Universal Composition with Responsive Environments*

Jan Camenisch[1], Robert R. Enderlein[1,2], Stephan Krenn[3], Ralf Küsters[4], Daniel Rausch[4]

[1] IBM Research – Zurich, Rüschlikon, Switzerland
jca@zurich.ibm.com
[2] Department of Computer Science, ETH Zürich, Zürich, Switzerland
asiacrypt2016@e7n.ch
[3] AIT Austrian Institute of Technology GmbH, Vienna, Austria
stephan.krenn@ait.ac.at
[4] University of Trier, Trier, Germany
{kuesters,rauschd}@uni-trier.de

Abstract. In universal composability frameworks, adversaries (or environments) and protocols/ideal functionalities often have to exchange meta-information on the network interface, such as algorithms, keys, signatures, ciphertexts, signaling information, and corruption-related messages. For these purely modeling-related messages, which do not reflect actual network communication, it would often be very reasonable and natural for adversaries/environments to provide the requested information immediately or give control back to the protocol/functionality immediately after having received some information. However, in none of the existing models for universal composability is this guaranteed. We call this the *non-responsiveness problem*. As we will discuss in the paper, while formally non-responsiveness does not invalidate any of the universal composability models, it has many disadvantages, such as unnecessarily complex specifications and less expressivity. Also, this problem has often been ignored in the literature, leading to ill-defined and flawed specifications. Protocol designers really should not have to care about this problem at all, but currently they have to: giving the adversary/environment the option to not respond immediately to modeling-related requests does not translate to any real attack scenario.
This paper solves the non-responsiveness problem and its negative consequences completely, by avoiding this artificial modeling problem altogether. We propose the new concepts of responsive environments and adversaries. Such environments and adversaries must provide a valid response to modeling-related requests before any other protocol/functionality is activated. Hence, protocol designers do no longer have to worry about artifacts resulting from such requests not being answered promptly. Our concepts apply to all existing models for universal composability, as exemplified for the UC, GNUC, and IITM models, with full definitions and proofs (simulation relations, transitivity, equivalence of various simulation notions, and composition theorems) provided for the IITM model.

Keywords: universal composability; protocol design; cryptographic security proofs; responsive environments

* This project was in part funded by the European Commission through grant agreements n°s 321310 (PERCY) and 644962 (PRISMACLOUD), and by the *Deutsche Forschungsgemeinschaft* (DFG) through Grant KU 1434/9-1.

1 Introduction

One of the most demanding tasks when designing a cryptographic protocol is to define its intended security guarantees, and to then prove that it indeed satisfies them. In the best case, these proofs should guarantee the security of the protocol in arbitrary contexts, i.e., also when composed with other, potentially insecure, protocols. This would allow one to split complex protocols into smaller components, which can then be separately analyzed one by one and once and for all, thus allowing a modular security analysis. Over the past two decades, many models to achieve this goal have been proposed [3, 8–10, 19, 22, 24, 27–29], with Canetti's UC model being one of the first and most prominent ones.

All these models have in common that the designer first needs to specify an *ideal functionality* $\mathcal{F}$ defining the intended security and functional properties of the protocol. Informally, a *real protocol* realizes $\mathcal{F}$ if no efficient distinguisher (the *environment*) can decide whether it is interacting with the ideal functionality and a *simulator*, or with the real world protocol and an *adversary*.

Urgent requests/messages. In the specifications of such real protocols and ideal functionalities, it is often required for the adversary (and the environment) to provide some meta-information via the network interface to the protocol or the functionality, such as cryptographic algorithms, cryptographic values of signatures, ciphertexts, and keys, or corruption-related messages. Conversely, protocols/functionalities often have to provide the adversary with meta-information, for example, signaling information (e.g., the existence of machines) or again corruption-related messages. Such meta-information does not correspond to any real network messages, but is merely used for modeling purposes. Typically, giving the adversary/environment the option to not respond immediately to such modeling-specific messages does not translate to any real attack scenario. Hence, often it is natural for protocol designers to expect that the adversary/environment (answers and) returns control back to the protocol/functionality immediately when the adversary is requested to provide meta-information or when the adversary receives meta-information from the protocol/functionality. In the following, we call such messages from protocols/ideal functionalities on the network interface *urgent messages* or *urgent requests*.

Urgent requests occur in many functionalities and protocols from the literature, see, e.g., [1, 4, 5, 8, 11–13, 15, 16, 21, 25, 26, 31]. This is not surprising as the exchange of meta-information between the adversary/environment and the protocols/functionalities is an important mechanism for protocol designs in any UC-like model. For example, one can specify the behaviour of cryptographic values or algorithms by an ideal functionality in a natural manner without having to worry about how these values are generated or the parameters for the algorithms are set up, e.g., using a CRS. Also, protocols should be able to provide the adversary with meta-information in situations where it is not intended to give control to the adversary, such as certain information leaks (e.g, honest-but-curious corruption) or signaling messages. In general, it seems impossible to dispense with urgent requests altogether, and certainly, such requests are very convenient and widely used in the literature (see also §3).

The non-responsiveness problem. In the existing universal composability models, it currently is not guaranteed that urgent requests are answered immediately by the adversary: when receiving an urgent request on the network interface, adversaries and environments can freely activate protocols and ideal functionalities in between, on network and I/O interfaces, without answering the request. In what follows, we refer to this problem as the *problem of non-responsive adversaries/environments* or the *non-responsiveness problem*.

This problem formally does not invalidate any of the UC-style models. It, however, often makes the specification of protocols and functionalities much harder and the models less expressive (see below). Most disturbingly, as mentioned, the non-responsiveness problem is really an artificial problem: urgent requests do not correspond to any real messages, and the adversary not responding promptly to such requests does not reflect any real attack scenario. Hence, non-responsiveness forces protocol designers to take care of artificial adversarial behavior that was unintended in the first place and is merely a modeling artifact.

In particular, protocol designers currently have to deal with various delicate problems: i) While waiting for a response to an urgent request, a protocol/ideal functionality might receive other requests, and hence, protocol designers have to take care of interleaving and dangling requests. ii) While a protocol/ideal functionality is waiting for an answer from the adversary to an urgent request, other parties and parts of the protocol/ideal functionality can be activated in the meantime (via the network or the I/O interface), which might change their state, even their corruption status, and which in turn might lead to race conditions (see §3 for examples from the literature).

This, as further discussed in the paper, makes it difficult to deal with the non-responsiveness problem and results in unnecessarily complex and artificial specifications of protocols and ideal functionalities, which, in addition, are then hard to re-use. In some cases, one might not even be able to express certain desired properties. As explained in §3, there is no generic and generally applicable way to deal with the non-responsiveness problem, and hence, one has to resort to solutions specifically tailored to the protocols at hand.

Importantly, the non-responsiveness problem propagates to higher-level protocols as they might not get responses from their subprotocols as expected. The security proofs also become more complex because one, again, has to deal with runs having various dangling and interleaving requests as well as unexpected and unintuitive state changes, which do not translate into anything in the real world, but are just an artifact of the modeling.

Clearly, in the context of actual network messages, one has to deal with many of the above problems in the specifications of protocols and ideal functionalities too. But, in contrast to the non-responsiveness problem, dealing with the asynchronous nature of networks has a real counterpart, and these two types of interactions with the adversary should not be confused.

In the literature, urgent requests and the non-responsiveness problem occur in many protocols and functionalities. Nevertheless, protocol designers frequently ignore this problem (see, e.g., [1, 4, 5, 13, 14, 18, 21, 25, 26, 30, 31]), i.e., they seem to implicitly assume that urgent request *are* answered immediately, probably, at least as far as ideal

functionalities are concerned, because their simulators promptly respond to these kinds of requests. As a result, protocols and ideal functionalities are underspecified and/or expose unexpected behavior, and thus, are not usable in other (hybrid) protocols, or security proofs of hybrid protocols are flawed (see §3).

Our contribution. In this paper, we propose a universal composability framework with the new concept of *responsive environments* and *adversaries*, which should be applicable to all existing UC-style models (see below). This framework completely avoids and, by this, solves the non-responsiveness problem as it guarantees that urgent requests *are* answered immediately. This really is the most obvious and most natural solution to the problem: there is no reason that protocol designers should have to take care of the non-responsiveness problem and its many negative consequences.

More specifically, the main idea behind our framework is as follows. When a protocol/ideal functionality sends what we call a *restricting* message to the adversary/environment on the network interface, then the adversary/environment is forced to be responsive, i.e., to reply with a valid response before sending any other message to the protocol. This requires careful definitions and non-trivial proofs to ensure that all properties and features that are expected in models for universal composition are lifted to the setting with responsive environments and adversaries.

By using our framework and concepts, protocols and ideal functionalities can be modeled in a very natural way: protocol designers can simply declare urgent requests to be restricting messages, which hence have to be answered immediately. This elegantly and completely solves the non-responsiveness problem. In particular, protocol designers no longer have to worry about this problem, and specifications of protocols and ideal functionalities are greatly simplified, as one can dispense with artificial solutions. In fact, as illustrated in §6, with our concepts we can easily fix existing specifications from the literature in which the non-responsiveness problem has not been dealt with properly or has simply been ignored as protocol designers often implicitly assumed responsiveness for urgent messages. In some cases, we can now even express certain functionalities in a natural and elegant way that could not be expressed before (see §3.2.2 and §6). Of course, with simplified and more natural functionalities and protocols, also security proofs become easier because the protocol designer does not have to consider irrelevant and unrealistic adversarial behavior and execution orders.

We emphasize that protocol designers must exercise discretion when using restricting messages: *such messages should be employed for meta-information used for modeling purposes only, and not for real network traffic, where immediate answers cannot be guaranteed in reality.*

We illustrate that our framework and concepts apply to existing models for universal composability. This is exemplified for three prominent models: UC [8], GNUC [19], and IITM [22, 24]. In the full version of this paper [6], we provide full proofs for the IITM model. In particular, we define all common notions of simulatability, including UC, dummy UC, strong simulatability, and blackbox simulatability with respect to responsive environments and adversaries, and show that all of these notions are equivalent. This result can be seen as a sanity check of our concepts, as it has been a challenge in previous models (see, e.g., the discussions in [20, 24]). We also prove in detail that all composition theorems from the original IITM model carry over to the IITM model with our concepts.

Related work. The concept of responsive adversary and environments is new and has not been considered before.

In [2], composition for restricted classes of environments is studied, motivated by impossibility results in UC frameworks and to weaken the requirements on realizations of ideal functionalities. In this setting, environments are restricted in that they may send only certain sequences of messages to the I/O interfaces of protocols and functionalities. These restrictions cannot express that urgent requests are answered immediately and also do not restrict adversaries in any way. Hence, this approach cannot be used to solve the non-responsiveness problem, which anyway was not the intention of the work in [2].

In the first version of his seminal work [8], Canetti introduced *immediate functionalities*. According to the definition (cf. page 35 of the 2001 version), an immediate functionality uses an *immediate adversary* to guarantee that messages are delivered immediately between the functionality and its dummy. To be more precise, an immediate functionality may force an immediate adversary to deliver a message to a specific dummy party within a single activation. This construct was necessary as in the initial version of Canetti's model, the ideal functionality could not directly pass an output to its dummy but had to rely on the adversary instead. In current versions of UC, the problem addressed by immediate adversaries has vanished completely because ideal functionalities can directly communicate with their dummies. Clearly, immediate adversaries do not address, let alone solve, the non/responsiveness problem, which is about immediate answers for certain request to the adversary on the network interface rather than between a functionality and its dummies.

Outline. In §2, we briefly recall basic terminology and notation. We observe in §3 that the non-responsiveness problem affects many protocols from the literature, with many papers ignoring the problem altogether, resulting in underspecified and ill-specified protocols and functionalities, that are thus hard to re-use. Furthermore, that section shows that properly taking this problem into consideration is quite difficult and does not have a simple and generally applicable solution. Our universal composability framework with responsive environments and adversaries is then presented in §4. This section is kept quite model independent to highlight the main new concepts and the fact that these concepts are not restricted to specific models. §5 then illustrates how our concepts can be implemented in the UC, GNUC, and IITM models. §6 shows how the problems with non-responsive environment and adversaries discussed in §3 can be avoided elegantly with our restricting messages and responsive environments/adversaries. We conclude in §7. Further details can be found in the full version of this paper [6]. In particular, as mentioned before, we provide full details for the IITM model with responsive environments and adversaries in the full version.

