM} \leq_{IITM} \phi^{\xi\text{-id}}_{IITM}$ 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 the full version, 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 $\mathcal{S}^{\text{UC-bounded}}_{IITM}$ constructed in the proof of Theorem 4, we have the following reverse implication:

Theorem 5. *Let $\mathcal{A}^{UC\text{-bounded}}_{Dum,IITM}$ 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 $\mathcal{S}^{UC\text{-bounded}}_{IITM}$ be an IITM simulator that is of the form as the one described in the proof of Theorem 4.*

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

We provide the proof in the full version [21]. 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 ξ-identity bounded UC environment that distinguishes π_{UC} and ϕ_{UC}, then Theorem 5 implies that for all IITM simulators in the UC runtime class there is an IITM environment distinguishing $\pi^{\xi\text{-id}}_{IITM}$ and $\phi^{\xi\text{-id}}_{IITM}$.

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 $\pi_{UC} \leq^{\xi}_{UC} \phi_{UC}$ and the UC composition theorem can be applied to ρ_{UC} to obtain $\rho^{\phi\to\pi}_{UC} \leq_{UC} \rho_{UC}$. Let ρ_{IITM} and $\rho^{\phi\to\pi}_{IITM}$ be the IITM protocols obtained by applying the mapping from Section 3.2 to ρ_{UC} and $\rho^{\phi\to\pi}_{UC}$.*[15] *Then $\rho^{\phi\to\pi}_{IITM} \leq_{IITM} \rho_{IITM}$.*

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 π_{IITM}/ϕ_{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 π_{IITM}/ϕ_{IITM}, which offer only a single external I/O tape to/from the machine with code c_π/c_ϕ, this syntactical requirement of the IITM theorem actually corresponds to the "subroutine respecting" requirement for π_{UC}/ϕ_{UC} in the UC theorem.[16] 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 π_{UC}/ϕ_{UC}, except for inputs to and outputs from highest-level instances running code c_π/c_ϕ. 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 $\rho^{\phi\to\pi}_{IITM} \leq_{IITM} \rho_{IITM}$ we have to make some slight syntactical adjustments to ρ_{IITM} such that the semantic "subroutine respecting" property is also reflected by the tape connections.

[15] Note that $\rho^{\phi\to\pi}_{UC}$ also contains some UC composition shell code introduced by the UC composition theorem to replace the code c_ϕ with c_π. $\rho^{\phi\to\pi}_{IITM}$ is thus obtained by mapping the overall machine codes, including the UC composition shell code.

[16] 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.

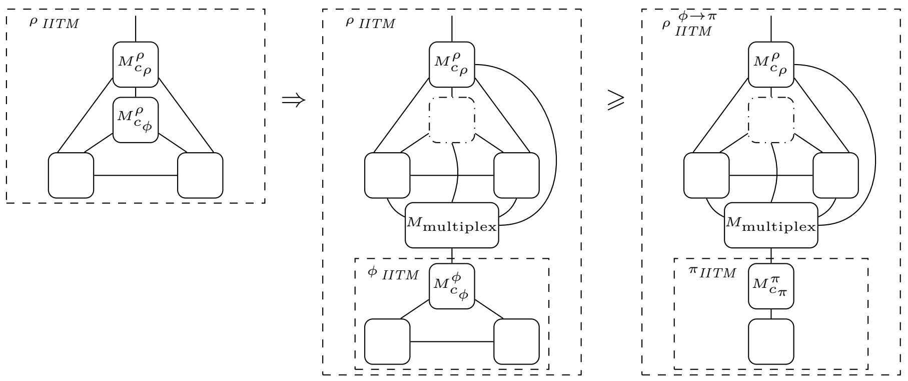

Fig. 4. Overview of the static structures of the protocols in this section. Left: $\rho_{\,IITM}$ mapped as per Section 3.2. Middle: $\rho_{\,IITM}$ after redirecting all inputs/outputs from $M^{\rho}_{c_\phi}$ to $\phi_{\,IITM}$. The machine $M^{\rho}_{c_\phi}$ is formally still present but not used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

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^{ρ}_{i}. 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^{ρ}_{i}, 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^{ρ}_{i}. Since inputs to and outputs from $M^{\rho}_{c_\phi}$ are the only way for higher-level instances in ρ to interact with instances in any session of the subroutine ϕ (by the subroutine respecting property), this syntactic modification of $\rho_{\,IITM}$ does not actually change its behavior. It, however, consistently moves all sessions of ϕ 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 $\rho^{\,\phi\to\pi}_{\,IITM}$ is then defined by simply replacing the set of machines $\phi_{\,IITM}$ with the set of machines π_{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 π_{IITM}. As a result, in $\rho^{\,\phi\to\pi}_{\,IITM}$ all inputs to and outputs from sessions of $\phi_{\,IITM}$ are now instead handled by sessions of π_{IITM}, which is just as in $\rho^{\,\phi\to\pi}_{\,UC}$. In other words, reconnecting this tape has the same effect as adding the UC composition shell code, which internally changes the code c_ϕ to instead be c_π 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 $\rho\,^{\phi\to\pi}_{IITM}$. 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 ρ_{UC} meets the requirements of the UC composition theorem. Let ρ_{IITM} and $\rho\,^{\phi\to\pi}_{IITM}$ be the IITM protocols from above. Then immediately by the IITM composition theorem, $\rho\,^{\phi\to\pi}_{IITM} \leq_{IITM} \rho_{IITM}$.*

