Polynomial Runtime and Composability

Dennis Hofheinz
Karlsruhe Institute of Technology

Dominique Unruh
University of Tartu

Jörn Müller-Quade
Karlsruhe Institute of Technology

May 31, 2012

Abstract

We devise a notion of polynomial runtime suitable for the simulation-based security analysis of multi-party cryptographic protocols. Somewhat surprisingly, straightforward notions of polynomial runtime lack expressivity for reactive tasks and/or lead to an unnatural simulation-based security notion. Indeed, the problem has been recognized in previous works, and several notions of polynomial runtime have already been proposed. However, our new notion, dubbed *reactive polynomial time* is the first to combine the following properties:

- it is *simple* enough to support simple security/runtime analyses,
- it is *intuitive* in the sense that all intuitively feasible protocols and attacks (and only those) are considered polynomial-time,
- it supports *secure composition* of protocols in the sense of a universal composition theorem.

We work in the Universal Composability (UC) protocol framework. We remark that while the UC framework already features a universal composition theorem, we develop new techniques to prove secure composition in case of reactively polynomial time protocols and attacks.

Keywords: Universal composability, polynomial runtime, multi-party protocols, protocol composition.

Contents

1 Introduction

1.1 Introduction to the problem

The security of cryptographic protocols is often based on the hardness of certain computational problems, such as, e.g., inverting a given trapdoor one-way permutation. Breaking the protocol security then requires solving the underlying computational problem. To prove this, one generally considers reductions, i.e., one translates a successful cryptographic attack on the protocol security into an algorithm that solves the underlying computational problem. For such a reduction to work, it is necessary that the complexity of protocol runs is bounded, so that the protocol situation can be translated into the setting in which the computational assumption is formulated. Typically, computational assumptions are formulated against algorithms which are probabilistic polynomial time. That means, one usually assumes that an arbitrary but fixed polynomial upper-bounds the runtime of the algorithm.

So it is not merely of aesthetic interest to find a notion that captures the notion of polynomial time complexity for cryptographic protocols. It is also a practical necessity to conduct security proofs.

Our goal in this contribution is to find a useful and meaningful notion of polynomial time complexity for cryptographic protocols that matches the intuition of what is feasible. In particular, the induced security notion should be useful when analyzing the composition of protocols.

More specifically, we endeavor to find a notion of polynomial-time together with a variant of the UC security notion such that the following requirements are fulfilled:

Flexibility: All "intuitively feasible" protocols and protocol tasks (and only those) should be considered polynomial-time. In particular, natural protocol tasks like secure channels should be polynomial-time and not be excluded for formal reasons.

Soundness: All "intuitively feasible" attacks (i.e., adversaries) should be considered polynomial-time. Otherwise, we would have no guarantee that a secure protocol indeed withstands real-world attacks. In particular, in the context of universal composability the very important "dummy adversary" should be polynomial-time.

Completeness: *Only* "intuitively feasible" attacks should be considered polynomial-time. Otherwise, the resulting security notion would be too strong and the security of protocols could not be reduced to computational hardness assumptions.

Composability: The security notion should support secure composition of an arbitrary number of concurrent protocol instances in arbitrary contexts (universal composition).

Simplicity: the notion should be easy to formulate, and for all practical cases, it should be easy to decide whether a protocol or attack runs in polynomial time.

Note that flexibility can be seen as a dual to soundness and completeness. In particular, flexibility captures the feasibility of protocols, while soundness resp. completeness capture the feasibility of attacks. Thus, this distinction distinguishes requirements on *algorithms* from requirements on *adversaries*.

The UC framework. Since we strive for composability, we work in the protocol framework of universal composability (UC) [Can01, Can05a].[1] The UC framework [Can01, Can05a] defines the security of a protocol (often called the real protocol) by comparison with an ideal protocol. The ideal protocol usually comprises only a single trusted machine, a so-called ideal functionality, which is secure by construction. The ideal protocol can be thought of as the specification of the protocol task that should be achieved by the real protocol. In the UC framework, the real protocol is considered to be a secure implementation of the ideal protocol if only those attacks are possible in the real protocol that are also possible in the ideal protocol. More precisely, for any adversary that interacts with (attacks) the real protocol, there is a corresponding adversary (the simulator) that interacts with the ideal protocol such that no protocol environment interacting with both the protocol and the adversary can distinguish between an execution of the real and an execution of the ideal protocol. In this case we say that the real protocol emulates the ideal protocol. To be able to use computational assumptions in the protocol design, one usually requires the adversary, the simulator, the environment, and both the ideal and the real protocol to be polynomial-time.

[1] We stress that our observations and results apply as well in any other protocol framework in which security is defined through an *interactive* simulation. In particular, our results apply also in the frameworks of Reactive Simulatability (RSIM) [PW01, BPW04b], SPPC [DKMR05], IITM [Küs06], Task-PIOA [CCK+06a, CCK+06b], and environmental security [Gol04, Section 7.7.2].

Since ideal functionalities can model very different protocol tasks, the UC framework is very versatile. Furthermore, it gives very strong composability guarantees: If a protocol π emulates a protocol ϕ, and a protocol ρ that uses the ideal protocol ϕ as a subprotocol emulates some ideal functionality $\mathcal{F}$, then after replacing ϕ by its implementation π, ρ still emulates $\mathcal{F}$. This enables the modular design of security protocols.

We give a detailed overview over the UC framework in Section 2.

1.2 Our work

Our approach: reactively polynomial-time protocols. We propose a new notion of polynomial runtime for cryptographic protocols; *reactive polynomial time*. The basic intuition behind our notion is that a protocol should be considered polynomial-time as long as it is not possible to make it run more than a polynomial number of steps. However, the precise polynomial bounding the number of steps should depend on the context the protocol runs with. For example, if the protocol runs in a context that gives extremely long input, the protocol should be allowed to run longer. On the other hand, we should not allow contexts that input messages of superpolynomial length; it would be too restrictive to require that the protocol runs in polynomial-time on superpolynomial inputs. Thus we restrict the contexts to *a priori* polynomial-time machines, i.e., machines whose running time, in any situation, is bounded by a fixed polynomial (depending only on the machine). Note that *a priori* polynomial time is the classical notion of polynomial time for reactive machines, employed, e.g., in [Can01, PW01]. We then get the following definition of reactive polynomial time:

Definition 1 (Reactive polynomial time (informal)) *A protocol π runs in* reactive polynomial time *iff for any* a priori *polynomial-time machine $\mathcal{Z}$, we have there is a polynomial p, such that the network $\pi \cup \{\mathcal{Z}\}$ (in which $\mathcal{Z}$ interacts with π) runs, with overwhelming probability, at most p steps.*

It is easy to see according to this definition, the polynomial p bounding the running time of π may depend on $\mathcal{Z}$ (and thus be large enough so that π can process all inputs coming from $\mathcal{Z}$). On the other hand, since $\mathcal{Z}$ is *a priori* polynomial-time, π is not required to run in polynomial-time on superpolynomially long inputs or on inputs that are infeasible to find.

Notice that we only require $\pi \cup \{\mathcal{Z}\}$ to have runtime p with overwhelming probability, not with probability 1. This may seem like an unnecessary complication, yet this relaxation is essential to allow for a composition theorem; see Section 9.1 for a detailed explanation and proof. (We remark, however, that it is *not* essential whether the considered protocol *context* is polynomial-time or only polynomial-time with overwhelming probability; see Section 4, page 31 for an explanation.)

It should also be noted that being reactively polynomial-time is a property of the protocol as a whole, not of the individual machines (unlike the property of being *a priori* polynomial-time).

Given the notion of reactive polynomial time, we can now rephrase the notion of UC with respect to reactively polynomial-time protocols.

Definition 2 (UC with respect to reactive polynomial time (informal)) *We call an adversary $\mathcal{A}$ valid for a protocol π iff $\pi \cup \{\mathcal{A}\}$ (the protocol π running together with the adversary $\mathcal{A}$) is reactively polynomial-time. Analogously for simulators.*

Let π and ϕ be protocols. We say π emulates ϕ iff for any adversary $\mathcal{A}$ that is valid for π there is a simulator $\mathcal{S}$ that is valid for ϕ such that no a priori *polynomial-time environment $\mathcal{Z}$ can distinguish between $\pi \cup \{\mathcal{A}\}$ and $\phi \cup \{\mathcal{S}\}$.*

Notice that the only difference to the classical notion of UC [Can01] is that we do not quantify over *a priori* polynomial-time adversaries and simulators, but instead over valid adversaries and simulators, i.e., over adversaries and simulators that keep the protocol reactively polynomial-time.

The advantage of this notion is that it behaves better than classical UC when the protocols are reactive polynomial-time (and not just *a priori* polynomial-time). For example we get the following composition theorem:

Theorem 3 (Universal composition theorem (informal)) *Let π, ϕ, and ρ be protocols. Let ρ^π denote the protocol where ρ invokes an arbitrary number of instances of π as subprotocols. Analogously for ρ^ϕ. Assume that π and ρ^π are reactively polynomial-time, and that π emulates ϕ. Then ρ^π emulates ρ^ϕ.*

We stress that with respect to classical notions of UC, this theorem only holds if π and ϕ are *a priori* polynomial time. Yet, considering only *a priori* polynomial-time protocols would exclude many natural protocols (see Section 1.4).

There is one noteworthy limitation to our composition theorem: It takes as an explicit assumption that ρ^π is reactively polynomial-time. Thus, in order to apply the theorem, one needs to manually verify that the composed protocol ρ^π is reactively polynomial-time in order to derive its security with the composition theorem. Fortunately, in most cases the runtime of a protocol is simple to analyze, while its security is the interesting property.

Simplicity. We claim that our definitions are quite simple. Of course, simplicity is both a matter of taste, as well as a matter of comparison with other notions. The most basic notion, *a priori* polynomial-time (which is used in [Can01, PW01]) is arguably simpler than our notion, but it excludes protocols whose running time depends on the input length, and it is subject to certain technical artifacts (see our discussion in Section 1.4). Prior notions that try to solve the problems of *a priori* polynomial-time are, in our opinion, more complicated than reactive polynomial time. Often, they have to introduce distinctions of various types of channels between machines; the maximum allowed running time depends in different ways on the amount of communication on the different channels types. ([Can05a] distinguishes six types of tapes for input/output, subrouting invocation/results, and incoming/outgoing messages; [HMQU05] introduces a special connection between environment and adversary that is counted differently.) Other approaches introduce some methods of filtering unneeded incoming communication so that

it does not waste runtime. ([Bac02, BPW04b] introduce so-called length-functions that allow to close down a connection; [DKMR05] extends this approach to so-called guards that are general purpose filters on incoming connections.) Other approaches use artificial padding of inputs or outputs in order to satisfy certain certain length requirements that come from the definition of polynomial time. (In some notions, a functionality leaking some length l can be securely realized if it sends l in unary encoding while it cannot be realized if it sends l in binary; see Section 8. See also our discussion on padding in Section 3.) Our notion of reactive polynomial time does not need any such technical tools in the network and machine model.

It remains, of course, the question whether it is simple to check whether a given protocol is reactive polynomial time. In general, this is of course an undecidable problem. (Even for *a priori* polynomial time, this problem is undecidable.) We believe, however, that for a natural protocol, it is easy to see that it is reactively polynomial-time: In most cases, there will be a (simple to find) upper bound on the running time that is a polynomial of the total length of the protocol inputs. Deciding whether a given adversary is valid may be harder. In contrast to protocols, we cannot expect that adversaries are in any way natural. Fortunately, we show that without loss of generality, one can assume a particular adversary, the *dummy adversary* (Theorem 19). This adversary just forwards all messages; its running time is trivial to analyze.

Flexibility. We claim that the notion of a reactively polynomial-time protocol captures all intuitively feasible protocol tasks (and only those). We admit that such a claim is hard to formally substantiate, since the set of intuitively feasible protocol tasks is not formally defined. However, it is clear that the notion of reactive polynomial time generalizes *a priori* and *a posteriori* polynomial runtime bounds as discussed above. Furthermore, it is easily verified that the problematic use cases described in Section 1.3 below *can* be modeled as reactively polynomial-time protocols (resp. ideal functionalities). On the other hand, a reactively polynomial protocol along with a valid adversary behaves efficiently in any given *a priori* polynomial-time protocol context (except with small probability). In particular, such a system can be simulated (up to negligible error) inside one single machine that is polynomial-time in the usual, static sense (i.e., *a priori* polynomial). Hence, every reactively polynomial-time protocol is efficient in this sense. Summarizing, we claim that our notion is flexible.

Soundness and completeness. We consider only adversaries that are valid, i.e., adversaries that, together with the protocol, are reactively polynomial-time. This implies that the adversary never (up to negligible probability) runs more than a polynomial number of steps (Lemma 12). Thus only "intuitively feasible" attacks are considered; we achieve completeness. On the other hand, if the protocol is reactively polynomial-time, all reasonable adversaries are considered: We consider all adversaries that do not make the protocol loose its reactive polynomial time property, i.e., all adversaries that do not "introduce" superpolynomial-time computations. Thus all "intuitively feasible" attacks are covered; we achieve soundness. Moreover, we stress that none of the technical artifacts discussed in Section 1.4 occur. In particular, neither adversaries nor protocol machines can be "exhausted". That is, we do not get the artificial condition that a machine is forced

to ignore messages because the environment sends to many inputs. Additionally, in our notion the important "dummy adversary" is valid (Theorem 19), which is important both for composability and soundness. On the other hand, using reactively polynomial-time adversaries induces a security notion that lies (strictly) in between the traditional UC security notion and a relaxation of UC discussed in [Lin03] (Theorem 41). Thus, our new notion still provides a reasonable and useful definition of security.