2 Preliminaries

In this section, we briefly recap the basic concepts of universal composability and fix some notation and terminology. The description is independent of the model being used and can easily be mapped to any concrete model, such as UC, GNUC, or IITM. For now, we ignore runtime issues as they are handled differently in the models and only implicitly assume that all systems run in polynomial time in the security parameter and the length of the external input (if any). Runtime issues are discussed in detail in §5.

Universal composability models use *machines* to model programs. Each machine may have I/O and network tapes/interfaces. These machines are then used as blueprints to create *instances* which execute the machine code while having their own local state. Machines can be combined into a *system* $\mathcal{S}$. In a run of $\mathcal{S}$, multiple instances of machines may be generated and different instances can communicate by sending messages via I/O or network interfaces. Given two systems $\mathcal{R}$ and $\mathcal{Q}$, we define the system $\{\mathcal{R}, \mathcal{Q}\}$ which contains all machines from $\mathcal{R}$ and $\mathcal{Q}$.

There are three different kinds of entities, which can themselves be considered as systems and which can be combined to one system: *protocols*, *adversaries*, and *environments*. One distinguishes *real* and *ideal* protocols, where ideal protocols are often called *ideal functionalities*. An ideal protocol can be thought of as the specification of a task, whereas a real protocol models an actual protocol that is supposed to realize the ideal protocol (cf. Definition 2.1). These protocols have an I/O interface to communicate with the environment and a network interface to communicate with the adversary. An *adversary* controls the network communication of protocols and can also interact with the environment. *Environments* connect to the I/O interface of protocols and may communicate with the adversary, cf. Figure 1 for an illustration of how environments, adversaries, and protocols are connected.

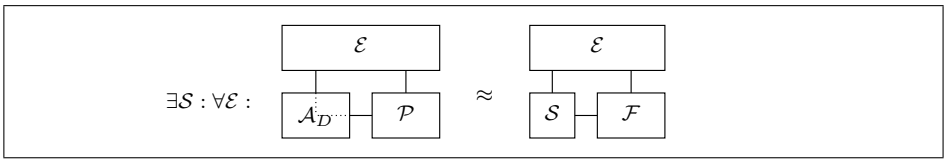

Fig. 1: A real protocol $\mathcal{P}$ realizing an ideal functionality $\mathcal{F}$; $\mathcal{A}_D$ denotes the dummy adversary which just forwards messages to and from the environment $\mathcal{E}$.

Environments try to distinguish whether they run with a real protocol and an adversary or an ideal protocol and an adversary (then often called a simulator or ideal adversary). An environment may get some external input to start a run. It is expected to end the run by outputting a single bit.

Given an environment $\mathcal{E}$, an adversary $\mathcal{A}$, and a protocol $\mathcal{P}$, we denote both the combined system and the output distribution of the environment by $\{\mathcal{E}, \mathcal{A}, \mathcal{P}\}$. We use the binary operator $\equiv$ to denote two output distributions that are negligibly close in the security parameter η (and the external input, if any).

Now, in models for universal composability, the realization of an ideal protocol by a real protocol is defined as follows.

Definition 2.1 (Realization Relation). *Let $\mathcal{P}$ and $\mathcal{F}$ be protocols, the* real *and* ideal protocol*, respectively. Then, $\mathcal{P}$* realizes *$\mathcal{F}$ ($\mathcal{P} \leq \mathcal{F}$) if for every adversary $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that $\{\mathcal{E}, \mathcal{A}, \mathcal{P}\} \equiv \{\mathcal{E}, \mathcal{S}, \mathcal{F}\}$ for every environment $\mathcal{E}$.*

We note that, in the definition above and in all reasonable models, instead of quantifying over all adversaries, it suffices to consider only the dummy adversary $\mathcal{A}_D$ which forwards all network messages between $\mathcal{P}$ and $\mathcal{E}$. Intuitively, this is true because $\mathcal{A}$ can be

subsumed by $\mathcal{E}$. Hence, we have that $\mathcal{P} \leq \mathcal{F}$ iff there exists an ideal adversary $\mathcal{S}$ such that $\{\mathcal{E}, \mathcal{A}_D, \mathcal{P}\} \equiv \{\mathcal{E}, \mathcal{S}, \mathcal{F}\}$ for every environment $\mathcal{E}$.

The main result in any universal composability model is a composition theorem. Informally, once a protocol $\mathcal{P}$ has been proven to realize an ideal protocol $\mathcal{F}$, one can securely replace (all instances of) $\mathcal{F}$ by $\mathcal{P}$ in arbitrary higher-level systems without affecting the security of the higher-level system.

3 The Non-Responsiveness Problem and its Consequences: Examples from the Literature

We have already introduced and discussed the non-responsiveness problem and sketched its consequences in §1. In this section, we illustrate this problem and its consequences by examples from the literature. We also point to concrete cases in which this problem has been ignored (i.e., immediate answers to urgent requests were assumed implicitly) and where this has led to ill-defined protocols and functionalities as well as invalid proofs and statements.

3.1 Underspecified and Ill-Defined Protocols and Functionalities

In many papers, the non-responsiveness problem is ignored in the specifications of both ideal functionalities and (higher-level) protocols (see, e.g., [1, 4, 5, 13, 14, 18, 21, 25, 26, 30, 31]). We discuss a number of typical cases in the following.

Ideal functionalities. An example of a statement that one often finds in specifications of ideal functionalities is one like the following (see, e.g., [1, 4, 13, 18, 21, 25, 26]):

$$\begin{array}{c}\text{“send <some message> to the adversary;}\\ \text{upon receiving <some answer> from the adversary do <something>”,}\end{array} \tag{1}$$

where the message sent to the adversary, in our terminology, is an urgent request, i.e., as explained in §1, some meta-information provided to the adversary or a request for some meta-information the adversary is supposed to provide. For example, ideal functionalities might ask for cryptographic material (cryptographic algorithms and keys, ciphertexts, or signatures), ask whether the adversary wants to corrupt a party, or simply signal their existence.

In specifications containing formulations as in (1) it is not specified what happens if the adversary does not respond immediately, but, for example, other requests on the I/O interface are received; intermediate state changes in other parts might also occur, which might require different actions. There does not seem to exist a generic solution to handle such problems (see §3.2.1 and §3.2.3). It rather seems to be necessary to find solutions tailored to the specific protocol and ideal functionality at hand, making it even more important to precisely specify the behavior in case the adversary does not respond immediately to urgent requests.

Many research papers on universal composability focus on proposing new functionalities and realizations thereof, including proofs that a realization actually realizes a functionality; to a lesser extent the functionalities are then used in higher-level protocols. In realization proofs, one might not notice that formulations as that in (1) are problematic because for such proofs an ideal functionality $\mathcal{F}$ runs alongside a (friendly) simulator and this simulator might provide answers to urgent requests immediately (see

also Figure 1). However, if used in a hybrid protocol (see Figure 2), an ideal functionality $\mathcal{F}$ runs alongside a (hostile) adversary/environment. In this case, it is important that specifications capture the case that urgent requests are not answered immediately. If this is ignored or not handled correctly, it yields i) underspecified protocols, with the problem that they cannot be re-used in hybrid protocols, which in turn defeats the purpose of universal composability frameworks, and ii) possibly false statements.

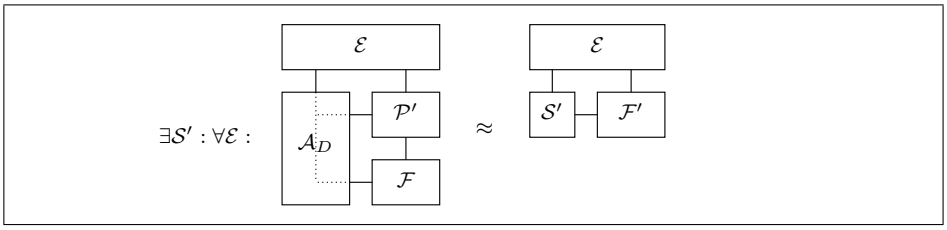

Fig. 2: An $\mathcal{F}$-hybrid protocol $\mathcal{P}'$ realizing some ideal functionality $\mathcal{F}'$.

To illustrate these points by a concrete example, we consider the "signature of knowledge" functionality $\mathcal{F}_{\text{sok}}(L)$ proposed by Chase and Lysyanskaya [14]. This functionality contains a Setup instruction (reproduced in Figure 3), where the adversary provides the keys and algorithms, and signing and verification instructions that then use those keys and algorithms without requiring interaction with the adversary - a very common mechanism in the literature (see, e.g., [1,5,12,15,16,30]). This functionality is explicitly intended to be used in a hybrid setting to realize delegatable credentials.

Upon receiving a value $(\texttt{Setup}, sid)$ from any party P, verify that $sid = (M_L, sid')$ for some sid'. If not, then ignore the request. Else, if this is the first time that $(\texttt{Setup}, sid)$ was received, **hand** $(\texttt{Setup}, sid)$ **to the adversary; upon receiving** $(\texttt{Algorithms}, sid, \mathsf{Verify}, \mathsf{Sign}, \mathsf{Simsign}, \mathsf{Extract})$ **from the adversary**, where Sign, Simsign, Extract are descriptions of PPT TMs, and Verify is a description of a deterministic polytime TM, store these algorithms. Output the stored $(\mathsf{Algorithms}, sid, \mathsf{Sign}, \mathsf{Verify})$ to P.

Fig. 3: The Setup instruction of $\mathcal{F}_{\text{sok}}(L)$ from [14].

If the adversary does not respond to the first $(\texttt{Setup}, sid)$ request, all subsequent requests (e.g., a Setup request by a different party) will cause the functionality to use or output the undefined Sign and Verify algorithms, which is a problem: Chase and Lysyanskaya provide a protocol in the $\mathcal{F}_{\text{sok}}(L)$-hybrid model that can be used for realizing delegated credentials, i.e., an ideal functionality for signatures on a signature. They then prove that this protocol realizes the functionality. They, however, missed the fact that $\mathcal{F}_{\text{sok}}(L)$ may interact with a non-responsive adversary in the hybrid world. Such an adversary can force $\mathcal{F}_{\text{sok}}(L)$ to use undefined algorithms, and their simulator does not handle that situation in the ideal world. It is thus easy to distinguish the real from the

ideal world. Hence, their proof is flawed, and in fact it seems that the statement cannot be proven.

(Higher-level) protocols. As already mentioned in the introduction, real protocols often also send urgent requests to the adversary (e.g., signaling their existence or asking whether the adversary wants to corrupt). In addition, one often finds protocol specifications containing formulations of the following form to make requests to subprotocols (see, e.g., [5, 14, 30, 31]):

$$\begin{array}{c}\text{“send <some message> to } \mathcal{F}\text{;}\\ \text{upon receiving <some answer> from } \mathcal{F} \text{ do <something>.”}\end{array} \tag{2}$$

Intuitively, $\mathcal{F}$ might indeed model some non-interactive functionality, such as signature functionalities. However, because of the use of urgent requests in such functionalities, even when completely uncorrupted, $\mathcal{F}$ might not return answers right away. So, again, formulations as the one in (2) are greatly underspecified. What happens if other requests are received at the network or I/O interface? Should they be ignored? Or may they be queued somehow? Also, the state and status (such as corruption) of other parts of the protocol or subprotocols might change while waiting for answers from $\mathcal{F}$. Again, as illustrated in the following subsections, dealing with this is not easy and often requires solutions tailored to the specific protocol and functionality at hand, making a full specification of the behavior particularly important.