We provide full details, including the formal definitions of ρ_{IITM}, $\rho\,^{\phi\to\pi}_{IITM}$ and the proof of Corollary 2 in the full version [21]. 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 ρ are actually necessary. 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 ρ replaces *all* sessions of subroutines running code ϕ with sessions running code π. Similarly, the IITM theorem applied to a protocol ρ replaces *all* sessions of a set of machines ϕ with sessions of a set of machines π. Observe that we can use the above modeling technique not just to move *all* sessions of ϕ to a new set of machines. Under certain conditions, we can rather more generally move a proper subset of the sessions of ϕ to a new set of machines, say ϕ', while moving the other sessions to a different set, say ϕ'', where ϕ' and ϕ'' still run the same code c_ϕ. We then obtain a simple corollary of the UC and IITM composition theorems (also for similar models), where we can replace ϕ' with a realization π' but replace ϕ'' with a different realization π''. In other words, our technique allows for *replacing subsets of sessions*. This can be useful, e.g., if ϕ is an ideal signature functionality, where each session models one key pair. Then we might want to implement certain keys with a signature scheme π' but others with a different signature scheme π'', say, depending on where they are used within a higher-level protocol ρ. We give full details, including requirements on ρ, in [21].

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 π_{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 π_{IITM} as described in Section 3.2. We then add a universal Turing machine M_{UT} that all other machines M_{c_i} connect to via pairs of I/O tapes. Each instance of M_{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 π_{IITM} wants to send a message to an instance with ID (pid, sid) and code $c \notin \mathsf{Codes}$, i.e., where M_c does not exist in π_{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 π_{IITM} behaves just as π_{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 the full version [21] 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.

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 ϕ_{UC}, including any security and composability results, into the IITM model, we are no longer limited to only considering combinations of ϕ_{IITM} with such mapped protocols. We can rather consider any combination of ϕ_{IITM} with arbitrary other IITM protocols. This includes cases where a higher-level IITM protocol $\mathcal{P}$ is designed based on top of ϕ_{IITM}, which can then, by the IITM composition theorem, be composed with any existing UC realization π_{IITM} of ϕ_{IITM}. One can also consider novel realizations of ϕ_{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 higher-level IITM protocol $\mathcal{P}$ can be defined in such a way that different sessions of $\mathcal{P}$ share the same instance of ϕ_{IITM} and $\mathcal{P}$ could also work without pre-established SIDs etc. We refer the reader to [4, 15, 17, 18] 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 [18], 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).

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. We leave exploring the details of this direction for future work.

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

References

1. Backes, M., Dürmuth, M., Hofheinz, D., Küsters, R.: Conditional Reactive Simulatability. International Journal of Information Security (IJIS) **7**(2), 155–169 (4 2008)
2. Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain UC. In TCC 2020, Part III. LNCS, vol. 12552, pp. 1–30. Springer, Heidelberg (Nov 2020).
3. Camenisch, J., Enderlein, R.R., Krenn, S., Küsters, R., Rausch, D.: Universal composition with responsive environments. In ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 807–840. Springer, Heidelberg (Dec 2016).

4. Camenisch, J., Krenn, S., Küsters, R., Rausch, D.: iUC: Flexible universal composability made simple. In ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 191–221. Springer, Heidelberg (Dec 2019).
5. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001. pp. 136–145. IEEE Computer Society Press (Oct 2001).
6. Canetti, R.: Universally Composable Security. J. ACM **67**(5), 28:1–28:94 (2020)
7. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer, Heidelberg (Feb 2007).
8. Canetti, R., Rabin, T.: Universal composition with joint state. In CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (Aug 2003).
9. 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)
10. Hofheinz, D., Shoup, V.: GNUC: A new universal composability framework. Journal of Cryptology **28**(3), 423–508 (Jul 2015).
11. Hofheinz, D., Unruh, D., Müller-Quade, J.: Polynomial runtime and composability. Journal of Cryptology **26**(3), 375–441 (Jul 2013).
12. 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 [18] for a full and revised version.
13. 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).
14. 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).
15. Küsters, R., Tuengerthal, M.: Composition theorems without pre-established session identifiers. In ACM CCS 2011. pp. 41–50. ACM Press (Oct 2011).
16. Küsters, R., Tuengerthal, M.: Ideal key derivation and encryption in simulation-based security. In CT-RSA 2011. LNCS, vol. 6558, pp. 161–179. Springer, Heidelberg (Feb 2011).
17. 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)
18. Küsters, R., Tuengerthal, M., Rausch, D.: The IITM Model: a Simple and Expressive Model for Universal Composability. J. Cryptol. **33**(4), 1461–1584 (2020)
19. Maurer, U.: Constructive cryptography - a primer (invited paper). In FC 2010. LNCS, vol. 6052, p. 1. Springer, Heidelberg (Jan 2010)
20. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In ACM CCS 2000. pp. 245–254. ACM Press (Nov 2000).
21. Rausch, D., Küsters, R., Chevalier, C.: Embedding the UC Model into the IITM Model. Cryptology ePrint Archive, Report 2022/224 (2022), https://eprint.iacr.org/2022/224