Composability. Concerning composability, our notion has a certain (unavoidable) limitation: If two networks S_1, S_2 are both reactively polynomial-time, the network $S_1 \cup S_2$ consisting of all machines in S_1 and S_2 is not necessarily reactively polynomial-time. This stands in contrast to other notions of polynomial time such as *a priori* polynomial time. Yet, we claim that this limitation is an unavoidable consequence of the flexibility requirement. There are machines that, on their own, would be intuitively considered to be polynomial-time, yet the composition of several such machines would not be polynomial-time with respect to any reasonable definition. For example, two machines that echo all incoming data should be polynomial-time individually, but if they are combined to echo each other's messages, their running time is unbounded. Since the flexibility requirement demands that such (intuitively polynomial-time) machines are considered polynomial-time, it is not possible to satisfy flexibility and composability simultaneously.

However, the most important aspect of the composability requirement is satisfied by our notion, namely that security (as opposed to running time) is preserved under composition.

We demonstrate that our notion induces a composable security notion by proving the universal composition theorem (Theorem 3/21). This proof is considerably more complex than proofs of composability for previous notions of polynomial runtime (such as, e.g., the proofs from [Can01, BPW04a, Can05a, DKMR05]). Ironically, this complexity seems to result from the simplicity of our notion: in the proof, it is necessary to prove that certain combinations of protocol pieces are still reactively polynomial-time. The good news is that these results do not have to be proven during the design of the protocol (except for the condition that ρ^π is reactively polynomial-time). As a consequence, our composition theorem needs only relatively few assumptions, which might come in very handy during protocol design. We now provide further details.

Common structure of (universal) composition theorems. Put very briefly, a (universal) composition theorem states that whenever *one* protocol instance is secure, then also *multiple* protocol instances are secure, even when used in arbitrary contexts. In the UC framework, security means existence of a simulator. Hence, to prove a UC composition theorem, one usually (explicitly) constructs a simulator $\mathcal{S}^\infty$ for many protocol instances from a simulator $\mathcal{S}$ for one protocol instance. This construction usually (e.g., [Can01, BPW04a]) is conceptually simple: $\mathcal{S}^\infty$ is the combination of multiple instances of $\mathcal{S}$.[2] To prove security, one must show that

1. the constructed simulator $\mathcal{S}^\infty$ is valid (in the sense that $\mathcal{S}^\infty$ fulfils the respective polynomial-time notion), and

[2] Of course, we are oversimplifying here. A more accurate presentation will be given in Section 2.1.

2. $\mathcal{S}^\infty$ achieves a successful simulation (in the sense of the UC security definition).
The first of these properties is usually trivially verified, while the second property is shown using a hybrid argument.

The obstacle with reactively polynomial-time simulators. In the case of reactively polynomial-time protocols and adversaries, however, the first property ($\mathcal{S}^\infty$ is a valid adversary) is *not* trivially verified. Concretely, as hinted above, the composition of several reactively polynomial-time machines may no longer be reactively polynomal. As an example, consider a "double-repeater" R that resends every incoming message *twice* (i.e., on incoming message x, it sends xx). Any single such machine is clearly reactively polynomial-time. However, *pipelining* k such machines R yields a machine R' which, e.g., sends 1^{2^k} when receiving 1. Thus, R' is exponential-time and not reactively polynomial-time. We stress that we consider this property of our notion of reactive polynomial time not to be an artifact, but a necessity. The lack of composability of the notion itself is simply the price we have to pay for completeness, i.e., for the ability to model natural functionalities such as secure channels or (double-)repeaters. If we want to model such machines (and this is the design decision we made), then we have to deal with the technical consequences.

Our techniques to overcome the obstacle. Hence, we have to explicitly *prove* that the combined simulator $\mathcal{S}^\infty$ constructed in the composition is, together with the composed protocol, reactively polynomial-time. To this end, we use not only that one simulator instance $\mathcal{S}$ is reactively polynomial-time. We also employ the fact that $\mathcal{S}$ achieves UC indistinguishability. More concretely, we show that if $\mathcal{S}^\infty$ was not reactively polynomial-time, then we could distinguish a simulation by $\mathcal{S}$ from a real attack on a single protocol instance.

We proceed as follows: in the l-th step, we consider a hybrid system H_l. H_l consists of a fixed larger protocol ρ, together with l instances of the ideal subprotocol ϕ, each running with an instance of simulator $\mathcal{S}$. (The remaining subprotocol instances requested by ρ are instances of π running with the dummy adversary.)

Essentially, we will prove that *all* l ideal subprotocol simulations in H_l adhere to a single polynomial runtime bound T *that does not depend on* l. To show our claim, we proceed by induction on l. By assumption on ρ^π and $\mathcal{S}$, we know that such a bound T exists in H_1. Fix this bound for all l. Now assume we have proved that all ideal simulations in H_{l-1} adhere to bound T. Now consider an environment $\mathcal{Z}_l^*$ that internally simulates H_{l-1}, but relays one real subprotocol instance to the outside (cf. Figure 1 (c) on page 23). If $\mathcal{Z}_l^*$ runs with π and the dummy adversary, this setup equals H_{l-1}. But if $\mathcal{Z}_l^*$ runs with ϕ and $\mathcal{S}$, this setup equals H_l. Hence, by induction hypothesis, $\mathcal{Z}_l^*$ will observe that when running with π and dummy adversary, all internally simulated ideal subprotocol instances adhere to the runtime bound T.

Since π emulates ϕ, this implies that the same holds also when $\mathcal{Z}_l^*$ runs with ϕ and $\mathcal{S}$. Note the unfortunate asymmetry of this argument: we *can* conclude that $l-1$ ideal subprotocol instances in H_l adhere to bound T. (Namely, this holds for all ideal instances that are internally simulated inside $\mathcal{Z}_l^*$.) However, we cannot immediately deduce that the bound T applies to the l-th ideal subprotocol instance that corresponds to the ϕ-

instance that is relayed outside of $\mathcal{Z}_l^*$. In particular, we cannot immediately deduce that our induction claim holds for H_l.

Our solution to this complication is to *randomly shuffle the order of subprotocol invocations*. Concretely, $\mathcal{Z}_l^*$ uniformly selects a subset of $l-1$ subprotocol instances as ideal sessions, and another uniform instance as the instance that is relayed to the outside. As a consequence, runtime bounds derived for internally simulated ideal instances automatically apply to the instance that is relayed outside of $\mathcal{Z}_l^*$. (Namely, if only a single ideal instance did not adhere to bound T, then this instance is simulated inside $\mathcal{Z}_l^*$ and hence "caught" with probability $(l-1)/l$.) We can hence derive that also in H_l, all ideal subprotocol instances adhere to the fixed runtime bound T. By induction, we get that the system ρ^ϕ, running with simulator $\mathcal{S}^\infty$, is reactively polynomial-time as desired.

Of course, this exposition neglects a few technicalities. (In particular, most claims only hold with overwhelming probability.) The full proof will deal with these issues.

1.3 Some problematic use cases

To illustrate the kind of natural protocols that may be rejected by too restrictive a definition of polynomial time, we give two simple and natural examples of problematic protocol tasks. Recall that, since we strive for composability, we work in the UC framework. Hence, protocol tasks are specified as ideal functionalities (that reflect the ideally desired behavior). We will need these examples in Section 1.4 below to illustrate difficulties with other approaches to defining polynomial-time. We stress that these examples to not pose any problems with our notion; both are reactively polynomial-time.

Secure channels. A natural protocol functionality is that of a secure channel, again modeled as a single machine. For simplicity, let us say that the machine accepts only inputs of the form (`send`, *receiver*, *message*), and gives outputs of the form (`message`, *sender*, *message*) (where the semantics should be clear).

We stress that this ideal functionality may be activated arbitrarily often, with arbitrarily large *message* inputs. Hence, this functionality does not satisfy a polynomial-time notion that bounds the number of activations or the size of allowed inputs *a priori*. This eliminates most so far presented polynomial-time notions, except for (a) the variation on *a posteriori* polynomial-time bounds, (b) the notion from [HMQU05], (c) and the most recent polynomial-time definition of the UC model. In particular, all polynomial-time definitions that enforce an *a priori* runtime bound on machines do not allow to model a simple secure channel. In some situations, machines that just forward messages are also called repeaters or forwarders. These names are usually used when the machines are not intended to represent connections between parties, but instead are used as a technical tool in definitions or proofs.

A database functionality. Consider a publicly available centralized database, formalized as an ideal functionality, i.e., as a single database machine. The database machine accepts inputs of the form (`store`, *key*, *data*) and (`retrieve`, *key*), with the obvious semantics (namely, an input (`store`, *key*, *data*) stores *data* under *key*, and (`retrieve`, *key*) retrieves that data again).

We stress that this database machine may be activated arbitrarily often, with arbitrarily large (`store`) inputs. Hence, similar to the preceding case of a secure channel, this database machine does not satisfy a polynomial-time notion that *a priori* bounds the number of activations or the size of allowed inputs. Additionally, observe that the quotient of output and input size of database queries may be arbitrarily large: consider one party storing a large database entry and then another party retrieving it—the `retrieve` instruction itself is short, although the corresponding output may become arbitrarily large. This latter property prevents a modeling in the most recent version of the UC framework.[3]

Notice that the database functionality is reactive polynomial-time: in any given *a priori* polynomial-time protocol context, only a fixed polynomial number of `retrieve` queries can happen, each retrieving only a polynomially-sized piece of data. Thus the running time of the functionality is polynomial in any such context.

The same problems as with the database functionality also occur when considering an anonymous bulletin-board (as often used in remote voting protocols, e.g., [JCJ05, MCC08]). Here every user can post messages (which corresponds to storing an entry in the database), and every user can read the bulletin-board (which corresponds to retrieving an entry from the database).

1.4 Straightforward approaches and why they fail

An *a priori* polynomial bound on the overall runtime. Probably the most obvious approach is to allow only machines of polynomial (time) complexity as entities in a protocol run. That is, there is a fixed polynomial q_M, so that machine M halts and cannot be activated again after at most $q_M(k)$ overall steps. (Here and in the following, $k \in \mathbb{N}$ denotes the security parameter, that intuitively measures the "amount of desired security.") We assume that this bound is an *a priori* runtime bound; that is, we assume that q_M only depends on the machine M, but not on the context M is run in (in particular, not on the runtime of the machines M interacts with). This bound applies to honest protocol parties as well as to adversarial entities. In the UC setting, these are the adversary, the simulator, and the environment.

This approach has several disadvantages. First, it becomes impossible to formulate natural protocol tasks with an (*a priori*) unbounded number of activations (such as the examples from Section 1.3). This is a violation of flexibility.

An obvious workaround (extensively used, for instance, in the "cryptographic library" [BPW03]) would be to artificially bound *in advance* the number and size of inputs to a cryptographic system. For instance, a secure channel might shut down after a certain (fixed in advance) number of transmitted bits. We do not recommend this workaround: it might not be clear in advance how often, say, a secure channel will be used. Furthermore, this workaround creates the additional (intuitively unneccessary) hassle of

[3] Technically speaking, [Can05a] allows the database as a *functionality*, however it does not allow a *protocol party* with that behavior; in particular, this makes it impossible to implement this functionality, even when using a uncorruptible trusted party. See Section 3 for details.

fixing and keeping track of all concrete running time bounds. Strictly speaking, even the finally deployed protocol implementation would need to keep track of the number of its activations and stop working after a given time.

But there is a second, very severe technical drawback that becomes apparent when considering the composition of cryptographic protocols. Recall that in the UC security definition, the environment that represents the a larger protocol context, is chosen *last*. But if all protocol machines have *a priori* runtime bounds, there is an environment that can "exhaust" all protocol machines and even a given adversary, e.g., by sending them useless messages and force them to waste their limited runtime by processing them. This has been shown not only to cause severe technical artifacts. It actually renders many natural protocol tasks formally impossible when allowing only machines with *a priori* polynomial runtime bounds, cf. [HMQU05, Küs06].

An *a priori* polynomial runtime bound per activation. As a second straightforward approach, let us consider machines that perform only a polynomial number of steps in each activation (possibly even dependent on input size instead of security parameter), but may be activated an unbounded number of times. This overcomes the flexibility problems of forcing an upper *a priori* polynomial bound on the overall runtime. However, when considering a network of two machines, even if both machines are *a priori* polynomial-time per activation, the two machines can run infinitely long by activating each other over and over again.[4] Thus, the notion (at least if defined machine-wise) is not applicable to networks of machines. (And there is no obvious way to extend the notion to apply to networks as a whole.) Hence, we either need a definition of polynomial-time that composes in the sense that a network of polynomial-time machines is polynomial-time again, or, failing that, a definition that can be directly applied to the whole network (like our definition of reactive polynomial time and like the *a posteriori* definitions described below).

An *a posteriori* polynomial bound on overall runtime. This gives reason to consider machines that are polynomial-time *for any given machine (of arbitrary complexity)* they interact with. (For zero-knowledge, several such notions appear in the literature; an explicit discussion and analysis has been conducted in [Gol07].) We claim that, while an *a posteriori* runtime bound is useful in the zero-knowledge context, it does not constitute a good definition of polynomial runtime for *general* protocols.

For general protocols, by *a posteriori* runtime we mean that every protocol machine and the adversary run in polynomial time in every given (but arbitrary) context.

For instance, consider a secure channel R. Since we did not fix an *a priori* upper bound on the size of the incoming data, R forwards incoming data of *arbitrary length*. In particular, R runs in exponential time when interacting with a machine M that sends 1^{2^k} to R. Hence even a secure channel would also not satisfy the *a posteriori* polynomial

[4] If we allow a machine to send messages to itself, we can even get the same effect with a single machine activating itself. This is another indication that the notion of *a priori* polynomial time per activation does not capture the intuitive notion of polynomial time.

runtime definitions from the zero-knowledge case.[5,6] As above, this violates flexibility and, when also enforcing an *a posteriori* polynomial runtime bound for adversaries, it might endanger soundness. In fact, in the context of UC, the *dummy adversary*, which basically is the same as a secure channel/repeater, would not be allowed by an *a posteriori* polynomial runtime bound. The dummy adversary is an essential technical tool to prove composition theorems, cf. also Section 6. Hence, we also cannot guarantee composability.