3.2 Problems Resulting from Non-Responsiveness

We now discuss challenges resulting from the non-responsiveness problem (when actually taken into account, rather than ignored) and illustrate them by examples from the literature.

3.2.1 Unintended State Changes and Race Conditions

As mentioned before, a general problem one has to take care of when dealing with the non-responsiveness problem is that while a protocol is waiting for answers to urgent requests, the adversary might cause changes in the state of other parts of the protocol/functionality and of subprotocols, which in turn influences the behavior of the protocol. Keeping track of the actual current overall state might be tricky, and race conditions are possible.

The following is a simple example which illustrates that the problem can occur already locally within a single functionality. It can often become even trickier in higher-level protocols which use urgent requests themselves and where possibly several subroutines use urgent requests.

We consider the dual-authentication certification functionality $\mathcal{F}_{\text{D-Cert}}$ [31]. In this functionality, the adversary needs to be contacted when verifying a signature (a common mechanism to verify cryptographic values that is also used in many other functionalities [7, 13, 21]). Such requests are urgent as this is supposed to model local computations. However, the adversary may not answer immediately.

More specifically, Figure 4 shows the Verify instruction of $\mathcal{F}_{\text{D-Cert}}$. Assume now that S' has received a message m and a signature σ for this message, which supposedly was created by an honest party P with SID sid. Now, if the signature actually was not

Upon receiving a value $(\texttt{Verify}, sid, m, \sigma)$ from some party S', **hand** $(\texttt{Verify}, sid, m, \sigma)$ **to the adversary**. **Upon receiving** $(\texttt{Verified}, sid, m, \phi)$ **from the adversary**, do:

1. If $(m, \sigma, 1)$ is recorded then set $f = 1$.
2. Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.
3. Else, if there is an entry (m, σ, f') recorded, then set $f = f'$.
4. Else, set $f = \phi$, and record the entry (m, σ, ϕ).

Output $(\texttt{Verified}, sid, m, f)$ to S'.

Fig. 4: The Verify instruction of $\mathcal{F}_{\text{D-Cert}}$ from [31].

created by P, the verification should fail as P is not corrupted. However, as the adversary gets activated during this allegedly local task, it could corrupt the signer during the verification process, return $\phi = 1$, and therefore let the functionality *accept* σ. This behavior is certainly unexpected and counterintuitive.

Such a functionality also considerably complicates the security analysis of any higher-level application that uses $\mathcal{F}_{\text{D-Cert}}$ as a subroutine, as one has to also consider the possiblity of a party getting corrupted during the invocation of a subroutine modeling a local task, which, even worse, in that case returns unexpected answers.

3.2.2 Problems Expressing the Desired Properties

The following is an example where the authors struggled with the non-responsiveness problem in that it finally led to a functionality that, as the authors acknowledge, is not completely satisfying. This functionality, denoted $\mathcal{F}_{\text{NIKE}}$, is supposed to model a non-interactive key exchange and was proposed by Freire et al. [17]. Figure 5 shows a central part of this functionality, namely, the actual key exchange. A party P_i may ask for the key that is shared between the parties P_i and P_j. If this session of P_i and P_j is considered corrupted, namely, because one of the parties is corrupted, and no key has been recorded for this session yet, the adversary is allowed to freely choose the key that is shared between the two parties. The functionality uses an urgent request to model this, i.e., it directly sends a message to the adversary if she is allowed to choose a key.

Upon input $(\texttt{init}, P_i, P_j)$ from P_i, if $P_j \notin \Lambda_{\text{ref}}$, return $(P_i, P_j, \perp)$ to P_i. If $P_j \in \Lambda_{\text{reg}}$, we consider two cases:

- Corrupted session mode: if there exists an entry $(\{P_i, P_j\}, K_{i,j})$ in Λ_{keys}, set $key = K_{i,j}$. Else, **send** $(\texttt{init}, P_i, P_j)$ **to the adversary. After receiving** $(\{P_i, P_j\}, K_{i,j})$ **from the adversary**, set $key = K_{i,j}$ and add $(\{P_i, P_j\}, K_{i,j})$ to Λ_{keys}.
- Honest session mode: if there exists an entry $(\{P_i, P_j\}, K_{i,j})$ in Λ_{keys}, set $key = K_{i,j}$, else choose $key \leftarrow \{0,1\}^k$ and add $(\{P_i, P_j\}, K_{i,j})$ to Λ_{keys}.

Return (P_i, P_j, key) to P_i.

Fig. 5: The init instruction of $\mathcal{F}_{\text{NIKE}}$ from [17].

As the authors state, they would have liked to also model "immediateness" of the functionality, i.e., a higher-level protocol that requests a key should be able to expect an answer without the adversary being able to interfere with the protocol in the meantime. This indeed would be expected and natural because $\mathcal{F}_{\mathrm{NIKE}}$ models a *non-interactive* key exchange. However, this is in conflict with allowing the adversary to choose the key of a corrupted session. The authors suggest that one option to also model immediateness might be to let the adversary choose an algorithm upon setup, which is then used to compute the keys for corrupt parties. Nevertheless, they chose the non-immediate modeling because the other solution would lead to "technical complications"; it would also limit the adaptiveness of the adversary and might add other problems. Indeed, code upload constructs (see also §3.2.3), in general, do not solve the non-responsiveness problem.

As a consequence of the formulation chosen in $\mathcal{F}_{\mathrm{NIKE}}$, the adversary can now, e.g., block requests, which again also needs to be considered in any higher-level protocol using $\mathcal{F}_{\mathrm{NIKE}}$ as a subroutine, even though in the real world the honest party would always obtain some key because of the non-interactivity of the primitive.

More generally, ideal functionalities that use urgent messages (which in current models are not answered immediately) might have weaker security guarantees than their realizations, in particular when the functionality is supposed to model a non-interactive task, because the realization might not give control to the adversary. So for hybrid protocols one might not be able to prove certain properties when using an ideal functionality, whereas the same protocol using the realization of the ideal functionality instead might enjoy such properties.

This is in contrast to one of the goals of universal composability models, namely, reducing the complexity of security analyses by enabling the use of conceptually simpler ideal functionalities as subroutines.

3.2.3 The Reentrance Problem

As already mentioned in §1, a protocol designer has to specify the behavior of protocols and ideal functionalities upon receiving another input (on the I/O interface) while they are waiting for a response to an urgent request on the network. In other words, protocols and ideal functionalities have to be *reentrant*. Note that, as pointed out, a protocol has to be reentrant not only when it uses urgent requests itself, but also if a subroutine uses such messages.

As explained next, dealing with the reentrance problem can be difficult. Approaches to solve this problem complicate the specifications of protocols and ideal functionalities, and none of them is sufficiently general to be applicable in every case.

We now illustrate this by an example ideal functionality. However, similar issues occur in specifications for real and hybrid protocols. Let $\mathcal{F}$ be any ideal functionality which sends an urgent request to the adversary upon its first creation, say, to retrieve some modeling-related information. This is a common situation. For example, ideal functionalities often require some cryptographic material such as keys and algorithms from the adversary before they can continue their execution (e.g., functionalities for digital signatures or public-key encryption). We also assume that $\mathcal{F}$ is meant to be realized by a real protocol consisting of two independent parties/roles A and B (e.g., signer and verifier). We further assume that both of these parties also send an urgent

request to the adversary upon their first activation and expect an answer before they can continue with their computation. Again, this is a common situation as, for example, real protocols often ask for their corruption status or notify the adversary of their creation.[5] While the above is only one illustrative example, it already describes a large and common class of real and ideal protocols often encountered in the literature.

We now present several approaches to make $\mathcal{F}$ reentrant in the above sense, i.e., to deal with I/O requests while waiting for a response to an urgent request on the network. We show that the obvious approaches in general cannot be used. In particular, with most of these approaches $\mathcal{F}$ cannot even be realized by A and B in the setting outlined above. This in turn shows that solutions that are tailored to the specific functionality at hand and even the envisioned realization are required, which is very unsatisfactory, as this leads to more complex and yet less general functionalities and protocols.

Ignore requests. After sending an urgent request to the adversary, the most straightforward approach would be to ignore all incoming messages until a response from the adversary is received.[6] This, however, is not only an unexpected behavior in many cases – for example, why should a request silently fail if the ideal functionality models a local computation? – but the ideal functionality in fact might no longer be realizable by some real protocols:

If $\mathcal{F}$, in our example functionality, would simply ignore incoming messages, an environment can distinguish $\mathcal{F}$ (with a simulator) from the realization A and B (with the dummy adversary). It first sends a message to A which, as we assume, then in turn sends an urgent request to the dummy adversary and hence to the environment. Now the environment, which does not have to respond to urgent requests immediately, sends a message to B which in turn also sends an urgent request to the adversary and hence to the environment. Consider the behavior of the ideal world in this case: After receiving the message for A, $\mathcal{F}$ will send an urgent request to the simulator. The simulator, however, cannot answer this urgent request because it has to simulate A by sending an urgent request to the environment. (This might be the case because the simulator first has to consult the environment before answering the urgent request by $\mathcal{F}$ or because $\mathcal{F}$ does not return control to the simulator after receiving an answer to the urgent request.) The environment then sends the second message (for B) to $\mathcal{F}$, which is ignored because $\mathcal{F}$ still waits for an answer to its urgent request. This behavior is different from the real world, and thus, the environment can distinguish the real world from the ideal one.

This illustrates that an ideal functionality that simply blocks *all* requests while waiting for a response to an urgent request can in general not be realized by two or more *independent* parties that also send urgent requests to the adversary. Instead one needs to adjust the blocking approach to the specific protocols at hand. For example, often it might be possible to block messages that would be processed by a single party in the real protocol, while messages for other parties are still processed. But this does not work if, for instance, $\mathcal{F}$ cannot process messages for any party before receiving a response to its

[5] The latter is, for example, required by the definition of "subroutine respecting protocols" in the 2013 version of UC [8]. While prompt responses by the adversary are formally not required, they would be very convenient for all of the reasons discussed in §3.2.

[6] Alternatively, one could send error messages as response to intermediate requests. However, the exact same problems discussed for the approach of ignoring requests occur.

urgent requests, e.g., because $\mathcal{F}$ first needs to receive cryptographic material (algorithms, keys, etc.). Thus, in this case yet another workaround is required.

Queuing of intermediate requests. Another potential general approach to deal with the reentrance problem is to store all incoming messages to process them later on. The simplest implementation of this approach would be the following: Upon receiving another input while still waiting for a response to an urgent request, the ideal functionality stores the input in a queue and then ends its activation. After receiving a response from the adversary, the ideal functionality processes the messages stored in the queue.

This approach is vulnerable to the same attack as the previous approaches: if the environment executes this attack in the real world, it will eventually receive an urgent request from B. This, however, cannot be simulated in the ideal world. The simulator does not get control when B is activated as the ideal functionality simply ends its activation after queuing the input for B.

Another problem with this approach is that in all current universal composability models, a machine is allowed to send only one message per activation. Hence, the ideal functionality will never be able to catch up with the inputs that have been stored. Every time it is activated by another input, it will have to process both the new input and several older inputs that are still stored in the queue. But it can only answer one of these messages at a time. This observation leads to another approach based on the queuing of unanswered requests which we discuss in the full version of this paper [6]. This approach, which does not seem to have been used in the literature so far, is, however, very complex and weakens the security of the ideal functionality to an extent that for some tasks is unacceptable: it allows the adversary to determine the order in which requests are processed by an ideal functionality.

Further approaches. In the full version of this paper [6], we discuss several alternative approaches, namely, *default answers* and *code uploads*, which, however, can merely help reduce the use of urgent requests, but do not solve the reentrance problem, let alone the general non-responsiveness problem.

3.2.4 Unnatural Specifications of Higher-Level Protocols

Higher-level protocols have to deal with the non-responsiveness problem for two reasons. First, they might use urgent requests themselves. Second, subprotocols might use urgent requests, and hence, if requests are sent to subprotocols (even for those that intuitively should model non-interactive primitives), the adversary might get control. In both cases, higher-level protocols have to deal with the problem that while waiting for answers, the state of other parts of them and of any of their subprotocols might change and new requests (from the network or I/O interface) might have to be processed. This can lead to unnecessarily complex and often unnatural specifications, if the non-responsiveness problem is actually taken into account rather than being ignored (which in turn would result in underspecified, and hence, unusable protocols).