A natural way to relax the *a posteriori* runtime bounds definition would be the following: one could allow machines M that have polynomial time complexity when running with any (*a priori*) *polynomial-time* machine M'.

Let us call this modified *a posteriori* notion *a posteriori polynomial-time in bounded contexts (APPT-BC)*. Note that the secure channel R from above is indeed APPT-BC. We claim that the APPT-BC notion enjoys flexibility: A protocol that does not run in polynomial-time when interacting with an *a priori* polynomial-time context cannot, intuitively, be considered polynomial-time. Thus any intuitively polynomial-time protocol is APPT-BC. Yet, the APPT-BC notion does not enjoy composability. First, two individually APPT-BC protocols are not necessarily APPT-BC when running together (consider two repeaters echoing each other's messages). This on its own is not necessarily a problem; the same happens with our notion of reactive polynomial time (see the discussion on composability on page 7). Second, a security notion based on APPT-BC does not even allow for secure composition of *one* protocol instance with a larger protocol. We prove this fact in Section 9.1.[7]

Notice, however, that APPT-BC is already very close to our notion of reactive polynomial time. The only difference is that in the definition of reactive polynomial time, we allow for a negligible probability that the protocol runs in superpolynomial time.

Acyclic runtime dependencies. One reason why definitions of polynomial runtime can be difficult is that two machines (e.g., secure channels) can be combined such that

[5]There is a subtlety here: by "polynomial-time," we mean polynomial in the (global) security parameter, whereas in the zero-knowledge case, it is customary to assume that "polynomial-time" means polynomial in the size of the input. However, in the context of general protocols, the former interpretation of "polynomial-time" is preferred, since it allows for a meaningful analysis of composed and nested protocols as well as protocols with constant input size like oblivious transfer.

[6]In fact, *a priori* and *a posteriori* polynomial runtime coincide when arbitrary, unbounded contexts are considered. Namely, say that a machine M runs at most q steps when running in a context C, where $q = q_C(k)$ is a polynomial (in the security parameter) that may depend on C. Then, there is a context C^* that *maximizes* M's runtime by, for each security parameter k, acting like $\operatorname{argmax}_C q_C(k)$. By definition, $q_{C^*}(k) \geq q_C(k)$ for all contexts C, and hence $q_{C^*}(k)$ is a *single* polynomial that bounds M's runtime in arbitrary contexts. Thus, M's runtime is already *a priori* polynomially bounded. (Note that $\operatorname{argmax}_C q_C(k)$ exists. Otherwise would could construct a context C^* with $q_{C^*}(k) \geq 2^k$ which would be a contradiction.) We conclude that we do not gain on generality by allowing *a posteriori* runtime bounds, at least when we consider arbitrary, *unbounded* contexts.

[7]The intuitive reason is that real and ideal protocol might behave identically only up to a small probability. Hence, real and ideal protocol might give slightly different (runtime) guarantees to adversary and environment. Now a larger protocol that uses the real, resp. ideal protocol as a subprotocol might *ensure* that the runtime of the real subprotocol will *always* be bounded, while the runtime of the ideal protocol will only *almost always* be. This can lead to a situation in which *any* successful simulation will *sometimes* (with negligible probability) require superpolynomial time.

they send messages back and forth and consume an unlimited amount of runtime. This problem can be solved by the following approach: In a network of machines, one defines an acyclic directed graph on the set of machines. If there is an edge from M' to M, we call M' the parent of M. Then we call a machine M polynomial-time if its running time is bounded by an fixed (*a priori*) polynomial in the total length and number of incoming messages sent by the parents of M . Incoming messages not coming from the parents of M are allowed, but do not increase the allowed running time of M.

Although this approach allows for more protocols than *a priori* polynomial-time (better flexibility), many protocols will still be rejected by such a definition as there is not distinguished direction in which messages flow. For example, a database functionality (described in Section 1.3 below) would not be considered polynomial-time because in some cases the database would need running time from the parties retrieving data from the database, and in some cases the parties retrieving the data would need running time from the database.

Another problem is that it is not clear which running time dependency should hold between the protocol, the environment, and the adversary or simulator. If the protocol gets running time from the adversary or simulator, the latter may be forced to terminate before the protocol run is complete, leading to soundness or completeness issues. If the adversary or simulator gets running time from the protocol, the protocol may be unable to react to messages arriving over the insecure network (that is controlled by the adversary), and hence some natural protocols will be disallowed (flexibility deficits). (The latest version of the UC framework [Can05a] uses a variant of this approach. Much of the complexity of the definition of polynomial-time there is due to the necessity to clarify which machine gets running time from which.)

Padding. Furthermore, the database example from Section 1.3 also illustrates why padding, a solution often advocated to circumvent the runtime restrictions in the UC framework [Can05a] is not always applicable. By padding we mean that a protocol specification or functionality expects inputs that are padded to a suitable length such that the machine receiving these inputs is allowed to run longer. In the case of the database functionality however, padding does not solve the problem, since a party retrieving an entry does not know in advance what the length of the data returned from the database will be, and thus that party cannot know how long a padding has to be used. Also the party storing the entry cannot add sufficient padding because it cannot know how many times the entry will be retrieved. It seems possible to *interactively* pad messages, in the sense that before giving the actual response, the database first requests padding of a suitable length from the user. However, this approach seems unneccessarily cumbersome. (More details on this problem are given in Section 3 which also gives further examples of problems that might occur with too restrictive notions of polynomial time.)

1.5 Previous work

In the context of zero-knowledge. Perhaps zero-knowledge protocols [GMR89] were the first example of protocol tasks for which it was recognized that *a priori* runtime

bounds lead to surprising artifacts. For instance, certain definitions of zero-knowledge require for every adversary the existence of an *expected polynomial-time* simulator. (There are also arguments for allowing expected polynomial-time adversaries [Fei90, Chapter 3].) This is so since the successful simulation of an attack may require a number of adversary rewindings that is only expected polynomial (as, e.g., in [GK96]). Goldreich [Gol07] gives an excellent overview over different flavors of expected polynomial-time notions in the zero-knowledge setting. We stress, however, that the issue of rewindings does not apply to our setting, in which we are dealing with an *interactive* environment. (Hence, the artifacts resolved by expected polynomial time are orthogonal to the artifacts in our case.)

Another artifact that does concern our case is the following *dependency problem* that arises with concurrent black-box zero-knowledge (e.g., [CKPR01]). Namely, with black-box zero-knowledge, a simulator should be independent in particular of the internal structure of the respective adversary. At the same time, however, the simulator should be applied to an arbitrarily large (polynomial) number of zero-knowledge sessions that may be scheduled and interleaved in an arbitrary way. In [CKPR01], this problem is solved by letting the simulator (and hence its complexity) depend *only* on the number of sessions. (This solution does not work in our case, since we want protocols as well as adversaries to be more powerful than just *a priori* polynomial-time. Hence letting one entity depend on the complexity of the other entities would lead to a cyclic definition.)

Length functions. Backes [Bac02] observed the technical artifacts that arise with *a priori* polynomial runtime bounds in conjunction with an interactive protocol environment (cf. also Section 1.4). His solution, which has been incorporated into the Reactive Simulatability framework (RSIM) [BPW04b], was to employ *length functions*, a technical tool to guard machines from being flooded with useless messages. This overcomes the soundness issues of straightforward approaches. Yet, since these RSIM machines still have an (*a priori*) polynomial upper bound on the overall runtime, this does not achieve flexibility. Natural tasks like that of a public key encryption system (that allows an unbounded number of encryptions) still cannot be expressed. Besides, length functions are a rather technical tool, that resulted from a technical artifact, and are intuitively not easily explainable.

Continuously polynomial time. In this situation, Hofheinz et al. [HMQU05] suggested to allow protocols that are, as a whole, polynomial time in their input size. This achieves flexibility. With a specific, dedicated restriction on allowed attacks, they also achieved (and demonstrated with examples) completeness and soundness of their definition. Namely, in their setting, neither protocols nor adversaries are required to ever terminate; however, the "relative computation speed" of adversary and protocol has to be polynomially related, and only polynomial-length execution prefixes are considered. However, they do not give a universal composition theorem that would allow for the composition of more than a constant number of protocol instances. Furthermore, their restriction of allowed attacks is somewhat unintuitive and lacks simplicity.

In the UC framework. In the Universal Composability (UC) framework [Can01,

Can05a] of Canetti, there are a number of approaches to define polynomial runtime. In the initial formulation [Can01], an *a priori* polynomial overall bound on the number of computational steps of each protocol entity was mandated. When the technical artifacts of this became clear, several definitions were proposed [Can04a, Can04b, Can05b]. The most recent[8] version [Can05a] of the UC framework uses a definition in which machines may be activated in principle infinitely often. However, at any point in time, a certain condition must be fulfilled that relates the runtime so far with the input/output behavior of that machine. In particular, the input which a machine M gives to other (sub-)machines must be smaller in size than the overall input M gets. This means that a protocol has to take care that its own input is large enough in size such that all necessary subprotocol invocations are allowed. In many cases, *padding* of the "top-level inputs" is necessary, which complicates the specification of natural tasks (see Section 3 for more details). In Section 3, we also show that composability might be a problem, since the technical tool of a (complete) dummy adversary is not available which however is used in the proof of the Universal Composition Theorem. Besides, the current UC notion of polynomial runtime is arguably somewhat complicated and not simple.

In the SPPC framework. In a different line of work, Datta et al. [DKMR05] propose different notions of polynomial runtime for cryptographic protocols in the SPPC framework [DKMR05]. In [DKMR05], a natural extension to the length function approach from [Bac02] is put forward. Specifically, where length functions merely allowed a machine to block messages from certain "spamming" senders, the *guards* from [DKMR05] allow a machine to specify *algorithms* that decide whether an incoming message is blocked or not. The computational steps used for deciding whether a message is blocked or not are *not* counted as computational steps of the receiving machine. However, the notion from [DKMR05] requires that machines still have an *a priori* polynomial upper runtime bound, thus inducing the same flexibility issues as with length functions.

In the IITM framework. In the IITM framework [Küs06], two variants of polynomial-time are proposed. The *IO-enriching* definition allows machines to have a running time which is polynomial in the total length of the inputs it got over so-called *enriching* input/output-channels. The definition additionally requires that the enriching channels form an directed acyclic graph, so that no two machines can give each other running time; this assures that the overall running time of the network stays polynomial. The *IO-network-enriching* definition additionally allows network channels, i.e., channels between different parties to be enriching. The IO-enriching definition bears a certain resemblance to that of [Can05a]. In particular, it also imposes the limitation that in a communication protocol, the running time of the recipient cannot depend on the length of the input of the sender (as that input would have to be transmitted through a network channel).[9] The IO-network-enriching definition resolves this problem; for this notion, however, no universal composition theorem is given that would allow for the composition of more than a constant number of protocol instances.

[8] See, however, the addendum below.

[9] Again, "padding" can be used here to circumvent this problem. See Section 3 for a description of the padding approach in the context of the UC model.

In the Task-PIOA framework. The Task-PIOA framework [CCK+06a, CCK+06b] proposes a simulation-based security notion that inherits many facets of the UC model. However, the Task-PIOA model uses different kinds of abstractions (in particular with respect to process scheduling) and thus allows for more flexible system specifications. The main focus of the Task-PIOA model thus lies not on polynomial runtime issues; the model of polynomial-time considered in [CCK+06b] requires that all machines are *a priori* polynomial-time per activation (that is, there is no limit on the number of activations, but in each activation, the running time is restricted by a fixed polynomial in the security parameter). The problem of two machines activating each other indefinitely is avoided by requiring the scheduler to only schedule a polynomial number of activations. This approach inherits many of the flexibility restrictions of the *a priori* polynomial-time approach; in particular, it is not possible to formulate a machine that implements a secure channel without imposing an *a priori* bound on the length of messages.

Other frameworks. In [Gol07], it is investigated whether the notion of expected polynomial time allows for composability. Although this question is answered positively, their approach does not allow machines to run in polynomial time in the length of the *incoming communication*. (It must be stated that allowing such protocols was not the aim of [Gol07], their goal was to give the simulator additional power which is needed in some settings.)

Summarizing, while previous notions can be used to express many natural protocols and protocol tasks, there are natural protocol tasks that *cannot* be expressed by existing notions (excluding perhaps [HMQU05], although [HMQU05] is not known to support *universal* composability).

Addendum. After submitting the initial version of our manuscript, we were made aware of a revision of the Universal Composability framework [Can08]. This revision features a modified definition of polynomial runtime that improves upon that of the earlier UC version [Can05a]. In particular, a suitable variant of the dummy adversary can be proven complete in the setting of [Can08]. On the other hand, padding of inputs as described above is still necessary; the examples given in Section 3 apply.

Technically, the revision [Can08] largely follows the earlier version [Can05a]. The crucial modification is to consider only *balanced environments*. Essentially, balanced environments send at least as much (up to a *fixed* polynomial factor) data to the adversary as to the protocol. A balanced environment thus guarantees that a simulator is invoked sufficiently often, and with sufficiently long inputs to produce a successful simulation. (A similar technique has been used in [HMQU05].) This allows to prove the dummy adversary complete. In particular, the counterexample to the completeness of the dummy adversary from [Can05a] we point out in Section 3 does not work in the setting of [Can08]. Our counterexample crucially employs *non-balanced* environments.