We illustrate this by a joint state realization, which represents one form of a higher-level protocol: Consider a digital signature functionality $\mathcal{F}_{\mathrm{sig}}$. Let us assume that $\mathcal{F}_{\mathrm{sig}}$ is specified in such a way that at the beginning it asks the adversary for signing and verification algorithms and keys before it answers other requests; as already mentioned, this is a very common design pattern. Because the adversary might not answer requests

for the cryptographic material right away (non-responsiveness), $\mathcal{F}_{\text{sig}}$ might receive further requests while waiting for the answer. Let us assume that $\mathcal{F}_{\text{sig}}$ ignores/drops all such requests (this seems to be the option mainly used in the literature, see, e.g., [4,23]).[7]

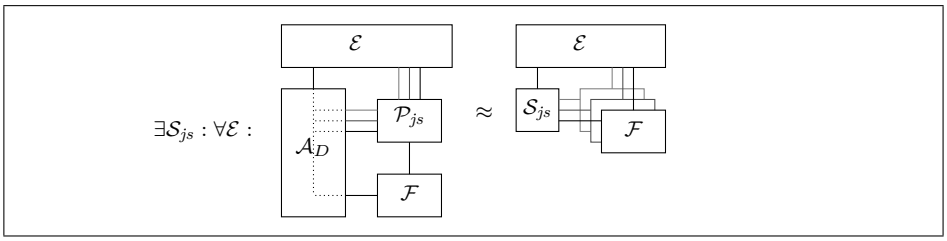

Fig. 6: Joint state realization.

In a joint state realization of $\mathcal{F}_{\text{sig}}$, one instance of $\mathcal{F}_{\text{sig}}$ (per party) is used to realize all sessions of $\mathcal{F}_{\text{sig}}$ (for one party) in the ideal world (see also Figure 6). The idea behind the joint state realization is that if in session sid a message m is to be signed/verified, then one would instead sign/verify the message (sid, m). In this way, messages of different sessions cannot interfere. In the realization proof, a simulator would provide an instance $\mathcal{F}_{\text{sig}}$ in session sid with a signing and verification algorithm that exactly mimics the behavior of $\mathcal{F}_{\text{sig}}$ in session sid (i.e., signing/verifying prefix messages with sid). Unfortunately, because of the non-responsiveness problem, the joint state realization is more complex than that, even if, for the purpose of the discussion, we ignore the handling of corruption. To see this, assume that the environment sends a signing request for some message m in session sid. The joint state realization would now invoke $\mathcal{F}_{\text{sig}}$ with (sid, m). Before $\mathcal{F}_{\text{sig}}$ can answer, $\mathcal{F}_{\text{sig}}$ asks the adversary for the cryptographic material. Hence, the adversary/environment gets activated again, and the environment can send a new, say, signing request for message m' in session sid'. As $\mathcal{F}_{\text{sig}}$ is still waiting for the adversary to provide the cryptographic material, this later request will be ignored by $\mathcal{F}_{\text{sig}}$ and hence will never be answered. To mimic this behavior in the ideal world, the simulator should not provide the cryptographic material to the instance of $\mathcal{F}_{\text{sig}}$ in session sid' (otherwise, $\mathcal{F}_{\text{sig}}$ in session sid' would return a signature for m'). But then, this instance of $\mathcal{F}_{\text{sig}}$ is blocked completely. Hence, in turn, the joint state realization also has to block all further requests for session sid'. That is, it has to store all SIDs for which it received requests while waiting for $\mathcal{F}_{\text{sig}}$ to respond, and all future requests for all such SIDs have to be dropped.

This is very unnatural and certainly would not correspond to anything one would do in actual implementations: there one would simply prefix messages with SIDs, but one would never block requests for certain SIDs. This is just an artifact of the non-responsiveness problem, i.e., the fact that, in current models, urgent requests (in this case the request for cryptographic material by $\mathcal{F}_{\text{sig}}$) might not be answered immediately.

[7] As explained in §3.2.3, this approach, just as all other approaches discussed in §3.2.3, does not work in general, e.g., when the signer and verifier are independent and send urgent requests to the adversary upon first activation. It really depends on the details of $\mathcal{F}_{\text{sig}}$ and its realization.

4 Universal Composability with Responsive Environments

The non-responsiveness problem and the resulting complications shown in §3 are artificial problems. As urgent requests exist only for modeling purposes but do not model any real network traffic, a real adversary would not be able to use them to carry out attacks. Still, in all current universal composability models, the non-responsiveness of adversaries enables attacks that do not correspond to anything in reality. If we could force the adversary to answer urgent requests immediately, which, as already mentioned before, would be the natural and expected behavior, there would not be any need for coming up with workarounds that try to solve the non-responsiveness problem in the specifications of protocols and functionalities and one would not have to consider such artificial attacks in security proofs.

In this section, we present our framework which extends universal composability models by allowing protocol designers to specify messages that have to be answered immediately by (responsive) environments and adversaries. We first give a brief overview of our approach, then define in more detail responsive environments, responsive adversaries and the realization relation in this setting, and finally prove that the composition theorems still hold for our extension. As our framework and concepts can be used by any universal composability model and to highlight the new concepts, we keep this section independent of specific models. In particular, we mostly ignore runtime considerations. In §5, we then discuss in detail how our framework can be adapted to specific models.

4.1 Overview

To avoid the non-responsiveness problem altogether, we introduce the concept of *responsive environments* and *responsive adversaries*. In a nutshell, when these environments and adversaries receive specific messages from the network (we call these messages *restricting*) then they have to respond to these messages immediately, i.e., without activating other parts of the protocol before sending an answer. Furthermore, depending on the restricting message, they may send an answer from a specific set of messages only. Restricting messages and the possible answers can be specified by the protocol designer; they are not hardwired into the framework. More specifically, restricting messages and the possible responses are specified by a binary relation $R \subseteq \{0,1\}^+ \times \{0,1\}^+$ over non-empty messages, called a *restriction*. If $(m, m') \in R$, then m is a restricting message and m' a possible answer to m. That is, if an environment/adversary receives m on its network interface, then it has to answer immediately with some m' such that $(m, m') \in R$.

This allows a protocol designer to specify all urgent requests as restricting messages by defining a restriction R appropriately; such requests are then answered not only immediately but also with an expected answer. Therefore the adversary can no longer interfere with the protocol run in an unintended way by activating other parts of the protocol or sending unexpected inputs before answering an urgent request.

Note that this concept is very powerful and needs to be handled with care: While, as motivated above, it does not weaken security results if one models urgent requests as restricting messages, one must not use such messages when modeling real network traffic, as real network messages are not guaranteed to be answered immediately in reality.

4.2 Defining Responsiveness

To define responsive environments and responsive adversaries, we first precisely define the notion of a restriction. As mentioned, restrictions are used to define both restricting messages, which have to be answered immediately by the environment/adversary, and possible answers to each restricting message.

Definition 4.1. *A* restriction R *is a set of pairs of non-empty messages, i.e.,* $R \subseteq \{0,1\}^+ \times \{0,1\}^+$*, such that, given a pair of messages* (m, m')*, it is efficiently decidable whether* R *allows* m' *as an answer to* m*. We define* $R[0] := \{m | \exists m' : (m, m') \in R\}$*. A message* $m \in R[0]$ *is called a* restricting message*.*

The idea is that if an environment/adversary receives m on the network interface, there are two cases: If m is not a restricting message, i.e., $m \notin R[0]$, then the environment/adversary is not restricted in any way. Otherwise, if $m \in R[0]$, then the first message (if any) sent back to the protocol (both on the network and I/O interface of the protocol) has to be some message m' with $(m, m') \in R$. This message has to be sent on the network interface of the same machine that issued the request m, without any other message being sent to another machine of the protocol (see also Definition 4.2).

By requiring efficient decidability we ensure that environments are able to check whether some answer is allowed by the restriction; this is necessary, e.g., for Lemma 4.4. We refer to §5 for the exact definitions of "efficiently decidable", which depend on the runtime definitions of the underlying models.

As mentioned in §4.1, only urgent requests should be defined as restricting messages via a restriction. For example, upon creation of a new instance by receiving a message m, instances of protocols are often expected to first ask the adversary whether they are corrupted before they process the message m. An adversary can be forced to answer such a request immediately by the following restriction:

$$R := \{(m, m') | m = \texttt{AmICorrupted?}, m' = (\texttt{Corruption}, b), b \in \{\mathsf{false}, \mathsf{true}\}\}.$$

We now formalize the responsiveness property of environments and adversaries.

Definition 4.2 (Responsive Environments). *An environment* $\mathcal{E}$ *is called* responsive *for a system of machines* $\mathcal{Q}$ *with respect to a restriction* R *if in an overwhelming set of runs of* $\{\mathcal{E}, \mathcal{Q}\}$ *every restricting message from* $\mathcal{Q}$ *on the network is answered correctly, i.e., for any restricting message* $m \in R[0]$ *sent by* $\mathcal{Q}$ *on the network, the first message* m' *that* $\mathcal{Q}$ *receives afterwards (be it on the network interface or the I/O interface of* $\mathcal{Q}$*), if any, is sent by* $\mathcal{E}$ *on the network interface of* $\mathcal{Q}$ *to the same machine of* $\mathcal{Q}$ *that sent* m *and* m' *satisfies* $(m, m') \in R$*. By* $\mathsf{Env}_R(\mathcal{Q})$ *we denote the set of responsive environments for* $\mathcal{Q}$*.*

In the above definition, "same machine" typically means the same *instance* of a machine. So if an instance of a machine of $\mathcal{Q}$ sent a restricting message m on the network interface to the environment, the first message m' received by any instance of $\mathcal{Q}$ (on the network or I/O interface), including all currently running instances of $\mathcal{Q}$ and an instance that might be created as a result of m', has to be sent back on the network interface to the same instance of $\mathcal{Q}$ which sent m, and m' has to satisfy $(m, m') \in R$. The exact definition of "same machine" depends on the model under consideration (see §5).

The system $\mathcal{Q}$ usually is either $\{\mathcal{A}_D, \mathcal{P}\}$, where $\mathcal{P}$ is a real protocol and $\mathcal{A}_D$ is the dummy adversary, or $\{\mathcal{S}, \mathcal{F}\}$, where $\mathcal{S}$ is an ideal adversary and $\mathcal{F}$ is an ideal protocol.

Responsive adversaries have to provide the same guarantees as responsive environments; however, they have to do so only when running in combination with a responsive environment. In other words, they can use the responsiveness property of the environment to ensure their own responsiveness property.

Definition 4.3 (Responsive Adversaries). *Let $\mathcal{Q}$ be a system and let $\mathcal{A}$ be an adversary that controls the network interface of $\mathcal{Q}$. Then, $\mathcal{A}$ is called a* responsive adversary *if, for all $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}, \mathcal{Q}\})$, in an overwhelming set of runs of $\{\mathcal{E}, \mathcal{A}, \mathcal{Q}\}$ every restricting message from $\mathcal{Q}$ on the network is immediately answered (in the sense of Definition 4.2). We denote the set of all such adversaries for a protocol $\mathcal{Q}$ by $\mathrm{Adv}_R(\mathcal{Q})$.*

We note that the dummy adversary $\mathcal{A}_D$ is responsive.