Besides, the notion of a polynomial-time machine is slightly simplified compared to [Can05a]. This change is technical in nature and keeps the spirit of the definition from [Can05a]. Specifically, this simplified definition still requires that inputs are suitably padded in order to allow subprotocol invocations.

1.6 Organization

After introducing some notation, we review the Universal Composability framework (in which our work takes place) and the UC composition theorem in Section 2. We motivate our work by highlighting the problematic aspects of previous polynomial runtime notions in Section 3. Our own polynomial runtime notion is presented in Section 4. In Section 5 and Section 6, we prove some basic but important properties of our notion, which will turn out useful in the proof of the composition theorem in Section 7. Section 8 gives an example of our notion in action. In Section 9, we discuss two variations of our notion. Finally, Section 10 relates our notion to the standard UC definitions.

1.7 Notation

We say an algorithm A is polynomial-time if A's runtime is bounded by a polynomial in the length of A's *first* input (assuming that A's input is a tuple of bitstrings). This notation facilitates the use of a security parameter k, since we will usually pass 1^k as the first argument. Two ensembles $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ of probability distributions are statistically indistinguishable, if there is a negligible function μ such that for all $k \in \mathbb{N}$, $z \in \{0,1\}^*$, the statistical distance between $X(k,z)$ and $Y(k,z)$ is bounded by $\mu(k)$. Two ensembles $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ are computationally indistinguishable (written $X(k,z) \approx Y(k,z)$) if for every nonuniform probabilistic polynomial-time algorithm C there exists a negligible function μ such that for all $k \in \mathbb{N}$, $z \in \{0,1\}^*$ we have that $\big|\Pr[D(1^k, z, X(k,z)) = 1] - \Pr[D(1^k, z, Y(k,z)) = 1]\big| \leq \mu(k)$.

2 The UC framework

We briefly review the framework proposed by [Can01]. We omit the details that are orthogonal to our result, for these we refer to [Can01]. An interactive Turing machine (ITM) is a Turing machine that has additional tapes for incoming and for outgoing communication.[10] An ITM may be activated by a message on an incoming communication tape. At the end of an activation, the ITM may send a message on an outgoing communication tape to another ITM. The recipient of a message is addressed by the unique ID of that ITM. The actions of an ITM may depend on a global parameter $k \in \mathbb{N}$, the so-called security parameter. (One can, e.g., assume that the security parameter is stored on a special tape of the ITM.)

A network is modeled as a (possibly infinite) set of ITMs. Such a set of ITMs we

[10] Actually, the UC framework distinguishes various types of incoming and outgoing communication tapes, namely tapes for input, output, subroutine invocation, subroutine results, incoming messages and outgoing messages. These distinctions are necessary to formulate the notion of polynomial-time given in [Can05a]. However, these distinctions are immaterial for our definition of polynomial time, thus we will only consider incoming and outgoing communication tapes in this exposition.

call a system of ITMs.[11,12] We call a system S of ITMs executable if it contains an ITM $\mathcal{Z}$ with a distinguished input and output tape. An execution of S with input $z \in \{0,1\}^*$ and security parameter $k \in \mathbb{N}$ is the following random process: First, $\mathcal{Z}$ is activated with the message z on its input tape. Whenever an ITM $M_1 \in S$ finishes an activation with an outgoing message m addressed to another ITM $M_2 \in S$ on its outgoing communication tape, the other ITM M_2 is invoked with incoming message m on its incoming communication tape. If an ITM terminates its activation without an outgoing message the ITM $\mathcal{Z}$ is activated. If an ITM sends a message to a non-existing ITM, $\mathcal{Z}$ is activated with that message. $\mathcal{Z}$ may send messages in the name of any non-existing machine.[13] When the ITM $\mathcal{Z}$ sends a message on its output tape, the execution of S terminates. The output of $\mathcal{Z}$ we denote by $\mathrm{EXEC}_S(k,z)$ (where we set $\mathrm{EXEC}_S(k,z) := 0$ if the execution does not terminate).[14] Furthermore, by $\mathrm{TIME}_S(k,z)$ we denote the total number of steps executed by all ITMs in S. If the execution does not terminate, we set $\mathrm{TIME}_S(k,z) := \infty$. Further we write $\mathrm{TIME}_S(k,z,M)$ for the total number of steps executed by the ITM $M \in S$. Given a system of ITMs π (representing a protocol) and two ITMs $\mathcal{Z}$ (environment) and $\mathcal{A}$ (adversary) we will usually write $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ and $\mathrm{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ for $\mathrm{EXEC}_{\pi\cup\{\mathcal{A},\mathcal{Z}\}}(k,z)$ and $\mathrm{TIME}_{\pi\cup\{\mathcal{A},\mathcal{Z}\}}(k,z)$.

We stress that ITMs are probabilistic machines, in the sense that they possess a random tape that contains uniformly and independently distributed bits. This makes $\mathrm{EXEC}_S(k,z)$ and $\mathrm{TIME}_S(k,z,M)$ random variables, where the probability space is defined by the contents of the random tapes of all machines.

A network without the machine $\mathcal{Z}$ and without an adversary (the adversary is simply defined as being an ITM with a special id) is called a protocol.

Using the above network model, security is usually defined by comparison. We define an ideal protocol ϕ (formally a system of ITMs) that usually consists only of one machine, a so-called ideal functionality. Then we define what it means that another protocol π (securely) emulates ϕ.

Definition 4 (UC – classical definition) *Let π and ϕ be systems of polynomial-time ITMs. We say that π* emulates *ϕ if for any polynomial-time ITM $\mathcal{A}$ (the adversary) there exists a polynomial-time ITM $\mathcal{S}$ (the simulator) such that for any polynomial-time*

[11] Infinite systems are necessary to allow e.g., for systems where an arbitrary number of instances of a given ITM can be invoked. In the case of infinite systems we require the system to be uniform in the sense that given the ID of an ITM, we can compute the code of that ITM in deterministic polynomial-time.

[12] We stress that this notion of systems differs from the one introduced in [Can05a]. In our setting, following [Can01], a system of ITMs is plainly a set of machines. In the setting of [Can05a], a system contains an initial ITM and a so-called control function; all other ITMs come into existence when the initial ITM specifies their code (in a sense, the initial ITM "programs" the other ITMs). Specifically, [Can01] allows machines that "pop up into existence," while we do not. Again, this is no restriction (since we allow infinite sets of ITMs), but simply a more static way to think about protocol executions. This will make it easier for us to specify runtime properties of systems of ITMs. We stress that our results apply analogously when using the modeling from [Can05a] where ITMs are created dynamically.

[13] We allow $\mathcal{Z}$ to impersonate non-existing ITMs to simplify the formulation of Definition 10 below.

[14] Since our modeling will guarantee that all valid systems will terminate with overwhelming probability, the value of $\mathrm{EXEC}_S(k,z)$ in the case of non-termination is unimportant. We arbitrarily fix 0 for concreteness.

ITM $\mathcal{Z}$ (the environment) the following families of random variables are computationally indistinguishable:

$$\left\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*} \qquad \textit{and} \qquad \left\{\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*}$$

Note that for this definition to be complete, we have to specify what we mean by polynomial-time machines. In classical definitions of UC [Can01], polynomial-time machines are assumed to run a polynomial number of steps in the security parameter (we call this *a priori* polynomial-time; cf. Definition 8 below). Other approaches define other meanings of polynomial-time, see e.g., [Can05a].

For a complete definition of the UC framework, many more details must be specified, e.g., how secure and insecure channels are modeled, how messages are scheduled, how the adversary can corrupt parties, etc. Since these aspects are orthogonal to the results in this paper, we refer the interested reader to [Can05a].

2.1 The Composition Theorem

Arguably, one of the most important properties of the UC framework is its universal composition theorem. The composition theorem guarantees that whenever a protocol π emulates some ideal functionality $\mathcal{F}$, we can use π instead of $\mathcal{F}$ in any larger protocol context without losing security.

We will illustrate this with a small example. Assume that $\mathcal{F}_{\mathrm{COM}}$ is a functionality for commitments (it is not necessary for this example to know how this functionality is designed). Assume further that we are given some protocol π that emulates $\mathcal{F}_{\mathrm{COM}}$. Now we design a protocol $\rho^{\mathcal{F}_{\mathrm{COM}}}$ that uses the ideal commitment $\mathcal{F}_{\mathrm{COM}}$ and implements some more complex functionality $\mathcal{G}$. Since $\mathcal{F}_{\mathrm{COM}}$ is an ideal commitment, no cryptography is involved in using $\mathcal{F}_{\mathrm{COM}}$ (in particular, we have perfect hiding and binding properties). This greatly simplifies the proof that $\rho^{\mathcal{F}_{\mathrm{COM}}}$ implements $\mathcal{G}$. In some cases, $\rho^{\mathcal{F}_{\mathrm{COM}}}$ might not use any cryptography at all, and the security proof can be done by an information theoretical argument. Unfortunately, since $\mathcal{F}_{\mathrm{COM}}$ is an ideal assumption, $\rho^{\mathcal{F}_{\mathrm{COM}}}$ cannot be implemented in a real life setting. Instead one has to replace all calls to $\mathcal{F}_{\mathrm{COM}}$ by calls to the protocol π. The question arises whether the resulting protocol ρ^{π} still securely emulates $\mathcal{G}$.

Here the universal composition theorem of the UC framework comes into play. It guarantees that if π emulates $\mathcal{F}$, then ρ^{π} emulates $\rho^{\mathcal{F}}$. Since we also know that $\rho^{\mathcal{F}}$ emulates $\mathcal{G}$, it follows that ρ^{π} implements $\mathcal{G}$ (using the transitivity of the security notion) and hence ρ^{π} is a secure protocol for the task described by $\mathcal{G}$.

Note that without the composition theorem, we would have had to analyze ρ^{π} in one go instead of analyzing the simpler protocols π and $\rho^{\mathcal{F}}$ individually.

In order to state the universal composition theorem, one first needs to define the operation of composing, i.e., one needs to specify the meaning of constructions of the form ρ^{π}. We will now give an informal definition and refer to [Can05a] for details.

Definition 5 (Composition – informal) *Let a protocol π and a protocol ρ be given. Assume that the machines in ρ send messages to the machines in π. Then let ρ^{π} be*

the protocol that contains the machines from π and from ρ. In ρ^π, the machines in π are modified such that instead of expecting messages from the environment $\mathcal{Z}$ and sending messages to $\mathcal{Z}$, they expect messages from machines in ρ and send the answers to machines in ρ. (That is, π plays the role of the environment for ρ.) Furthermore, ρ can invoke arbitrarily many instances of π. We assume that the invocations of π are tagged with a session id that identifies the instance of π, and that the answers produced by an instance of π carry the same session id. New instances of π spring into existence whenever a new session id is used for the first time (by ρ or by the adversary).[15]

This definition also specifies the meaning of $\rho^{\mathcal{F}}$ for an ideal functionality $\mathcal{F}$ since a functionality is just a special case of a protocol.

Note that ρ is allowed to invoke arbitrarily many instances of π. In our example above, this would mean that ρ is allowed to use an arbitrary number of commitments instead of just a single one.

Using this notation, we can formulate the universal composition theorem of [Can05a].

Theorem 6 (Universal composition theorem) *Let π, ϕ, and ρ be* a priori *polynomial-time protocols. Assume that π emulates ϕ. Then ρ^π emulates ρ^ϕ.*

There is also a weaker variant of the universal composition theorem, which we call the simple composition theorem. Here we require that ρ invokes only one instance of π or ϕ, respectively.

Note the restriction that π, ϕ, and ρ have to be *a priori* polynomial-time. It is easy to see that the composition theorem does not hold if no computational restriction is put on these protocols.[16] Yet, the restriction to *strict* polynomial time is a strong one; one of the goals of this paper is to find a variant of the UC definition where this restriction is relaxed.

We give a short proof sketch of the universal composition theorem from [Can05a] to enable comparisons with our proof of the universal composition theorem in the case of reactive polynomial time (Section 7).

Proof sketch (of Theorem 6). Assume π, ϕ, and ρ as in Theorem 6, and let $\mathcal{A}$ denote the *dummy adversary*, i.e., an adversary that only executes orders from the environment $\mathcal{Z}$, and reports its own view to $\mathcal{Z}$. By assumption, π emulates ϕ, so that there exists a simulator $\mathcal{S}$ such that

$$\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \tag{1}$$

for all *a priori* polynomial-time environments $\mathcal{Z}$. (Here $\approx$ denotes computational indistinguishability.) Hence, informally, $\mathcal{S}$ emulates attacks on (one instance of) π, while actually running with (one instance of) ϕ.

[15] Formally, all possible instances of π are already present from the beginning and are only activated if needed. This is the reason why we need systems to be possibly infinite. However, for the intuition it is often easier to assume that machines are created when needed.

[16] Even if π emulates ϕ, the protocols might be distinguishable by an unbounded machine. Then an unbounded ρ can be constructed that determines whether it is running as ρ^π or ρ^ϕ and gives different output accordingly.

Our goal is to show that ρ^π emulates ρ^ϕ. The dummy adversary is complete in the sense that without loss of generality, it is the only adversary that needs to be considered (see Section 6 for a detailed discussion). Hence it suffices to construct a simulator $\mathcal{S}^\infty$ with

$$\text{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}} \tag{2}$$

for any *a priori* polynomial-time $\mathcal{Z}$.

Recall that the dummy adversary $\mathcal{A}$ only collects information and executes orders. Hence, the dummy adversary $\mathcal{A}$ attacking ρ^π can be seen as a combination of several dummy adversaries, namely dummy adversaries $\mathcal{A}_i$ that only attack one instance of subprotocol π each, and a dummy adversary $\mathcal{A}_\rho$ that only attacks ρ itself. (See Figure 1(a).) Each $\mathcal{A}_i$ is "responsible" for messages from one π-instance.

We will construct $\mathcal{S}^\infty$ as a combination of $\mathcal{A}_\rho$ and several $\mathcal{S}$-instances, one for each invoked instance of subprotocol ϕ. Similarly to protocol ρ^π, each $\mathcal{S}_i$ is responsible for messages from one ϕ-instance in ρ^ϕ. (See Figure 1(b).) Since each $\mathcal{S}$-instance by assumption simulates attacks performed on one π-instance, while running together with one ϕ-instance, this intuitively achieves that $\mathcal{S}^\infty$ simulates attacks on *many* π-instances, while running together with many ϕ-instances.

Now the only difference between ρ^π and ρ^ϕ is precisely that in ρ^π, all ϕ-instances of ρ^ϕ have been replaced with π-instances. Hence, $\mathcal{S}^\infty$ simulates attacks on ρ^π, while actually running with ρ^ϕ. To formally show that this holds, we have to *reduce* the fact that $\mathcal{S}^\infty$ is a good simulator to our assumption, namely the fact that $\mathcal{S}$ is a good simulator.

To this end, we will assume an arbitrary environment $\mathcal{Z}$ and show that

$$\text{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}. \tag{3}$$

We apply a hybrid argument. Namely, consider the hybrid network H_l which is a "mix" of real and ideal network in the following sense. H_l consists of $\mathcal{Z}$ and ρ, where the first l of ρ's subprotocol invocations are connected to an instance of ϕ (with simulator $\mathcal{S}_i$), and the remaining subprotocol instances are connected to an instance of π (with dummy adversary $\mathcal{A}_i$). The situation is depicted in Figure 1(c). In this notation, (3) is equivalent to

$$\text{EXEC}_{H_0} \approx \text{EXEC}_{H_{p(k)}},$$

where $p(k)$ is the number of subprotocol instances that ρ invokes. We will show (3) by a hybrid argument. More specifically, we will show that for $0 \leq l \leq p(k)$, we have

$$\text{EXEC}_{H_l} \approx \text{EXEC}_{H_{l+1}}. \tag{4}$$

Informally, this means that "changing one subprotocol instance from π to ϕ does not make a difference." However, our assumption that π emulates ϕ guarantees that changing *a single* subprotocol instance from π to ϕ does indeed *not* make a difference. All that remains is to formalize this intuition.

We thus build an environment $\mathcal{Z}_l^*$ that encompasses the whole hybrid network H_l, only without the $(l+1)$-th subprotocol instance (the part of Figure 1(c) enclosed by a dashed line). Hence, running $\mathcal{Z}_l^*$ with π and $\mathcal{A}$ yields an execution of H_l, and running

$\mathcal{Z}_l^*$ with ϕ and $\mathcal{S}$ yields an execution of H_{l+1}. Our assumption (1) on $\mathcal{S}$ hence guarantees that

$$\mathrm{EXEC}_{H_l} = \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_l^*} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_l^*} = \mathrm{EXEC}_{H_{l+1}},$$

which shows (4). Hence (3) holds, which means that we have proved our goal (2).

Finally, we also have to prove that our constructed simulator $\mathcal{S}^\infty$ is allowed in the sense that $\mathcal{S}^\infty$ is polynomial-time as required by Definition 4. For *a priori* polynomial-time notions this is usually easy to verify, since the combination of polynomially many polynomial-time machines always yields a polynomial-time machine.

3 Difficulties with prior notions

In order to illustrate the difficulties that can arise when trying to model polynomial time in UC-like notions, we will sketch a few of the problems that arise in prior notions of polynomial time. We will concentrate on difficulties with the UC framework of [Can05a]. However, we stress that we simply chose this example since [Can05a] is the most well-known and popular model. For instance, in the Reactive Simulatability (RSIM) framework [BPW04b], some of these issues are solved using so-called length-functions which are also known to lead to difficulties (see, e.g., [HMQU05]).

Network model. We first sketch very roughly how polynomial time is modeled in [Can05a]. Our description is far from complete but it should be sufficient to understand the examples below. The ITMs in a network are arranged in a hierarchy of invocation. The top level contains the environment $\mathcal{Z}$. The second level contains the machines directly invoked by $\mathcal{Z}$, namely the adversary (or simulator) and the protocol machines. Further levels might include subroutines of the protocol machines (these subroutines may, e.g., result from the composition, in this case they are the ITMs comprising the subprotocol). Finally, the lowest level will usually contain the functionalities, which are modeled as subroutines shared by different ITMs. There are two kinds of communication in the network. We have vertical communication between a machine and its subroutines, called subroutine input and subroutine output. And we have horizontal communication between different machines, which represents messages sent over the network. Commonly, these messages will be sent between machines on the same level or between machines on any level and the adversary. The adversary communicates with the environment using vertical communication (since protocol and adversary are considered subroutines of the environment), and with protocol machines using horizontal communication (since this represents communication over the network). The auxiliary input of $\mathcal{Z}$ is considered a subroutine input for $\mathcal{Z}$.

Polynomial time definition. In this setting, we model polynomial time by requiring the following property of any ITM in the network (cf. Definition 3 in [Can05a] for details and motivation):

Definition 7 (Canetti-PPT) *An ITM M is PPT in the sense of [Can05a] (short: Canetti-PPT) if and only if M runs in time which is polynomial in $n := k+n_I-n_O-k\cdot n_N$.*

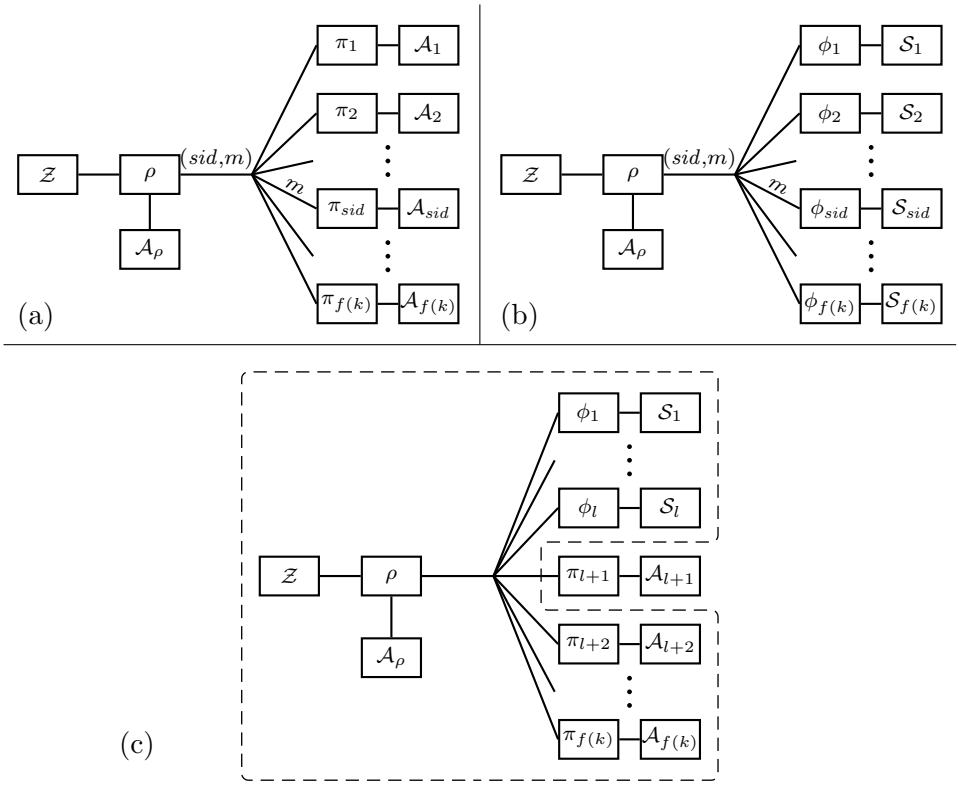

Figure 1: Relevant networks for the proof of Theorem 6. (a) depicts environment $\mathcal{Z}$, running with protocol ρ^π and dummy adversary $\mathcal{A}$. For presentation, $\mathcal{A}$ is split up into dummy adversaries $\mathcal{A}_\rho$ and $\mathcal{A}_i$ for protocol ρ and all respective π-instances. (b) illustrates $\mathcal{Z}$ running together with ρ^ϕ and the simulator $\mathcal{S}^\infty$ constructed during the proof. For presentation, $\mathcal{S}^\infty$ is split up into adversaries $\mathcal{A}_\rho$ and $\mathcal{S}_i$ for ρ and the respective ϕ-instances. (c) shows (surrounded by a dashed line) the hybrid environment $\mathcal{Z}_l^*$ used in the reduction that proves the settings (a) and (b) indistinguishable (from $\mathcal{Z}$'s point of view).

(That is, there is a fixed polynomial p such that the number of M's computational steps taken so far never exceeds $p(n)$.) Here k is the security parameter, n_I the total length of the subroutine inputs received from a higher level, n_O the total length of subroutine outputs passed to a lower level, and n_N the number of ITMs that M communicates with.

Note that we will always have $n_O \leq n_I$ (since otherwise $n < 0$), i.e., we cannot send longer inputs to subroutines than we get from a higher level. This is why it is necessary that $\mathcal{Z}$ gets some initial subroutine input, namely the auxiliary input.

Padding. At a first glance it might seem that the requirement that no ITM can call subroutines with inputs longer than the inputs of that ITM itself is very restrictive. However, this is solved by the use of padding: when designing a protocol and the corresponding ideal functionality, one requires all inputs to contain a padding of sufficient length such that the protocol machines are able to call their subroutines/functionalities. For example, a functionality $\mathcal{F}$ for secure message transmission would expect an input of the form $(m, 1^{t(|m|)})$ where t is a polynomial that depends on the protocol we would like to use to implement $\mathcal{F}$. Although an explicit treatment of this padding can be cumbersome in some cases, it at least allows to write protocols without an *a priori* bound on their runtime.

However, an example for a protocol where the use of padding meets its limits is the case of the database functionality $\mathcal{D}$ described in Section 1. This functionality represents a publicly available centralized database. The functionality $\mathcal{D}$ accepts queries of the form (`store`, *key*, *data*) and (`retrieve`, *key*). Upon `retrieve`, the *data* previously stored with *key* is returned. As a functionality, this machine is Canetti-PPT even without any padding (it does not invoke subroutines, so $n_O = 0$, and thus the functionality is allowed to run in polynomial time in the total length of the queries).

However, even simple protocol machines that *use* the database $\mathcal{D}$ may not be polynomial-time any more. For instance, consider a party P_1 that wants to copy the entry stored at key_1 to key_2. With the current specification of the database functionality, this is only possible by retrieving the data *data* stored at key_1 and then storing *data* under key key_2. However, to do so, P_1 needs to run $\Omega(|data|)$ steps. Thus the *input* (e.g., from the protocol environment $\mathcal{Z}$) of P_1 needs a padding whose length is dependent on $l := |data|$. For one, this length l might not be known in advance (it depends on the inputs of other protocol parties), so it is unclear how to specify the length of the padding P_1 expects. It seems possible to *interactively* let P_1 ask its own environment for a suitably long padding depending on the size l of the database entry. However, these solutions are (seemingly unnecessarily) cumbersome and might make the analysis more complicated. Furthermore, even if we would model P_1 to have an interactive protocol interface that, e.g., first requests additional padding of sufficient length and then copies the data, this might have implications on the simulatability of the protocol: in some cases, whether and to what extent the database is used might have to be hidden from the environment; for example, if in the real and the ideal model, a different number of queries to the database is performed by some larger protocol.

Dummy adversary and composition. A very instructive case is the question

whether the dummy adversary is complete. Intuitively, the dummy adversary is an adversary that simply does what it is told by the environment and forwards all messages received from the protocol to the environment. By completeness of the dummy adversary we mean that it is sufficient to consider only the dummy adversary as a real adversary $\mathcal{A}$ in the UC security definition Definition 4. (See Section 6 for a detailed exposition.) Validity[17] and completeness of such a dummy adversary is crucial for the proof of the Universal Composition Theorem. Unfortunately, a machine as in Definition 16 that just forwards messages in both directions is not Canetti-PPT (i.e., it is not valid) since it may have to forward messages that come from the protocol, i.e., via horizontal communication. In order to handle this problem, [Can05a] proposes to define the dummy adversary $\tilde{\mathcal{A}}$ as follows:

- When asked by the environment $\mathcal{Z}$ to send a message m to the protocol, that message m is sent. (Since $\mathcal{A}$ is a subroutine of $\mathcal{Z}$, this is permitted.)
- When receiving a message m from the protocol, the adversary $\tilde{\mathcal{A}}$ first sends $l := |m|$ to $\mathcal{Z}$. If it then receives 1^l from $\mathcal{Z}$, it sends m to $\mathcal{Z}$.

This definition now allows to forward arbitrary messages, however, it raises the following difficulties: First, it is very sensitive to the machine and network model. In particular, for $\tilde{\mathcal{A}}$ to compute $l = |m|$, it is necessary that messages are always prefixed with their length (otherwise $\tilde{\mathcal{A}}$ will take time $\Omega(l)$ for measuring l). Further, it is necessary that m is still accessible when 1^l is received from the environment, although $\tilde{\mathcal{A}}$ did not have the runtime to copy m to some working tape. However, assuming a suitable machine model, these problems are easily solvable. More problematic is the second difficulty: The dummy adversary is not complete, i.e., security with respect to the dummy adversary does not imply security with respect to arbitrary Canetti-PPT adversaries.[18] Note that this poses a problem for two reasons: First, the dummy adversary is a very useful construct when proving the security of concrete protocols, allowing to consider only a single adversary, and second, the proof of the Universal Composition Theorem in [Can05a] uses the dummy adversary in an integral way (however, we do not know whether only the proof or the theorem itself is invalidated).