Also note that the definitions of both responsive environments and responsive adversaries depend on a specific system, i.e., an environment which is responsive for a system $\mathcal{Q}$ is not necessarily responsive for a system $\mathcal{Q}'$. If we required environments to be responsive for *every* system, we would also have to require this from simulators (ideal adversaries). This in turn would needlessly complicate security proofs. Let us elaborate on this. Many theorems and lemmas in UC-like models, such as transitivity of the realization relation (cf. Lemma 4.7) and the composition theorems (cf. Theorem 4.8 and Theorem 4.9), are proven by simulating (some instances of) adversaries/simulators and protocols within the environment. In such proofs, we need to make sure that if an environment is responsive, then it is still responsive if we move a simulator (ideal adversary) into the environment, i.e., run the simulator within the environment. Now, if we require strong responsiveness (i.e., responsiveness for all systems), then moving a simulator into a responsive environment might result in an environment that is no longer responsive (in the strong sense), unless we require from the simulator that it is responsive in the strong sense as well. However, imposing such a strong requirement on simulators seems unreasonable. Simulators are constructed in security proofs to work with exactly one protocol. So a protocol designer should only have to care about runs with this specific protocol, not with arbitrary systems that might try to actively violate the responsiveness property of the simulator. This is why we require responsiveness for specific systems only and this indeed is sufficient.

In fact, for security proofs, there are two important properties that should be fulfilled and for which we now show that they are. The first says that if an environment is responsive for one system, then it is also responsive for any system indistinguishable from that system. The second property says that a responsive environment can internally simulate a responsive adversary/simulator without losing its responsiveness property. In other words, we can move a responsive adversary/simulator into a responsive environment without losing the responsiveness property of the environment. As mentioned before, this is necessary, for example, for the transitivity of the realization relation and the composition theorems.

Lemma 4.4. *Let R be a restriction. Let $\mathcal{Q}$ and $\mathcal{Q}'$ be two systems of machines such that $\{\mathcal{E}, \mathcal{Q}\} \equiv \{\mathcal{E}, \mathcal{Q}'\}$ for all $\mathcal{E} \in \mathsf{Env}_R(\mathcal{Q})$. Then, $\mathsf{Env}_R(\mathcal{Q}) = \mathsf{Env}_R(\mathcal{Q}')$.*

For the proof of this lemma, we refer the reader to the full version of this paper [6].

Lemma 4.5. *Let R be a restriction. Let $\mathcal{Q}$ be a system, $\mathcal{A} \in \mathrm{Adv}_R(\mathcal{Q})$ be a responsive adversary, and $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}, \mathcal{Q}\})$ be a responsive environment. Let $\mathcal{E}'$ denote the environment that internally simulates the system $\{\mathcal{E}, \mathcal{A}\}$. Then, $\mathcal{E}' \in \mathsf{Env}_R(\mathcal{Q})$.*

For the proof of this lemma, we refer the reader to the full version of this paper [6].

4.3 Realization Relation for Responsive Environments

We can now define the realization relation for responsive environments. The definition is analogous to the one for general environments and adversaries (see Definition 2.1), but restricts these entities to being responsive.

Definition 4.6 (Realizing Protocols with Responsive Environments). *Let $\mathcal{P}$ and $\mathcal{F}$ be protocols, the* real *and* ideal protocol*, respectively, and R be a restriction. Then, $\mathcal{P}$* realizes *$\mathcal{F}$ with respect to responsive environments $(\mathcal{P} \leq_R \mathcal{F})$ if for every responsive adversary $\mathcal{A} \in \mathrm{Adv}_R(\mathcal{P})$, there exists an (ideal) responsive adversary $\mathcal{S} \in \mathrm{Adv}_R(\mathcal{F})$ such that $\{\mathcal{E}, \mathcal{A}, \mathcal{P}\} \equiv \{\mathcal{E}, \mathcal{S}, \mathcal{F}\}$ for every environment $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}, \mathcal{P}\})$.*

Just as in the case of Definition 2.1, we have that instead of quantifying over all responsive adversaries, it suffices to consider only the dummy adversary $\mathcal{A}_D$, which forwards all network messages between $\mathcal{P}$ and $\mathcal{E}$ (we provide a formal proof in the full version of this paper [6]). As already mentioned, $\mathcal{A}_D$ is always responsive. This means that in security proofs, one has to construct only one responsive simulator $\mathcal{S}$ for $\mathcal{A}_D$.

As mentioned before Lemma 4.5, the responsiveness of $\mathcal{S}$ is necessary for the transitivity of $\leq_R$. While the responsiveness of $\mathcal{S}$ is a property a protocol designer has to ensure, this property is easy to check and guarantee: upon receiving a restricting message from the protocol, it either answers immediately and correctly or sends only restricting messages to the environment until it can provide a correct answer to the original restricting message from the protocol. In such a situation, the simulator should not send a non-restricting message to the environment because, if it does so, it cannot make sure that it gets back an answer immediately from the environment and that the environment does not invoke the protocol in between. In the full version of this paper [6], we specify and provide a formal proof of this intuition.

We also note that Definition 4.6 is a generalization of Definition 2.1: with $R := \emptyset$, we obtain Definition 2.1.

We now prove that the realization relation with responsive environments is reflexive and transitive. This is crucial for the modular and step-wise design of protocols: once we have proven $\mathcal{P} \leq_R \mathcal{P}'$ and $\mathcal{P}' \leq_R \mathcal{P}''$, we want to conclude immediately that $\mathcal{P} \leq_R \mathcal{P}''$.

Lemma 4.7. *The $\leq_R$ relation is reflexive and transitive.*

For the proof of this lemma, we refer the reader to the full version of this paper [6].

4.4 Composition Theorems

The core of every universal composability model is the composition theorems. We now present a first composition theorem that handles concurrent composition of any (fixed) number of potentially different protocols.

Theorem 4.8. *Let R be a restriction. Let $k \geq 1$, $\mathcal{Q}$ be a protocol, and $\mathcal{P}_1, \ldots, \mathcal{P}_k$, $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be protocols such that for all $j \leq k$ it holds true that $\mathcal{P}_j \leq_R \mathcal{F}_j$.*
Then, $\{\mathcal{Q}, \mathcal{P}_1, \ldots, \mathcal{P}_k\} \leq_R \{\mathcal{Q}, \mathcal{F}_1, \ldots, \mathcal{F}_k\}$.

Proof. In what follows, we take the (equivalent) formulation of $\leq_R$ with the dummy adversary $\mathcal{A}_D$.

It suffices to prove the theorem for the case $k = 1$. The argument can then be iterated to obtain the theorem for $k > 1$ using transitivity of the $\leq_R$ relation. Let $\mathcal{S} \in \mathrm{Adv}_R(\mathcal{F}_1)$ be the simulator from the definition of $\mathcal{P}_1 \leq_R \mathcal{F}_1$. Define the simulator $\mathcal{S}'$ to forward messages between the environment and $\mathcal{Q}$, while internally simulating $\mathcal{S}$ for messages between the environment and $\mathcal{F}_1$. Now let $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}_D, \mathcal{Q}, \mathcal{P}_1\})$. For convenience, in what follows, we split $\mathcal{A}_D$ into $\mathcal{A}_D^{\mathcal{Q}}$ and $\mathcal{A}_D^{\mathcal{P}_1}$ where $\mathcal{A}_D^{\mathcal{Q}}$ forwards all communication betweeen $\mathcal{E}$ and $\mathcal{Q}$ and $\mathcal{A}_D^{\mathcal{P}_1}$ forwards all communication betweeen $\mathcal{E}$ and $\mathcal{P}_1$.

We first prove that $\{\mathcal{E}, \mathcal{A}_D, \mathcal{Q}, \mathcal{P}_1\} \equiv \{\mathcal{E}, \mathcal{S}', \mathcal{Q}, \mathcal{F}_1\}$. Suppose that this is not the case. Then we can define a new environment $\mathcal{E}'$ that distinguishes $\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\}$ from $\{\mathcal{S}, \mathcal{F}_1\}$. The environment $\mathcal{E}'$ internally simulates $\{\mathcal{E}, \mathcal{A}_D^{\mathcal{Q}}, \mathcal{Q}\}$, and hence, distinguishes with the same probability as $\mathcal{E}$. Now observe that $\mathcal{E}'$ is responsive for $\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\}$: All network messages from $\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\}$ in $\{\mathcal{E}, \mathcal{A}_D, \mathcal{Q}, \mathcal{P}_1\}$ are handled by $\mathcal{E}$ only, not by $\mathcal{Q}$. Moreover, as $\mathcal{E}$ is responsive for $\{\mathcal{A}_D, \mathcal{Q}, \mathcal{P}_1\}$, we have that these messages are answered correctly (in the sense of Definition 4.2), implying the responsiveness of $\mathcal{E}'$ for $\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\}$. This contradicts the assumption that $\mathcal{P}_1 \leq_R \mathcal{F}_1$, and hence $\{\mathcal{E}, \mathcal{A}_D, \mathcal{Q}, \mathcal{P}_1\} \equiv \{\mathcal{E}, \mathcal{S}', \mathcal{Q}, \mathcal{F}_1\}$ must be true.

We still have to show the responsiveness property of $\mathcal{S}'$, that is, $\mathcal{S}' \in \mathrm{Adv}_R(\{\mathcal{Q}, \mathcal{F}_1\})$. Let $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{S}', \mathcal{Q}, \mathcal{F}_1\})$. We have to show that all restricting network messages from $\mathcal{Q}$ and $\mathcal{F}_1$ to $\mathcal{E}$ and $\mathcal{S}'$ are answered correctly (in the sense of Definition 4.2). Suppose that there is a non-negligible set of runs of $\{\mathcal{E}, \mathcal{S}', \mathcal{Q}, \mathcal{F}_1\}$ in which a restricting network message from $\{\mathcal{Q}, \mathcal{F}_1\}$ is not answered correctly. As $\mathcal{S}'$ only forwards network messages from $\mathcal{Q}$ to the environment and the environment is responsive for $\{\mathcal{S}', \mathcal{Q}, \mathcal{F}_1\}$, we have that with overwhelming probability these messages are answered correctly. Hence, there must be a non-negligible set of runs in which network messages from $\mathcal{F}_1$ are not answered correctly. Now consider $\mathcal{E}'$ from above. Then there also is a non-negligible set of runs of $\{\mathcal{E}', \mathcal{S}, \mathcal{F}_1\}$ in which restricting messages on the network from $\mathcal{F}_1$ are answered incorrectly because, by construction of $\mathcal{E}'$, the behavior of the system $\{\mathcal{E}', \mathcal{S}, \mathcal{F}_1\}$ coincides with $\{\mathcal{E}, \mathcal{S}', \mathcal{Q}, \mathcal{F}_1\}$. We already know that $\mathcal{E}' \in \mathsf{Env}_R(\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\})$ from above. Also, by assumption, we have that $\{\mathcal{E}'', \mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\} \equiv \{\mathcal{E}'', \mathcal{S}, \mathcal{F}_1\}$ for all $\mathcal{E}'' \in \mathsf{Env}_R(\{\mathcal{A}_D^{\mathcal{P}_1}, \mathcal{P}_1\})$. Now, by Lemma 4.4, it follows that $\mathsf{Env}_R(\{\mathcal{A}_D, \mathcal{P}_1\}) = \mathsf{Env}_R(\{\mathcal{S}, \mathcal{F}_1\})$, and hence $\mathcal{E}' \in \mathsf{Env}_R(\{\mathcal{S}, \mathcal{F}_1\})$. This contradicts the responsiveness property of $\mathcal{S}$. □

The following composition theorem guarantees the secure composition of an unbounded number of instances of the same protocol system. To state this theorem, we consider single-session (responsive) environments, i.e., environments that invoke a single session of a protocol only. In universal composability models, instances of protocol machines have IDs that consist of party IDs and session IDs. Instances with the same session ID form a session. Instances from different sessions may not directly interact with each other. A *single-session environment* may invoke machines with the same session ID only. We denote the set of single-session environments for a system $\mathcal{Q}$ by $\mathsf{Env}_{R,\mathsf{single}}(\mathcal{Q})$.

We say that $\mathcal{P}$ *single-session realizes* $\mathcal{F}$ ($\mathcal{P} \leq_{R,\mathsf{single}} \mathcal{F}$) if there exists a simulator $\mathcal{S} \in \mathrm{Adv}_R(\mathcal{F})$ such that $\{\mathcal{E}, \mathcal{A}_D, \mathcal{P}\} \equiv \{\mathcal{E}, \mathcal{S}, \mathcal{F}\}$ for all $\mathcal{E} \in \mathsf{Env}_{R,\mathsf{single}}(\{\mathcal{A}_D, \mathcal{P}\})$. Now, the composition theorem states that if a single session of a real protocol $\mathcal{P}$ realizes a single session of an ideal protocol $\mathcal{F}$, then multiple sessions of $\mathcal{P}$ realize multiple sessions of $\mathcal{F}$.

Theorem 4.9. *Let R be a restriction, and let $\mathcal{P}$ and $\mathcal{F}$ be protocols. Then, $\mathcal{P} \leq_{R,\mathsf{single}} \mathcal{F}$ implies $\mathcal{P} \leq_R \mathcal{F}$.*

Proof. Let $\mathcal{S}$ be the simulator for $\mathcal{P} \leq_{R,\mathsf{single}} \mathcal{F}$. A new simulator $\mathcal{S}'$ for arbitrary responsive environments can be constructed just as in the original (non-responsive) composition theorem, i.e., $\mathcal{S}'$ internally keeps one copy of $\mathcal{S}$ per session and uses these copies to answer messages from/to the corresponding sessions.

The proof has two main steps: The first step shows indistinguishability of $\{\mathcal{A}_D, \mathcal{P}\}$ and $\{\mathcal{S}', \mathcal{F}\}$ for every responsive environment $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}_D, \mathcal{P}\})$. The second step shows the responsiveness property of the simulator.

The first part uses a hybrid argument in which one builds a series of single-session environments $\mathcal{E}_i, i \geq 1$, which internally simulate $\mathcal{E}$ such that all messages to the first $i-1$ sessions are sent to internally simulated instances of $\{\mathcal{S}, \mathcal{F}\}$, messages to the i-th session are sent to the (external) system $\{\mathcal{A}_D, \mathcal{P}\}$ or $\{\mathcal{S}, \mathcal{F}\}$, respectively, and the remaining messages are sent to internally simulated instances of $\{\mathcal{A}_D, \mathcal{P}\}$. As different sessions of a protocol do not directly interact with each other, it is easy to see that $\{\mathcal{E}_1, \mathcal{A}_D, \mathcal{P}\}$ behaves just as $\{\mathcal{E}, \mathcal{A}_D, \mathcal{P}\}$ (*), and $\{\mathcal{E}_n, \mathcal{S}, \mathcal{F}\}$ behaves just as $\{\mathcal{E}, \mathcal{S}', \mathcal{F}\}$, where $n \in \mathbb{N}$ is an upper bound of the number of sessions created by $\mathcal{E}$ (note that n is a polynomial in the security parameter and the length of the external input given to the environment, if any). Hence, the distinguishing advantage of $\mathcal{E}$ is bounded by the sum of the advantages of $\mathcal{E}_1, \ldots, \mathcal{E}_n$, i.e., it is sufficient to show that the advantages of $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are bounded by the *same* negligible function[8] to show that $\mathcal{E}$ cannot distinguish $\{\mathcal{A}_D, \mathcal{P}\}$ from $\{\mathcal{S}', \mathcal{F}\}$. In what follows, to show the existence of a single negligible function, we consider environments with external input because the argument is simpler in that case. Nevertheless, using sampling of runs, the argument also works without external input, i.e., in the uniform case (see the full version of this paper [6] for details).

To show that such a bound exists, it is first necessary to prove that there is a (single) negligible function f that, for every $i \leq n$, bounds the probability of $\mathcal{E}_i$ of violating the responsiveness property in runs of $\{\mathcal{A}_D, \mathcal{P}\}$ or $\{\mathcal{S}, \mathcal{F}\}$, respectively. Let $\hat{C}_i^{\{\mathcal{A}_D,\mathcal{P}\}}$ be the event that in runs of $\{\mathcal{E}_i, \mathcal{A}_D, \mathcal{P}\}$ the environment $\mathcal{E}$, which is internally simulated by $\mathcal{E}_i$, answers a restricting message of the external system $\{\mathcal{A}_D, \mathcal{P}\}$ *or* one of the internally simulated instances of $\{\mathcal{A}_D, \mathcal{P}\}$ and $\{\mathcal{S}, \mathcal{F}\}$ incorrectly; $\hat{C}_i^{\{\mathcal{S},\mathcal{F}\}}$ is defined analogously. Because $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}_D, \mathcal{P}\})$ and because of (*), we have that $\hat{C}_1^{\{\mathcal{A}_D,\mathcal{P}\}}$ is negligible. It also holds true that (**) there exists a single negligible function that bounds $|\Pr\left[\hat{C}_i^{\{\mathcal{A}_D,\mathcal{P}\}}\right] - \Pr\left[\hat{C}_i^{\{\mathcal{S},\mathcal{F}\}}\right]|$ for all $i \geq 1$. This is because one can define a single-

[8] It is not sufficient to show that the advantage of every environment $\mathcal{E}_i$ is bounded by a negligible function f_i, which is actually rather easy to show. The negligible functions f_i might be different and then their sum $f_1 + \cdots + f_n$ might not be negligible.

session responsive environment $\mathcal{E}'$ that gets i as external input and then simulates $\mathcal{E}_i$; $\mathcal{E}'$ aborts and outputs 1 as soon as a restricting message is about to be answered incorrectly, and 0 otherwise. Note that because the restriction R can be decided efficiently, $\mathcal{E}'$ can perform the task described. Also, by construction, $\mathcal{E}'$ is a single-session environment (it invokes a single external session only) and it is responsive (it stops the execution before the responsiveness requirement would be violated). As $\mathcal{E}'$ distinguishes $\{\mathcal{A}_D, \mathcal{P}\}$ and $\{\mathcal{S}, \mathcal{F}\}$ only based on the events $\hat{C}_i^{\{\mathcal{A}_D,\mathcal{P}\}}$ and $\hat{C}_i^{\{\mathcal{S},\mathcal{F}\}}$, and both systems are indistinguishable for every single session responsive environment, statement (**) holds true. Finally, observe that, for all $i \geq 2$, the systems $\{\mathcal{E}_{i-1}, \mathcal{S}, \mathcal{F}\}$ and $\{\mathcal{E}_i, \mathcal{A}_D, \mathcal{P}\}$ behave exactly the same, and hence $\Pr\left[\hat{C}_i^{\{\mathcal{A}_D,\mathcal{P}\}}\right] = \Pr\left[\hat{C}_{i-1}^{\{\mathcal{S},\mathcal{F}\}}\right]$. This implies that there is a single negligible function that bounds $\Pr\left[\hat{C}_i^{\{\mathcal{A}_D,\mathcal{P}\}}\right]$ for all $1 \leq i \leq n$ (here we need that n is polynomially bounded).[9] In particular, we have that the probability that $\mathcal{E}_i$ is not responsive for the system $\{\mathcal{A}_D, \mathcal{P}\}$ is bounded by a single negligible function independently of $i \leq n$.

We can now conclude the indistinguishability argument by showing that the advantages of $\mathcal{E}_i$, $1 \leq i \leq n$, in distinguishing $\{\mathcal{A}_D, \mathcal{P}\}$ from $\{\mathcal{S}, \mathcal{F}\}$ are bounded by the same negligible function. For this, we construct another single-session responsive environment $\mathcal{E}''$ analogously to $\mathcal{E}'$. The system $\mathcal{E}''$ expects $1 \leq i \leq n$ as external input (and otherwise stops) and then exactly simulates $\mathcal{E}_i$. Importantly, $\mathcal{E}''$ is responsive for $\{\mathcal{A}_D, \mathcal{P}\}$ because we have shown that every $\mathcal{E}_i$ violates responsiveness with at most the same negligible probability, i.e., the same bound also holds for $\mathcal{E}''$ for every input. As $\mathcal{E}''$ is a single-session responsive environment, its distinguishing advantage for the systems $\{\mathcal{A}_D, \mathcal{P}\}$ or $\{\mathcal{S}, \mathcal{F}\}$ is negligible for every possible input. Moreover, with external input i, its distinguishing advantage is the same as that for $\mathcal{E}_i$. Hence, the same negligible function that bounds the advantage of $\mathcal{E}''$ also bounds all advantages of $\mathcal{E}_i, i \leq n$. As mentioned at the beginning of the proof, this implies indistinguishability of $\{\mathcal{A}_D, \mathcal{P}\}$ and $\{\mathcal{S}', \mathcal{F}\}$ for every responsive environment $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{A}_D, \mathcal{P}\})$.

Having proved indistinguishability, it remains to show that $\mathcal{S}'$ is responsive, i.e., $\mathcal{S}' \in \mathrm{Adv}_R(\mathcal{F})$. Let $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{S}', \mathcal{F}\})$. We have to show that the probability that all restricted messages from $\mathcal{F}$ in runs of $\{\mathcal{E}, \mathcal{S}', \mathcal{F}\}$ are answered correctly (in the sense of Definition 4.2) is overwhelming. For this, consider the following single-session environment $\mathcal{E}'$ that is meant to run with $\{\mathcal{S}, \mathcal{F}\}$: The system $\mathcal{E}'$ first flips $r \leq n$, with n as above, and then internally simulates $\mathcal{E}$ and several sessions of $\{\mathcal{S}, \mathcal{F}\}$ such that messages from $\mathcal{E}$ to the r-th session are sent to the external session, whereas all other messages are processed by the internally simulated sessions. Note that $\{\mathcal{E}', \mathcal{S}, \mathcal{F}\}$ behaves just as $\{\mathcal{E}, \mathcal{S}', \mathcal{F}\}$, and hence, because $\mathcal{E} \in \mathsf{Env}_R(\{\mathcal{S}', \mathcal{F}\})$, by Lemma 4.4 we have that $\mathcal{E}'$ is responsive for $\{\mathcal{S}, \mathcal{F}\}$. Because $\mathcal{S}$ is a responsive adversary, this implies that there is only a negligible set of runs of $\{\mathcal{E}', \mathcal{S}, \mathcal{F}\}$ in which a restricting message of $\mathcal{F}$ is answered incorrectly (by $\mathcal{E}'$ or $\mathcal{S}$). Hence, the probability for this to happen is bounded by some negligible function f. From this and the fact that there are only polynomially many sessions, it follows that the probability that a restricting message

[9] Note that it also follows that $\Pr\left[\hat{C}_i^{\{\mathcal{S},\mathcal{F}\}}\right]$ is bounded for all $1 \leq i \leq n$. However, we do not need this result in the following.

from some session of $\mathcal{F}$ is answered incorrectly is negligible. Hence, $\mathcal{S}'$ is a responsive adversary. □

We note that Theorems 4.8 and 4.9 can be combined to obtain increasingly complex protocols. For example, one can first show that a single session of a real protocol $\mathcal{P}$ realizes a single session of an ideal protocol $\mathcal{F}$. Using the two theorems, it then follows, for example, that a protocol $\mathcal{Q}$ using multiple sessions of $\mathcal{P}$ realizes $\mathcal{Q}$ using multiple sessions of $\mathcal{F}$.

To conclude this section, we note that all of our lemmas and theorems have been proven using a single restriction R. Hence, formally, a protocol designer would have to use the same restriction in all of her security proofs in order to be able to use our results. However, as we show in the full version of this paper [6], this is actually not the case because it is very easy to extend and combine different restrictions while still retaining all security results. Also, as discussed in §6, there is in fact one generic restriction that would suffice for all purposes.

5 Responsive Environments in Concrete Models

In the preceding section, we have presented our universal composability framework with responsive environments in a rather model-independent way. In this section, we outline how to implement this framework in the prominent UC, GNUC, and IITM models to exemplify that our framework and concepts are sufficiently general to be applicable to any universal composability model. While these three models follow the same general idea, they differ in several details which affect the concrete implementation of our concepts in these models (see, e.g., [19, 24] for a discussion of these differences). The main differences and details to be considered concern runtime definitions and the mechanism for addressing (instances of) machines.