To see that the dummy adversary from [Can05a] is really *not complete* (in contradication to [Can05a, Claim 10]), assume a function f with the following property: We have $|f(t,x)| = |x|$, and $f(t,x)$ can be computed in time polynomial in $t + |x|$, but for any polynomial p, there is a polynomial $\tilde{p}$ such that $f(\tilde{p}(k), x)$ cannot be computed probabilistically in time $p(k)$ given a uniformly chosen $x \in \{0,1\}^k$. (more exactly, in time $p(k)$, the probability of guessing $f(\tilde{p}(k), x)$ is negligible). A candidate for such a function would be, e.g., applying some suitable hash function t-times to x.

We then define the protocol π to expect a message $(1^t, x)$ with $|x|, 2^t \leq k$ from $\mathcal{Z}$

[17] We say that an adversary $\mathcal{A}$ is valid if $\mathcal{A}$ is considered in the (UC) security definition, i.e., if $\mathcal{A}$ is in the set of "allowed" adversaries.

[18] This contradicts Claim 10 on page 45 of [Can05a]. The mistake in their proof was the assumption that the simulator $\mathcal{S}$ constructed there is always Canetti-PPT.

and then to send $(1^t, f(t,x))$ to the adversary.[19] Further, we define the protocol ϕ to expect a message $(1^t, x)$ with $|x|, 2^t \leq k$ from $\mathcal{Z}$ and then to send (t,x) to the adversary. Note that both π and ϕ are Canetti-PPT.

First, we show that π emulates ϕ with respect to the dummy adversary. The dummy adversary first sends the number $t + |f(t,x)| = t + |x|$ to the environment and only when receiving $1^{t+|x|}$, it sends $(1^t, f(t,x))$ to the environment. Thus the corresponding simulator also sends $t + |x|$ to the environment, and when receiving $1^{t+|x|}$, it computes $f(t,x)$ and sends $(1^t, f(t,x))$ to the environment. The simulator is Canetti-PPT since computing $f(t,x)$ and sending $(1^t, f(t,x))$ takes time polynomial in the length of $1^{t+|x|}$.

Now, we show that π does not emulate ϕ with respect to arbitrary Canetti-PPT adversaries. For a polynomial $\tilde{p}$, let $\mathcal{Z}_{\tilde{p}}$ be an environment that chooses a random $x \in \{0,1\}^k$ and sends $(1^{\tilde{p}(k)}, x)$ to the protocol. Let $\mathcal{A}$ be an adversary, that upon receipt of $(1^t, f(t,x))$ forwards $f(t,x)$ to the environment. Now a suitable simulator has to compute $f(t,x)$ from (t,x). Since the simulator has a fixed runtime polynomial p, there is a $\tilde{p}$ such that $f(\tilde{p}(k), x)$ cannot be computed in time $p(k)$. Thus, in an interaction with $\mathcal{Z}_{\tilde{p}}$, that simulator will return $f(t,x) = f(\tilde{p}(k), x)$ only with negligible probability, allowing $\mathcal{Z}$ to distinguish real and ideal model. Thus π does not emulate ϕ.

Dummy parties. A useful construct in UC-like security definitions is that of a dummy party. Such a dummy-party is used when considering a single ideal functionality as the protocol, for each player we then introduce a dummy-party that forwards the messages between the functionality and the environment. These parties are very useful, e.g., for modeling corruptions (in particular in the adaptive case) in the ideal model. (In [Can05a] such dummy-parties are introduced on page 51 under the caption "Ideal protocols".) However, since dummy-parties have to forward messages from the functionality to the environment, they are not Canetti-PPT. An interactive padding convention would have to be introduced similar to those used with the dummy adversary, but in this case the same padding convention would have to be followed by the parties in the real protocol since otherwise the environment could trivially distinguish the real and the ideal model.

Combining machines. A technical tool that is needed in many situations when working with UC-like security definitions is to construct a machine that simulates internal submachines. As noticed by [Bac02], the resulting machine is not polynomial-time, at least with respect to an *a priori* polynomial-time notion as in [Can01]: Assume that a machine M simulates two submachines M_1, M_2. Assume further that n messages are sent to M where n is larger than the runtime polynomial of M. Then M will, since it is *a priori* polynomial-time, have to stop even reading incoming messages. If then a message is sent to M_2, M will not be able to notice and answer. Since given two separate machines M_1, M_2, M_2 will not stop reacting just because we send many messages to M_1, it follows that M does not correctly simulate M_1, M_2.

Summary. We want to stress again that the problems mentioned in this section

[19] Depending on the exact machine model, we might also send 1^t and $f(t,x)$ in two separate messages if receiving a very long 1^t might make accessing $f(t,x)$ impossible.

do not compromise the essence of the results of [Can05a]. E.g., probably no "reasonable" cryptographic protocol will fail to compose because of quirks in the modeling of polynomial time; most results in the UC setting are robust with respect to the details of the modeling. However, to put these results on exact and rigorous foundations, it is necessary to develop a model of polynomial time that does not lead to any formal inconsistencies.

4 Our definition of polynomial runtime

In order to define a computational security notion, we first have to fix a definition of polynomial time. Classically, an ITM is considered to be polynomial-time if it runs in polynomial time in the security parameter. This notion we will call *a priori* polynomial time:

Definition 8 (*A priori* polynomial time) *An ITM M runs in* a priori polynomial time *if there is a polynomial p such that for any sequence of incoming messages, M runs at most $p(k)$ steps with probability 1 upon security parameter k.*[20]

However, as seen in the introduction, this definition is far from being flexible enough. Many protocols that are intuitively considered to be polynomial-time are rejected by this definition, e.g., a secure channel functionality or a database. Investigating these examples, we see that what we intuitively expect from a polynomial-time protocol is that when the protocol is used in an *a priori* polynomial-time context, the whole system still runs in polynomial time. For example, although a channel is not *a priori* polynomial-time (cf. Section 1.3), a channel can be implemented in polynomial time if the messages sent through it are generated by an *a priori* polynomial-time machine.

To capture even more protocols, we can slightly relax the condition, and only require that the whole system runs in polynomial time *with overwhelming probability*.[21] The resulting notion is maybe the weakest notion of polynomial time that still makes sense. Any weaker definition would allow for protocols that interact with an *a priori* polynomial-time environment and run a superpolynomial number of steps with non-negligible probability. We call this notion reactive polynomial time, and it is formalised by the following two definitions.

Definition 9 (Polynomial time with overwhelming probability) *An executable system S of ITMs runs in* polynomial time with overwhelming probability *if there is a polynomial p and a negligible function μ such that for all $k \in \mathbb{N}, z \in \{0,1\}^*$ we have $\mathrm{TIME}_S(k,z) > p(k)$ with probability at most $\mu(k)$.*

Definition 10 (Reactive polynomial time) *A system S of ITMs runs in* reactive polynomial time *if for any* a priori *polynomial-time ITM $\mathcal{Z}$ the system $S \cup \{\mathcal{Z}\}$ runs in polynomial time with overwhelming probability.*

[20] Remember that the program of M may depend on the globally known security parameter k.

[21] It turns out that this relaxation is indeed necessary for our security notion, see Section 9.1.

We remark that in this definition, $\mathcal{S}$ can impersonate any machine that the machines in S could ever run with (cf. footnote 13). For example, if S is a protocol and does not contain an adversary, then $\mathcal{Z}$ also controls messages that are sent over the (insecure) network (by impersonating the adversary). And if S already contains an adversary, then $\mathcal{Z}$ can only control the protocol inputs and outputs. In particular, Definition 10 makes sense both for protocols S without adversary, and systems S that include a protocol *and* an adversary.

We will comment below (after giving the security definition) on how easy it is to show that a system is reactively polynomial-time.

Why the generality? One of the main features (and design goals) of Definition 10 is its permissiveness. Essentially, Definition 10 stipulates only that a protocol should "behave well" in any (a priori) polynomial-time context. One particularly instructive example on the benefits of such a permissive notion is the case of a *repeater* machine. Namely, consider a machine R that simple relays its input verbatim to a dedicated output channel. Repeaters are "natural" machines in the sense that inserting them into an existing protocol should intuitively not make any difference. At the same time, repeaters can be powerful technical tools: the dummy adversary (see the proof of Theorem 6) is essentially a repeater; besides, during security proofs, it can be useful to insert a repeater to "split up a party in two". A similar case can be made for *secure channels*, which can be thought of as repeaters that hand extra information (e.g., the message length) to the adversary.

It is easy to see that Definition 10 allows repeaters or secure channels even as stand-alone protocols. On the other hand, as we briefly outline in Section 3, secure channels (and also repeaters) are not Canetti-PPT without modifications. Of course, one could think of augmenting, say, the Canetti-PPT runtime definition to explicitly consider repeaters as "polynomial-time." However, explicitly allowing repeaters would still only permit machines that are precisely repeaters, but not machines that are "essentially" repeaters, such as a secure message transmission functionality. Similarly, the dummy-adversary, who is essentially a repeater, would not be precisely a repeater, because it does some multiplexing and header-rewriting (messages to/from all machines of the protocol are forwarded through a single "connection" to the environment).

Of course, one could relax the definition of a repeater and explicitly allow certain forms of recoding and multiplexing. But it would seem that the resulting definitions would not be very simple any more, and in particular would be more difficult to use. More generally, in Section 3 (on page 24), we give another, more complex example of a functionality that requires the generality offered by Definition 10. Concretely, we provide a natural database functionality that is reactively polynomial-time, but cannot be directly modeled as Canetti-PPT.

Closure properties of reactive polynomial time. Notice that the notion of reactive polynomial time is not closed under the composition of networks. More precisely, if we have two reactively polynomial-time systems S_1, S_2 of ITMs, then $S_1 \cup S_2$ is not necessarily reactively polynomial-time. For example, S_1 could contain a machine R_1 that, when receiving a message x from any machine, sends x to a machine R_2 (not contained

in S_1). And S_2 could contain a machine R_2 that, when receiving a message x, forwards x to R_1. Both S_1 and S_2 are easily seen to be reactively polynomial-time. But in $S_1 \cup S_2$, as soon as R_1 receives a message x (say, from the environment), R_1 and R_2 enter an infinite loop in which they forward the message x to each other. (A more drastic case would be the one where R_1, given x, forwards $x\|x$, a string of twice the length.)

Such a lack of closure can, of course, be troublesome, because it implies that whenever we compose protocols or networks, we need to check again that the resulting system is reactively polynomial-time. We believe, however, that this lack of closure is a necessary evil: If we wish to admit all networks that are "intuitively polynomial-time", we have to admit networks such as S_1 and S_2. Notice that each of them, on its own, is harmless; the bad (non-polynomial-time) behavior arises only from the interaction of the two. Thus, unless we wish to impose strong conditions on the networks, we have to accept the lack of closure properties.

In the results of this paper the issue is handled by an extra condition in the composition theorem (Theorem 21) that requires us to check whether the composed protocol is reactively polynomial-time before applying the composition theorem. See the discussion on page 39; there we also discuss how our results apply to more restricted classes of protocols that are closed under composition.

Is the notion too permissive? At a first glance, this notion might seem *too* permissive (i.e., too general). One might argue that the system S is allowed a running time k^{2^c}, where k^c is the running time of $\mathcal{Z}$ for some constant c. It might seem that such constructions lead to too powerful a system S of possibly exponential runtime. However, this is not the case, since our definition guarantees that the overall network, and thus in particular S, will always run in polynomial time in k (Lemma 12 below). The seeming power only stems from the fact that the polynomial that bounds the running time may depend on $\mathcal{Z}$, thus there is no polynomial p *independent of* $\mathcal{Z}$ such that S runs in polynomial time in $p(k+t)$ where t is the running time of $\mathcal{Z}$.

We remark that this absence of a uniform polynomial bound p reflects the modeling of existing notions of zero-knowledge and simulatability. For example, in [Gol01], the definition of (non black-box) zero-knowledge is—roughly—formulated as follows: for any polynomial-time verifier there is a polynomial-time simulator such that the verifier's and the simulator's output is indistinguishable. In particular, the running time of the simulator does not have to be polynomially bounded in the running time of the verifier. Instead, it is only required that if the verifier runs in polynomial time, so does the simulator. In particular, the simulator might run, e.g., $t^{\log_k t}$ steps[22] where t is the running time of the (simulated) verifier and k the length of the common input x. This is analogous to our modeling if we identify the verifier's runtime with that of $\mathcal{Z}$ and the length of the common input with the security parameter.

However, if a uniform bound on the running time of S is needed, it is possible to strengthen the notion in a way that disallows an *arbitrary* dependency on $\mathcal{Z}$'s complexity. Namely, a stricter definition, called *uniform reactive polynomial time*, is also conceivable:

[22] Note that this should not be confused with the quasipolynomial $t^{\log t}$ which would not be allowed.

The runtime of S has to be bounded by $p(k+q)$ with overwhelming probability where q is the runtime of $\mathcal{Z}$ and p is a polynomial *independent of* $\mathcal{Z}$. (In contrast, Definition 10 allows p to depend on $\mathcal{Z}$.) Indeed, uniform reactive polynomial time is as suitable a notion of polynomial time as reactive polynomial time, and we show in Section 9.2 that the results of this paper also hold for that notion. We have chosen Definition 10 as our main notion because—although this may not be obvious at a first glance—it better reflects how polynomial-time is classically modeled in cryptography. We want to stress however, that this is just a design choice and that we prove all our results for both notions.

Why allow a negligible error? In Definition 10 we have introduced the notion of a reactively polynomial-time network S roughly as follows: For any ITM $\mathcal{Z}$, the network $S \cup \{\mathcal{Z}\}$ is *polynomial-time with overwhelming probability.* However, the reader might question whether the additional generality of allowing networks that run in superpolynomial time with negligible probability is not offset by the added complexity. Instead, we could require $S \cup \{\mathcal{Z}\}$ to be *a priori* polynomial-time; the resulting notion we call strong reactive polynomial time. Replacing reactive polynomial time by strong reactive polynomial time in Definition 11, we get a seemingly simpler security definition. Unfortunately, it can be shown that the resulting security definition does not fulfill the Universal Composition Theorem (Theorem 21). See Section 9.1 for additional details and proofs.

After giving the security definition, we will comment on why we do not allow a negligible error in the runtime guarantees of the context $\mathcal{Z}$. (Essentially, the answer is: "because we would not gain anything.")

Security notion. Equipped with the notion of reactive polynomial time, we can now look for a variant of the UC notion that can handle arbitrary reactively polynomial-time protocols (i.e., we want that all the usual properties like the composition theorem hold for reactively polynomial-time protocols). To design such a UC variant, we first have to specify what machines should be considered valid adversaries and simulators. With classical notions, a valid adversary/simulator would run in *a priori* polynomial time. However, this is not sufficient in our context, since in this case the adversary/simulator might have to terminate before the protocol. In this case the real protocol might continue to work without adversary, whereas the ideal protocol might rely on a simulator, making a successful simulation impossible (examples for such ideal protocol tasks are the public-key encryption functionality $\mathcal{F}_{\text{PKE}}$ and the signature functionality $\mathcal{F}_{\text{SIG}}$, cf. [Can01]). Hence, we instead try to find the largest class of adversaries/simulators for a given protocol such that the definition still makes sense, i.e., such that the overall system does not run in superpolynomial time. Obviously, we minimally require that the adversary and the protocol together are still reactively polynomial-time. It will turn out that this requirement is also sufficient to get the properties we expect from a UC notion (see the following sections). We therefore call an adversary/simulator valid if the network consisting of adversary/simulator and the real/ideal protocol is reactively polynomial-time. Finally, we have to specify which environments to allow. To ensure that the overall protocol is still at least polynomial-time with overwhelming probability, we require an

a priori polynomial environment. Note that in contrast to the adversary/simulator, an *a priori* polynomial-time environment is fully sufficient, since intuitively its task is to observe whether there is some polynomial p such that the real and the ideal protocol can be distinguished within time p. Combining these observations into a single definition, we propose the following variant of the UC definition that can handle reactively polynomial-time protocols:

Definition 11 (UC with respect to reactive polynomial time) *We say an ITM M is* valid *for π (or ϕ) if $\pi \cup \{M\}$ (or $\phi \cup \{M\}$) runs in reactive polynomial time.*

Then π emulates *ϕ* (with respect to reactive polynomial time) *if for any ITM $\mathcal{A}$ that is valid for π, there is an ITM $\mathcal{S}$ that is valid for ϕ such that for every* a priori *polynomial-time ITM $\mathcal{Z}$ the following families of random variables are computationally indistinguishable:*

$$\left\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*} \qquad \textit{and} \qquad \left\{\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*}$$

In the following, we will simply say "UC" and "emulate" instead of "UC/emulate with respect to reactive polynomial time".

Note that there might be other possibilities how to model a UC definition that can handle reactively polynomial-time protocols (e.g., one could define that an adversary $\mathcal{A}$ is valid if for *all* reactively polynomial-time protocols π, the network $\pi \cup \{\mathcal{A}\}$ is reactively polynomial-time). However, all other variants the authors have considered seem to break at least one of the properties that we minimally expect from a viable UC variant (i.e., the composition theorem holds, the relation is transitive and reflexive, and no networks running in superpolynomial time with non-negligible probability occur).

Note further that we only claim that our security definition makes sense when considering reactively polynomial-time protocols. If we apply the definition to unbounded protocols, unexpected effects may occur (e.g., the set of valid adversaries may be empty).

Why not allow a negligible error for the runtime bounds of the protocol context? Given that it is essential to allow a negligible error for the runtime bounds of protocol and adversary, the question arises why the runtime bound for the protocol context $\mathcal{Z}$ in Definition 11 has to hold with probability 1 (by Definition 8). Alternatively, one could allow environments $\mathcal{Z}$ that run in polynomial time only with overwhelming probability. We do not pursue this variation further because it leads to an equivalent Definition 11: Any $\mathcal{Z}$ that runs in *a priori* polynomial-time *except* with negligible probability $\mu(k)$ can be substituted with an *a priori* polynomial-time $\mathcal{Z}'$ that behaves like $\mathcal{Z}$, except with probability $\mu(k)$. Hence $\mathcal{Z}'$ distinguishes real and ideal protocol whenever $\mathcal{Z}$ does.

Similarly, one might allow $\mathcal{Z}$ to run in *a posteriori* polynomial time (see page 11). This would lead to an equivalent Definition 11, too, by an argument analogous to that given in footnote 6.

And if we instead quantify over environments $\mathcal{Z}$ that are APPT-BC (cf. page 12), then we might loose completeness as no guarantees can be made about the running time

of $\mathcal{Z}$ when running with $\pi \cup \{\mathcal{A}\}$ or $\phi \cup \{\mathcal{S}\}$ (since these networks are not necessarily *a priori* polynomial time).

How easy is it to show reactive polynomial time? Since we are interested in actually analyzing protocols, it is crucial that it is easy to check whether a given protocol, adversary or simulator is allowed in our setting. For all concrete protocols and ideal functionalities that we are aware of, this is easy to check: these protocols consist of a fixed polynomial number of rounds (for each protocol invocation or input) with messages and running time that are of polynomial size in the respective protocol input. (Ideal functionalities are generally even easier to handle, since they consist only of one machine.) Thus we immediately get that the protocol runs in polynomial time with any *a priori* polynomial-time $\mathcal{Z}$. The validity of adversaries and simulators may, at first glance, be harder to verify. After all, nothing is known *a priori* about a real adversary $\mathcal{A}$, and it is not immediately clear how the complexity of $\mathcal{A}$ would be in, say, a blackbox simulation inside the corresponding simulator $\mathcal{S}$.

Fortunately, there is a very simple real adversary, the so-called dummy adversary that we can restrict ourselves to, cf. Section 6. It suffices to give a good simulator for this *dummy adversary*. Thus, security can be proven by analyzing only a single simulator. All concrete constructions of such simulators that we are aware of are in fact valid in the sense of Definition 11. (In fact, since in many simulator descriptions occurring in the literature, there is no discussion of when the simulator actually *halts*, they may not be considered polynomial-time in any of the stricter notions of polynomial time occurring in prior work.)

Relation to classical notions. Furthermore, the reader might ask in what relation our notion stands to the classical UC definitions. Since the classical definitions are not meaningful for protocols that are not *a priori* polynomial-time, we are interested in the case that π and ϕ are *a priori* polynomial-time protocols. In this case, it turns out that UC with respect to reactive polynomial time lies strictly between two common classical definitions: UC and specialized-simulator UC[23]. That is, if π emulates ϕ with respect to classical UC, this strictly implies that π emulates ϕ with respect to reactive polynomial time, which in turn strictly implies that π emulates ϕ with respect to classical specialized-simulator UC. We believe that the fact that UC with respect to reactive polynomial time lies strictly between two established notions gives additional evidence that our notion indeed captures intuitive security requirements. See Section 10 for additional details and proofs.

5 Basic properties

In this section, we state some simple but important properties of our definition.

[23]Specialized-simulator UC is defined like UC, with the difference that the simulator may depend on the environment. We stress that we consider the specialized-simulator UC notion as defined by [Lin03], which is *not* equivalent to the UC notion from [Can05a]. There also exists a specialized-simulator UC variant in [Can05a] that *is* equivalent to standard UC (see [Can05a, Claim 12]).

Efficient executions. The first lemma guarantees that the executions $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ that are considered in Definition 11 do not run in superpolynomial time.

Lemma 12 *Let π be a (not necessarily reactively polynomial-time) protocol, $\mathcal{A}$ an adversary or simulator that is valid for π, and $\mathcal{Z}$ an* a priori *polynomial-time environment. Then there is an* a priori *polynomial-time probabilistic Turing machine M such that $M(1^k, z)$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ are statistically indistinguishable in k.*

Proof. Since $\mathcal{A}$ is valid for π, $\pi \cup \{\mathcal{A}\}$ is reactively polynomial-time. Since $\mathcal{Z}$ is *a priori* polynomial, it follows that $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ is polynomial-time with overwhelming probability. So there is a polynomial p such that $\text{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k, z) < p(k)$ with overwhelming probability. By letting $M(1^k, z)$ simulate $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ for at most $p(k)$ steps, the lemma follows. □

Reflexivity and transitivity. A very important property of UC-type security definitions which is often underestimated is that the relation of emulation is reflexive and transitive. A non-reflexive relation (i.e., a protocol does not emulate itself) would at least raise some doubts about the meaningfulness of the definition.[24] A non-transitive relation strongly lessens the usefulness of the composition theorem. For example, a typical use case of the composition theorem is the following: We have that π emulates ϕ and ρ^ϕ emulates τ (where ϕ and τ usually are ideal functionalities). Using the composition theorem we then get that ρ^π emulates ρ^ϕ which emulates τ. By transitivity, it follows that ρ^π emulates τ. It may seem that transitivity is a trivial property, but surprisingly many of our approaches failed this property.

Lemma 13 (Reflexivity, transitivity) *Let π, ϕ and ρ be protocols. Then π emulates π (reflexivity), and if π emulates ϕ and ϕ emulates ρ, then π emulates ρ (transitivity).*

Proof. We first show reflexivity: If $\mathcal{A}$ is a valid adversary for π, then $\mathcal{S} := \mathcal{A}$ is a valid simulator for π, and for all $\mathcal{Z}$ we have $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} = \text{EXEC}_{\pi,\mathcal{S},\mathcal{Z}}$, so π emulates π.

We now show transitivity: Let $\mathcal{A}$ be a valid adversary for π. Then, since π emulates ϕ, there is a valid simulator $\mathcal{S}$ for ϕ such that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\pi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable for all *a priori* polynomial-time $\mathcal{Z}$. Then $\mathcal{A}' := \mathcal{S}$ is a valid adversary for ϕ, so since ϕ emulates ρ, there is a valid simulator $\mathcal{S}'$ for ρ such that $\text{EXEC}_{\phi,\mathcal{A}',\mathcal{Z}}$ and $\text{EXEC}_{\rho,\mathcal{S}',\mathcal{Z}}$ are computationally indistinguishable for all *a priori* polynomial-time $\mathcal{Z}$. Using the transitivity of the computational indistinguishability, we see that for every $\mathcal{A}$ valid for π there is a $\mathcal{S}'$ valid for ρ such that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\rho,\mathcal{S}',\mathcal{Z}}$ are computationally indistinguishable for all *a priori* polynomial-time $\mathcal{Z}$. Thus π emulates ρ. □

On generalizations of transitivity. Successive application of Lemma 13 yields for any *constant* n that π_1 emulates π_n whenever π_i emulates π_{i+1} for all $1 \leq i < n$. We

[24] Unless, of course, the non-reflexivity is only due to syntactical reasons, e.g., if the ideal protocol is formally required to consist of a functionality.

cannot hope for more (e.g., if n is polynomial in the security parameter k). Namely, consider an infinite sequence $\pi_1, \pi_2, \ldots$ of protocols such that π_i emulates π_{i+1} for all i. Let $p(k)$ be any function with $\lim_{k\to\infty} p(k) = \infty$. In this situation, one might hope that π_1 emulates $\pi_{p(k)}$, where $\pi_{p(k)}$ is the protocol that behaves like $\pi_{p(i)}$ when invoked with security parameter $k = i$. (Such a form of transitivity would be extremely useful, e.g., to avoid "full-fledged hybrid arguments," and instead focus on two individual hybrid systems.) However, this "generalized transitivity" does *not* hold in general. For instance, say that π_i outputs 1 on security parameter $p(k) = i$, and 0 otherwise. Note that this implies that π_i emulates π_{i+1} for any *fixed* i. However, π_1 outputs 0 almost always, and $\pi_{p(k)}$ outputs 1 always.

Note that this impossibility is not a property specific to our definition, the example given here works with essentially any security notion unless it uses concrete security bounds.

One-bit output without loss of generality. Finally, the following lemma states that without loss of generality we can consider only environments that give a single bit of output. While this property is not necessary for a useful security definition (and indeed, some UC-like security notions do not fulfill it, e.g., specialized-simulator UC [Lin03]), it can sometimes be convenient to assume that the output consists of a single bit, and some authors even define the UC notion with respect to one-bit output.

Definition 14 (Emulation with respect to one-bit output) *We say that π emulates ϕ with respect to one-bit output, if Definition 11 applies when quantifying only over environments $\mathcal{Z}$ that give a single bit of output.*

Lemma 15 *π emulates ϕ with respect to one-bit output if and only if π emulates ϕ.*

Proof. By definition, UC implies UC with respect to one-bit output. So we only have to show the opposite direction. Assume that π does not emulate ϕ. Then (using the definition of computational indistinguishability), there is a valid adversary $\mathcal{A}$ for π such that for every valid simulator for ϕ, there exists an *a priori* polynomial-time environment $\mathcal{Z}$, a nonuniform probabilistic polynomial-time algorithm D and a sequence $z_k \in \{0,1\}^*$, such that $\left|\Pr[D(1^k, z_k, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z_k)) = 1] - \Pr[D(1^k, z_k, \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k, z_k)) = 1]\right|$ is not negligible. Let for the moment $\mathcal{A}$ and $\mathcal{S}$ be fixed. For the nonuniform probabilistic polynomial-time algorithm D, there is a uniform probabilistic polynomial-time algorithm $\hat{D}$ and a sequence d_k of strings of polynomial length such that $\hat{D}(1^k, d_k, z_k, x) = D(1^k, z_k, x)$. Let $\hat{\mathcal{Z}}$ be the environment that upon security parameter 1^k and auxiliary input (d_k, z_k) simulates $\mathcal{Z}$ with auxiliary input z_k. When $\mathcal{Z}$ would give output x, then $\hat{\mathcal{Z}}$ gives output $D(1^k, d_k, z_k, x)$.