To instantiate our universal composability framework with responsive environments for the models mentioned, we mainly have to concretize the definitions in §4.2 for these models, that is, the definitions of restrictions as well as of the responsive environments and adversaries. For some models we also have to adjust their runtime notions slightly. Before presenting the details for the specific models, let us briefly explain the central points to be taken care of:

Runtime. In the GNUC and IITM models, the runtime of systems/protocols is required to be polynomially bounded only for a certain class of environments. As we now want to consider responsive environments, we should restrict the class of environments considered in the GNUC and IITM models to those that are responsive. This also has some technical advantages. To see this, let $\mathcal{R}$ and $\mathcal{R}'$ be two systems/protocols. For example, $\mathcal{R}$ and $\mathcal{R}'$ could be the systems $\{\mathcal{E}, \mathcal{A}_D, \mathcal{Q}, \mathcal{P}\}$ and $\{\mathcal{E}, \mathcal{S}, \mathcal{Q}, \mathcal{I}\}$ as considered in the composition theorem (Theorem 4.8) when we want to prove that $\{\mathcal{Q}, \mathcal{P}\}$ realizes $\{\mathcal{Q}, \mathcal{I}\}$. We often face the situation that we know that, say, $\mathcal{R}$ satisfies the model's runtime bound for all environments in a certain class and that $\mathcal{R}$ and $\mathcal{R}'$ are indistinguishable for every *responsive* environment $\mathcal{E}$ (in this class). This implies that $\mathcal{R}'$ also has to satisfy the runtime notion, but only for all responsive environments of the class. Hence, one cannot necessarily use $\mathcal{R}'$, with any environment, in another system as it does not satisfy the model's runtime notion

(for non-responsive environments $\mathcal{E}$, the runtime of $\mathcal{R}'$ might not be polynomial). Hence, also from a technical point of view, it makes sense to relax the runtime notions in these models in that the runtime of systems/protocols should only be required to be polynomially bounded for responsive environments.

Definition of restrictions. According to Definition 4.1, we require that restrictions are "efficiently decidable". As mentioned, the exact definition depends on the model at hand. The important property this definition should satisfy is the following: An environment $\mathcal{E}'$ which internally simulates another environment $\mathcal{E}$ should be able to decide whether the output $\mathcal{E}$ produces is a correct answer (according to the restriction) when receiving some message as input. That is, $\mathcal{E}'$ must be able to check whether the input message was restricting at all, and if it was, $\mathcal{E}'$ must be able to check whether the response was valid. We often use such simulations in proofs. Depending on the model under consideration, we might not yet (at this point of the proof) have guarantees about the length of the restricting message sent to $\mathcal{E}$. A model-dependent definition of an efficiently decidable restriction should take this into account.

Definition of responsive environments. In the definition of responsive environments (Definition 4.2), we require that an answer to a restricting message be sent back to the same machine and we already explained that "same machine" typically means the same instance from which the restricting message has been received. This has to be specified for the different models.

Definition of responsive adversaries. Depending on the restriction R considered, in some models, in particular UC and GNUC, Definition 4.3 can be too restrictive, and, for example, the dummy adversary in these models might not satisfy the definition. The dummy adversary in these models is required to perform multiplexing. When it receives a message from an instance of the protocol and forwards this message to the environment, it has to prefix the message with the ID of that instance to tell the environment where the message came from. This alters the message, and the resulting message might no longer be restricting, depending on the definition of the restriction R. Hence, the environment would no longer be obliged to answer directly, and thus the (dummy) adversary would not be responsive. One way to fix this is to require a certain closure property of restrictions, namely that adding IDs at the beginning of restricting messages still yields restricting messages and that these message permit the same answers. But this is quite cumbersome. For example, by recursively applying this constraint one would have to require that R be closed under arbitrarily long prefixes of sequences of IDs. A more elegant solution that would still allow simple and natural restrictions would redefine what it means for a message from an adversary to the environment to be restricting. This is what we suggest for the UC and GNUC models (see below).

In what follows, we sketch how to adjust and concretize the runtime notions and the definitions for the UC, GNUC, and IITM models. As mentioned in the introduction, we have carried out the implementation of responsive environments in this model in full detail for the IITM model.

5.1 UC

For the UC model, we do not have to change the runtime definition because the runtime of a protocol is not defined w.r.t. a class of environments, but simply bounded by a fixed polynomial (see also below).

Definition of restrictions. For UC we require both R and $R[0]$ to be decidable in polynomial time in the length of the input. Because of UC's strict runtime definition, this is sufficient to satisfy the requirement mentioned above, namely, that an environment $\mathcal{E}'$ simulating another environment $\mathcal{E}$ can check whether a restricting message received by $\mathcal{E}$ is answered correctly by $\mathcal{E}$. To see this, recall that every machine in UC is required to be parameterized with a polynomial. At every point in the run, the runtime of every instance of a machine is bounded by this polynomial, where the polynomial is in $n := n_I - n_O$, with n_I being the number of bits received so far on the I/O interface from higher-level machines and n_O being the number of bits sent on the I/O interface to lower level machines. Environment machines have to satisfy this condition as well, where n_I is the number of bits of the external input (which contains the security parameter η). Hence, as protocols will receive only a polynomial number of input bits from the environment, they can send messages of polynomial length in the length of the external input plus η only. Therefore, given some message m that was received by an environment and a response m' to this message, the message pair (m, m') has at most polynomial length in the external input plus η, and an environment is able to decide within its runtime bound whether m' is a correct answer to m if we use the above definition of effectively decidable restrictions.

Definition of responsive environments. We require that a response to a restricting message be sent back to the *instance* of the machine that sent the restricting message. This is possible because every instance in UC is assigned a globally unique ID, which is then used to specify the sender and the recipient of a message.

Definition of responsive adversaries. As explained above, messages from the adversary to the environment and vice versa may contain a prefix (typically an ID). For reasons explained above, we say in UC that such a prefix is ignored for the sake of checking whether a message is restricting and whether the answer is correct. To be more specific, a message $m = (pre, \bar{m})$ from the adversary to the environment is restricting iff $\bar{m} \in R[0]$. Also, if m is restricting (in this sense), an answer $m' = (pre', \bar{m}')$ from the environment is allowed if $(\bar{m}, \bar{m}') \in R$ and $pre' = pre$. Using this definition, it is easy to see that the dummy adversary in UC, which adds some prefix to messages from a protocol to the environment and strips off a prefix from messages from the environment to a protocol, is responsive.

5.2 GNUC

The changes necessary for the GNUC model are similar to those for the UC model. However, the runtime notion has to be modified:.

Runtime. Let us first recall the relevant parts of the runtime definition of GNUC.[10] In this model, the runtime definition depends on the entity considered. For an environment $\mathcal{E}$, there has to exist a polynomial p that bounds the runtime of the environment in runs with *every* system where p gets as input the number of bits of all messages that have been received by the environment during the run, including the external input, plus the security parameter η. For a protocol $\mathcal{P}$, there has to exist a polynomial q such that the runtime of $\mathcal{P}$ is bounded by q in runs with any environment and the dummy adversary where q gets as input the number of bits that are output by the environment (to both the adversary and the protocol). This definition has to be changed such that the runtime of a protocol needs to be bounded only for all environments (in the sense of GNUC) that in addition are responsive.

Definition of restrictions. Analogously to UC, we require R and $R[0]$ to be decidable in polynomial time in the length of the input. This is sufficient to satisfy the described requirement ($\mathcal{E}'$ simulating $\mathcal{E}$) as the runtime of environments in GNUC depends on the number of bits received from a protocol. Hence, an environment is always able to read a potentially restricting message m entirely, whereas the length of an answer m' is bounded by the runtime bound of the environment.

Definition of responsive environments. Just as for UC, we require that responses to restricting messages be sent to the same *instance* of a machine. This is possible in GNUC because, again, all machines have globally unique IDs to address instances.

Definition of responsive adversaries. Just as for UC, the adversary in GNUC might (have to) add IDs as prefixes or remove such prefixes, therefore these prefixes are ignored in the definition of responsive adversaries.

5.3 IITM

Just as for the other models, we now outline how to adjust and concretize the runtime notion and the definitions from §4 for the IITM model. As mentioned, in the full version of this paper [6], we provide full details for the IITM model with responsive environments, with a brief summary of the results presented at the end of this subsection.

Runtime. In the IITM model, the runtime depends on the type of entity. For an environment $\mathcal{E}$, it is required that there exists a polynomial p (in the length of the external input, if any, plus the security parameter) such that for *every* system running with $\mathcal{E}$ the runtime of $\mathcal{E}$ with this system is bounded by p. For a protocol $\mathcal{P}$, it is merely required that it be environmentally bounded, i.e., for every environment $\mathcal{E}$ there is a polynomial q (again, in the length of the external input plus the security parameter) that bounds the overall runtime of runs of $\{\mathcal{E}, \mathcal{P}\}$ (except for at most a negligible set of runs).[11] Given a protocol $\mathcal{P}$, for an adversary $\mathcal{A}$ for $\mathcal{P}$ it is required only that $\{\mathcal{A}, \mathcal{P}\}$ be environmentally bounded. (Clearly, the dummy adversary is environmentally bounded.) To adjust the runtime notions for the setting with responsive environments, instead of quantifying over

[10] Note that there are several additional requirements, such as bounds on the number of bits that are sent by the environment as well as so-called invited messages. These details, however, are not relevant here.

[11] Here $\mathcal{E}$ may directly connect to $\mathcal{P}$'s network interface. Equivalently one could have $\mathcal{E}$ communicate with $\mathcal{P}$ on the network interface via a dummy adversary.

all environments in the definition of environmentally bounded protocols/adversaries, one should now quantify over responsive environments only, as motivated at the beginning of §5.

Definition of restrictions. We require that a restriction R is *efficiently decidable in the second component*, i.e., there is an algorithm A which expects pairs (m, m') of messages as input and which runs in polynomial time in $|m'|$ in order to decide whether m' is a correct answer to m according to R (see the full version of this paper [6] for a formal definition). This stronger definition is necessary to obtain the property described, namely, that an environment $\mathcal{E}'$ internally simulating another environment $\mathcal{E}$ can check that answers of $\mathcal{E}$ to restricting messages are correct. Owing to the very liberal runtime notion for protocols used in the IITM model, in proofs (e.g., of the composition theorem) we sometimes have to establish that a system is environmentally bounded. Therefore, we do not know a priori that the length of the message m is polynomially bounded. Hence, the environment might not be able to read m completely. Conversely, the length of m' is guaranteed to be polynomially bounded as it is output by the environment $\mathcal{E}$, which, by definition, is polynomially bounded. With R being efficiently decidable in the second component, $\mathcal{E}'$ can then efficiently decide whether m' is a correct answer to m. Compared with the definition of restrictions for the UC and GNUC models presented above, this formally is more restricted. It is, however, sufficient for all practical purposed, as discussed in §6, as one has to consider one generic restriction only and this restriction is efficiently decidable in the second component.

Definition of responsive environments. Unlike the UC and GNUC models, the IITM model does not hardwire a specific addressing mechanism for instances of machines and specific IDs for such instances into the model. Instead, it supports a flexible addressing mechanism which allows a protocol designer to specify how machine instances are addressed and what they consider to be their ID. More specifically, the IITM model allows a protocol designer to specify an algorithm run by machine instances that decides whether the message received is accepted by the instance or not. Therefore, in the IITM model, we can require only that responses to restricting messages be sent to the same machine, but not necessarily the same machine *instance*. This, however, is indeed sufficient. A protocol designer, can specify that a (protocol) machine accepts a message iff it is prefixed by a certain ID (the one seen in the first activation of the instance) as typically done in the IITM model. This ID can then be considered to be the ID of this machine instance, and messages output by this machine would also be prefixed by this ID. Now, a protocol designer can use restrictions to manually enforce that the same instance receives a response. Such a restriction would contain message pairs of the form $((id, m), (id, m'))$. By this, it is guaranteed that if a restricting message has been sent by a protocol machine instance with ID id, then the response is returned to this instance, as the response is prefixed with id.

Definition of responsive adversaries. For the IITM model, we do not have to change the definition of responsive adversaries. Adversaries in the IITM model do not have to add prefixes to messages, and hence, do not have to modify restricting messages. In particular, the dummy adversary simply forwards messages between the environment and the protocol without changing messages.

Detailed Results for the IITM Model. In the full version of this paper [6] we provide full details of the IITM model with responsive environments. That is, we adjust the runtime notion of the IITM model accordingly, and provide full definitions of restrictions, responsive environments and adversaries. Based on these definitions we define the various security notions for realization relations considered in the literature (now with responsive environments), namely, (dummy) UC, black-box simulatability, strong simulatability, and reactive simulatability. These new and adjusted notions have been carefully developed in order to be general and to preserve central properties. In particular, we show that all the notions mentioned for realization relations are equivalent (for reactive simulatability, this requires environments with external input). We also prove that these relations are reflexive and transitive. We finally prove the composition theorems for responsive environments. As should be clear from the proof sketches in §4, the proofs are more involved than those without responsive environments because one always has to ensure that the constructed environments and simulators are responsive. The full proofs are even more intricate and non-trivial because they take all model-specific details, such as the liberal runtime notions, into account. We note, however, that this is a once and for all effort. Protocol designers no longer have to perform such proofs. They can simply use the results. That is, responsive environments do not put any burden on the protocol designer. On the contrary, as explained, they greatly simplify the specification and analysis of protocols.

6 Applying Our Concepts to the Literature

Our new concepts of restricting messages and responsive environments and adversaries allow protocol designers to avoid the non-responsiveness problem elegantly and completely. As mentioned, urgent requests can simply be declared to be restricting messages, causing the adversary/environment to reply with a valid response before sending any other message to the protocol. This indeed seems to be the most reasonable and natural solution to the non-responsiveness problem. We now show that our approach indeed easily solves all the problems mentioned in §1 and §3.

The frequently encountered formulations of the form (1) mentioned in §3.1 can now actually be used without causing confusion and flawed specifications, if the message sent to the adversary is declared to be restricting: there will now in fact be an immediate answer to this message. Similarly, ideal functionalities which are intended to be non-interactive can now be made non-interactive (at least if uncorrupted; but, if desired and realistic, also in the corrupted case) just like their realizations, which solves the problems discussed in §3.2.2 (lack of expressivity), and also makes it possible to use the, again, often encountered specifications of the form (2): if such ideal functionalities have to send urgent requests to the adversary, such requests can be made restricting, and hence, prompt replies are guaranteed, i.e., if the (responsive) adversary/environment contacts the protocol at all again, it first has to answer the request. Clearly, the other problems caused by urgent requests not being answered immediately discussed in §3.2, namely, unintended state changes and race conditions, the reentrance problem, and unnatural specifications of higher-level protocols, vanish also; again, because urgent request now *are answered immediately*.

Two ways of defining restrictions. We note that there are two approaches to define restrictions R.

Tailored restrictions. One approach is to define restrictions tailored to specific protocols and functionalities. For example, for $\mathcal{F}_{\text{D-Cert}}$ the restriction could be defined as follows:

$$\left\{\left((\texttt{Verify}, sid, m, \sigma), (\texttt{Verified}, sid, m, \phi)\right) : sid, m, \sigma \in \{0,1\}^*, \phi \in \{0,1\}\right\}$$

Now, whenever the adversary is asked to verify some σ, the next message sent to the ideal functionality is guaranteed to be the expected response. This directly resolves the issues discussed in §3.2.1. Similarly, one could, for example, define restrictions for $\mathcal{F}_{\text{NIKE}}$ and $\mathcal{F}_{\text{sok}}$.[12]

We note that the above approach of defining a separate restriction for each protocol is general in the sense that it can be used independently of the underlying model for universal composition, and is thus applicable, e.g., to the UC, GNUC, and IITM models. Furthermore, this solution allows one to fix many ideal functionalities and their realizations found in the literature without any modifications to the specifications, including all examples mentioned in this document. However, since the composition theorems and the transitivity property assume one restriction, different restrictions have to be combined into a single one. This is always possible as shown in the full version of this paper [6]. Nevertheless, the following solution seems preferable.

Generic Restriction. Alternatively to employing tailored restrictions, one can use the following generic restriction:

$$R_G := \{(m, m') \mid m = (\texttt{Respond}, m''), m', m'' \in \{0,1\}^*\}.$$

This means that messages prefixed with `Respond` are considered to be restricting, and hence protocol designers can declare a message to be restricting by simply prefixing it by `Respond`. According to the definition of R_G, the adversary/environment can respond with any message to these messages, but protocols or ideal functionalities can be defined in such a way that they repeat their requests until they receive the expected answer: for instance, in the case of $\mathcal{F}_{\text{sok}}$, it can repeatedly send $m'' = (\texttt{Setup}, sid)$ to the adversary until it receives the expected algorithms. In this way, the adversary is forced to eventually provide an expected answer (if she wants the protocol to proceed).

Using this fixed multi-purpose restriction has the advantage that, in contrast to the former approach, there is no need to combine different restrictions. Also, in protocol specifications, the prefixing immediately makes clear which messages are considered to be restricting.

The main reasons we did not hardwire the generic restriction into our framework are twofold. First, this is not required to prove our results, but makes our framework only more general, and the flexibility might become useful in some situations. Second, as protocols and ideal functionalities have to send several requests until they get the

[12] Note that to show that the respective real protocols realize their ideal functionalities, according to Definition 4.6, one needs to prove that there exists a *responsive* simulator. However, it is easy to verify that the simulators constructed in [14, 17, 31] for the functionalities mentioned already are responsive, and thus these realizations can be used unalteredly also in a responsive setting.

expected answer, depending on the runtime notions used, they might run out of resources. In the IITM model, however, this is not an issue, and hence the generic restriction can be used.

7 Conclusion

In this paper, we highlighted the non-responsiveness problem, the fact that it has often been ignored in the literature, and its many negative consequences.

We have proposed a framework that completely avoids this problem. It enables protocol designers to declare urgent requests to be restricting messages, causing such requests to be answered immediately by (responsive) environments/adversaries. This, in particular, allows protocols and ideal functionalities to be defined in the expected and natural way. It also avoids unnecessarily complex and artificial specifications, unintended state changes and race conditions while waiting for responses to urgent requests, the reentrance problem, the lack of expressivity when modeling non-interactive tasks, and the propagation of such problems to higher-level protocols and proofs. We discussed how our concepts can be adopted by existing models for universal composition, as exemplified in this work for the UC, GNUC, and IITM models. In the full version of this paper [6], we also provide full details for the IITM model, showing that our concepts can seamlessly be integrated into the existing model without losing any of the properties of the setting without responsive environments: all security notions for the realization relations are formulated, shown to (still) be equivalent, and enjoy reflexivity and transitivity; the composition theorems also carry over to the setting with responsive environments.

References

1. Abe, M., Ohkubo, M.: A framework for universally composable non-committing blind signatures. In: ASIACRYPT 2009. LNCS, vol. 5912, pp. 435–450. Springer (2009)
2. Backes, M., Dürmuth, M., Hofheinz, D., Küsters, R.: Conditional Reactive Simulatability. International Journal of Information Security (IJIS) 7(2), 155–169 (April 2008)
3. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Information and Computation 205(12), 1685–1720 (2007)
4. Backes, M., Hofheinz, D.: How to break and repair a universally composable signature functionality. In: ISC 2004. LNCS, vol. 3225, pp. 61–72. Springer (2004)
5. Camenisch, J., Dubovitskaya, M., Haralambiev, K., Kohlweiss, M.: Composable & modular anonymous credentials: Definitions and practical constructions. ASIACRYPT 2015 (2015)
6. Camenisch, J., Enderlein, R.R., Krenn, S., Küsters, R., Rausch, D.: Universal Composition with Responsive Environments. Tech. rep., Cryptology ePrint Archive, Report 2016/034 (2016), available at http://eprint.iacr.org/2016/034.
7. Canetti, R.: Universally Composable Signature, Certification, and Authentication. In: CSFW 2004. pp. 219–233. IEEE (2004)
8. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001), see also https://eprint.iacr.org/2000/067.pdf for full and previous versions
9. Canetti, R., Cheung, L., Kaynar, D.K., Liskov, M., Lynch, N.A., Pereira, O., Segala, R.: Time-Bounded Task-PIOAs: A Framework for Analyzing Security Protocols. In: DISC 2006. LNCS, vol. 4167, pp. 238–253. Springer (2006)
10. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: TCC 2007. LNCS, vol. 4392, pp. 61–85. Springer (2007)

11. Canetti, R., Halevi, S., Katz, J.: Adaptively-secure, non-interactive public-key encryption. In: TCC 2005. LNCS, vol. 3378, pp. 150–168. Springer (2005)
12. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer (2003)
13. Canetti, R., Shahaf, D., Vald, M.: Universally composable authentication and key-exchange with global PKI. Cryptology ePrint Archive, Report 2014/432 (2014)
14. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: CRYPTO 2006. LNCS, vol. 4117, pp. 78–96. Springer (2006)
15. Damgård, I., Hofheinz, D., Kiltz, E., Thorbek, R.: Public-key encryption with non-interactive opening. In: CT-RSA 2008. LNCS, vol. 4964, pp. 239–255. Springer (2008)
16. Dowsley, R., Müller-Quade, J., Otsuka, A., Hanaoka, G., Imai, H., Nascimento, A.C.A.: Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. IEICE Transactions 94-A(2), 725–734 (2011)
17. Freire, E.S.V., Hesse, J., Hofheinz, D.: Universally composable non-interactive key exchange. In: SCN 14. pp. 1–20. LNCS, Springer (2014)
18. Hazay, C., Venkitasubramaniam, M.: On Black-Box Complexity of Universally Composable Security in the CRS Model. In: ASIACRYPT 2015. LNCS, vol. 9453, pp. 183–209. Springer (2015)
19. Hofheinz, D., Shoup, V.: GNUC: A new universal composability framework. Cryptology ePrint Archive, Report 2011/303 (2011)
20. Hofheinz, D., Unruh, D., Müller-Quade, J.: Polynomial runtime and composability. Journal of Cryptology 26(3), 375–441 (2013)
21. Kurosawa, K., Furukawa, J.: Universally composable undeniable signature. In: ICALP 2008, Part II. LNCS, vol. 5126, pp. 524–535. Springer (2008)
22. Küsters, R.: Simulation-Based Security with Inexhaustible Interactive Turing Machines. In: CSFW '06. pp. 309–320. IEEE (2006)
23. Küsters, R., Tuengerthal, M.: Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008). pp. 270–284. IEEE Computer Society (2008)
24. Küsters, R., Tuengerthal, M.: The IITM model: a simple and expressive model for universal composability. Cryptology ePrint Archive, Report 2013/025 (2013)
25. Laud, P., Ngo, L.: Threshold Homomorphic Encryption in the Universally Composable Cryptographic Library. Cryptology ePrint Archive, Report 2008/367 (2008)
26. Matsuo, T., Matsuo, S.: On universal composable security of time-stamping protocols. In: IWAP 2005. pp. 169–181 (2005)
27. Maurer, U.: Constructive Cryptography - A New Paradigm for Security Definitions and Proofs. In: TOSCA 2011. LNCS, vol. 6993, pp. 33–56. Springer (2011)
28. Maurer, U., Renner, R.: Abstract cryptography. In: ICS 2011. pp. 1–21. Tsinghua University Press (2011)
29. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: ACM CCS 00. pp. 245–254. ACM Press (2000)
30. Tian, Y., Peng, C.: Universally composable secure group communication. Cryptology ePrint Archive, Report 2014/647 (2014), http://eprint.iacr.org/
31. Zhao, S., Zhang, Q., Qin, Y., Feng, D.: Universally composable secure TNC protocol based on IF-T binding to TLS. In: NSS 2014. LNCS, vol. 8792, pp. 110–123. Springer (2014)