Let $\hat{z}_k := (d_k, z_k)$. Then $\Pr[\mathrm{EXEC}_{\pi,\mathcal{A},\hat{\mathcal{Z}}}(k, \hat{z}_k) = 1] = \Pr[D(1^k, z_k, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z_k)) = 1]$ and $\Pr[\mathrm{EXEC}_{\phi,\mathcal{S},\hat{\mathcal{Z}}}(k, \hat{z}_k) = 1] = \Pr[D(1^k, z_k, \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k, z_k)) = 1]$. Thus, we get that $\left|\Pr[\mathrm{EXEC}_{\pi,\mathcal{A},\hat{\mathcal{Z}}}(k, \hat{z}_k) = 1] - \Pr[\mathrm{EXEC}_{\phi,\mathcal{S},\hat{\mathcal{Z}}}(k, \hat{z}_k) = 1]\right|$ is not negligible. Summarizing, we have that there is a valid adversary $\mathcal{A}$ such that for any valid simulator $\mathcal{S}$ there exists an *a priori* polynomial-time environment $\hat{\mathcal{Z}}$ such that $\mathrm{EXEC}_{\pi,\mathcal{A},\hat{\mathcal{Z}}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\hat{\mathcal{Z}}}$

are not computationally indistinguishable. Thus π does not emulate ϕ with respect to one-bit output. □

6 Dummy Adversary

A very useful tool for dealing with a UC-like definition is the concept of the dummy adversary.

Definition 16 (Dummy Adversary) *The dummy adversary is the following machine. Whenever it receives a message from the protocol (this may include control messages like the responses to corruption requests), it forwards that message to the environment (including the id of the sender of the message). When it receives a message from the environment to send a given message to a given recipient (which may be a normal message, or a control message like a corruption request), the dummy adversary sends that message to the required recipient.*

The usefulness of the dummy adversary stems from the fact that for many variants of the UC definition (including ours, see below) one can without loss of generality consider only the dummy adversary (we say, the dummy adversary is complete). This has several advantages. First, security proofs can be formulated much simpler, since we can assume a single given adversary and construct a simulator for that given adversary (instead of formulating a generic transformation from adversaries to simulators). Second, even with classical UC definitions, the proof of the universal composition theorem uses the dummy adversary (at least if we allow polynomially many instances of the subprotocol). And third, some authors find it more intuitive to define security directly with respect to the dummy adversary.

Furthermore, in our situation, the dummy adversary has additional advantages. First, even the proof of the simplest case of the composition theorem (where only a single instance of the subprotocol may be invoked) heavily depends on the completeness of the dummy adversary. Second, the security definition as formulated in Definition 11 may be hard to handle, since it requires us to prove the existence of a valid simulator for every valid adversary. Since the definition of validity depends on the protocols under consideration, it may be very difficult to find a simple characterisation of the set of all adversaries. However, when using the dummy adversary, such a characterisation is not necessary, and it is sufficient to construct a concrete valid simulator for this concrete and simple adversary.

However, despite the seeming simplicity of the concept of the dummy adversary, some care has to be taken. In the classical UC notion, the adversary is required to be *a priori* polynomial-time. Since the dummy adversary does not have any *a priori* bound on the length or number of messages it delivers for the environment, it is not *a priori* polynomial-time. So in the classical UC notion one instead has to consider a family of dummy-adversaries that are parametrized over the maximum number and length of messages they can transmit. This introduces additional complexity into proofs using the

dummy adversary. Fortunately, it turns out that for our UC variant such a family of dummy-adversaries is not necessary since for every reactively polynomial-time protocol, the dummy adversary is valid.

Lemma 17 (Validity of the dummy adversary) *If π is a reactively polynomial-time protocol, the dummy adversary is valid for π.*

Proof. Assume that the dummy adversary $\tilde{\mathcal{A}}$ was not valid. Then there is an *a priori* polynomial-time ITM $\mathcal{Z}$ such that $\pi \cup \{\tilde{\mathcal{A}}, \mathcal{Z}\}$ is not polynomial-time with overwhelming probability. Since $\tilde{\mathcal{A}}$ only forwards messages between $\mathcal{Z}$ and π, we can construct an *a priori* polynomial-time ITM $\mathcal{Z}'$ that directly sends and receives those messages to and from π. Then $\mathcal{Z}' \cup \{\pi\}$ is not polynomial-time with overwhelming probability. This is a contradiction to the fact that $\mathcal{Z}'$ is *a priori* polynomial-time and π is reactively polynomial-time.[25] □

Of course, the validity of the dummy adversary does not yet ensure its usefulness. Instead, we need to be able to consider without loss of generality only the dummy adversary. This is guaranteed by the following theorem.

Definition 18 (Emulation with respect to the dummy adversary) *We say* π emulates ϕ with respect to the dummy adversary *if there is an ITM $\tilde{\mathcal{S}}$ that is valid for ϕ such that for every* a priori *polynomial-time ITM $\mathcal{Z}$ the ensembles* $\mathrm{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}$ *and* $\mathrm{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}}$ *are computationally indistinguishable. Here $\tilde{\mathcal{A}}$ denotes the dummy adversary.*

Theorem 19 (Completeness of the dummy adversary) *Assume that π is reactively polynomial-time. Then π emulates ϕ if and only if π emulates ϕ with respect to the dummy adversary.*

Proof. Assume that π emulates ϕ. Since the dummy adversary $\tilde{\mathcal{A}}$ is valid for π by Lemma 17, it directly follows that π emulates ϕ with respect to the dummy adversary.

Assume now that π emulates ϕ with respect to the dummy adversary $\tilde{\mathcal{A}}$. Let $\tilde{\mathcal{S}}$ be the corresponding simulator, i.e., $\tilde{\mathcal{S}}$ is valid for ϕ and the ensembles $\mathrm{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}}$ are computationally indistinguishable for any *a priori* polynomial-time $\mathcal{Z}$.

To show that π emulates ϕ we have to show that for any valid adversary $\mathcal{A}$, there is a valid simulator $\mathcal{S}$ such that the ensembles $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable for any *a priori* polynomial-time $\mathcal{Z}$. Let therefore $\mathcal{A}$ be an adversary that is valid for π, and let $\mathcal{Z}$ be an *a priori* polynomial-time environment. We will construct a valid simulator for ϕ that depends only on $\mathcal{A}$ (and not on $\mathcal{Z}$). The network consisting of π, $\mathcal{Z}$ and that adversary $\mathcal{A}$ is depicted in Figure 2 (a).

Since $\mathcal{A}$ is valid and $\mathcal{Z}$ is *a priori* polynomial-time, the network $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ is polynomial-time with overwhelming probability. In other words, there is a polynomial p such that $\mathrm{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k, z) \leq p(k)$ with overwhelming probability for all $z \in \{0,1\}^*$ and $k \in \mathbb{N}$.

[25] We stress that that by Definition 10, $\mathcal{Z}'$ may impersonate the adversary when running with π.

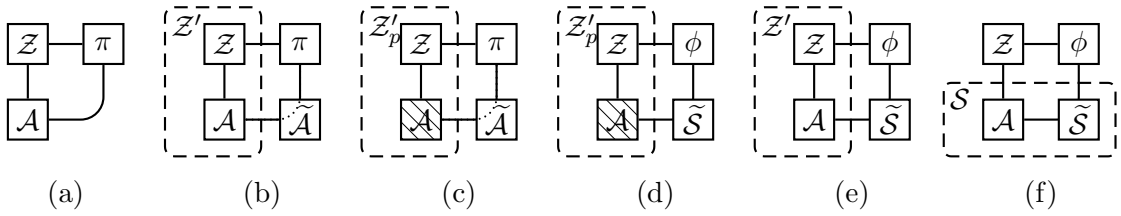

Figure 2: Networks in the proof of the completeness of the dummy adversary. The hatched background of machine $\mathcal{A}$ in (c) and (d) denotes an enforced runtime bound of $p(k)$.

We now construct the environment $\mathcal{Z}'$ which is supposed to run with the dummy adversary $\tilde{\mathcal{A}}$. The environment $\mathcal{Z}'$ simulates the original environment $\mathcal{Z}$ and the adversary $\mathcal{A}$. Whenever $\mathcal{A}$ sends a message to the protocol π, the environment $\mathcal{Z}'$ instead instructs the dummy adversary $\tilde{\mathcal{A}}$ to send that message. Conversely, whenever the dummy adversary $\tilde{\mathcal{A}}$ informs the environment $\mathcal{Z}'$ of an incoming message, that message is passed to the simulated adversary $\mathcal{A}$.

Obviously, the resulting network (cf. Figure 2 (b)) is a faithful simulation of the original network, in other words, $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} = \text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}$.

Now we modify $\mathcal{Z}'$ as follows, resulting in a new environment $\mathcal{Z}'_p$: When the running time of the simulated $\mathcal{A}$ exceeds $p(k)$, then $\mathcal{Z}'_p$ terminates with a special output `beep` (we assume that $\mathcal{Z}$ never outputs `beep`). Since $\text{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k,z) > p(k)$ only with negligible probability, the modified environment $\mathcal{Z}'$ terminates with output `beep` only with negligible probability (when running with π and $\tilde{\mathcal{A}}$, cf. Figure 2 (c)). Therefore $\text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'}$ and $\text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'_p}$ are computationally indistinguishable (in fact even statistically indistinguishable). Note further that since $\mathcal{Z}$ is *a priori* polynomial-time, and the simulated $\mathcal{A}$ runs at most $p(k)$ steps, the environment $\mathcal{Z}'_p$ is *a priori* polynomial-time, too.

Thus, since π emulates ϕ with respect to the dummy adversary, $\text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'_p}$ and $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'_p}$ (cf. Figure 2 (d)) are computationally indistinguishable.

Since $\text{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}'_p} = \texttt{beep}$ only with negligible probability, $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'_p} = \texttt{beep}$ holds only with negligible probability. Therefore we can replace $\mathcal{Z}'_p$ by $\mathcal{Z}'$, and thus $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'_p}$ and $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'}$ (cf. Figure 2 (e)) are computationally indistinguishable.

By constructing a simulator $\mathcal{S}$ that simulates both $\mathcal{A}$ and $\tilde{\mathcal{S}}$, we get the situation depicted in Figure 2 (f). Since this is essentially just a regrouping of machines, we have $\text{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'} = \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$.

Summarising our results so far, we have that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable. Note that this holds for any $\mathcal{Z}$, and that the construction of $\mathcal{S}$ does not depend on $\mathcal{Z}$.

It is left to show that $\mathcal{S}$ is valid for ϕ. Since $\tilde{\mathcal{S}}$ is valid for ϕ, the network $\phi \cup \{\mathcal{Z}'_p, \tilde{\mathcal{S}}\}$ is polynomial-time with overwhelming probability (Figure 2 (d)). Since the network $\phi \cup \{\mathcal{Z}'_p, \tilde{\mathcal{S}}\}$ behaves differently from $\phi \cup \{\mathcal{Z}', \tilde{\mathcal{S}}\}$ (Figure 2 (e)) only if $\mathcal{Z}'_p$ output `beep` which happens with negligible probability, the network $\phi \cup \{\mathcal{Z}', \tilde{\mathcal{S}}\}$ is polynomial-time

with overwhelming probability, too. Then also $\phi \cup \{\mathcal{S}, \mathcal{Z}\}$ (Figure 2 (f)) is polynomial-time with overwhelming probability. Since this holds for any *a priori* polynomial-time $\mathcal{Z}$, it follows that $\phi \cup \{\mathcal{S}\}$ is reactively polynomial-time, and therefore $\mathcal{S}$ is valid for ϕ. □

7 Universal Composition Theorem

Arguably the most salient property of the UC security definition (and other security definitions of the same kind like RSIM [PW01, BPW04b]) is the so-called composition theorem. The composition theorem guarantees that we can securely replace an ideal functionality with its implementation. More formally, the composition theorem states that whenever π emulates ϕ, then ρ^π emulates ρ^ϕ. The composition theorem is a well-known result for classical UC notions and comes in two flavors. One flavor allows ρ to invoke an arbitrary number of instances of the subprotocol π or ϕ, respectively (*universal composition theorem*), while the other, more restricted flavor requires ρ to invoke only a single instance of the subprotocol (called the *simple composition theorem* in the following). It is known that for some variants of the UC notion only the simple composition theorem holds [HU06]. For UC with respect to reactive polynomial time, however, the universal composition theorem holds (see below) of which the simple composition theorem is a direct consequence. Nevertheless, since the proof of the universal composition theorem is quite involved, here we start with the conceptually simpler theorem for simple composition. We believe that reading the proof for this simple composition theorem first will help the reader to familiarize himself with the setting and our model before attempting to go through the more involved proof of the universal composition theorem.

Theorem 20 (Simple Composition Theorem) *Let π, ϕ and ρ be protocols. Assume that π emulates ϕ. Assume that ρ calls only one subprotocol instance. Assume further that π and ρ^π are reactively polynomial-time. Then ρ^π emulates ρ^ϕ.*

On the assumptions in the composition theorem(s). We remark that there is an interesting asymmetry in the preconditions in Theorem 20 (and in the universal composition theorem, Theorem 21, below). Namely, it is required that π and ρ^π are reactively polynomial-time, while ϕ and ρ^ϕ need not be. Although probably protocols which are not reactively polynomial-time will not be used in applications of the composition theorem, the absence of additional proof obligations may make proofs that use the composition theorem simpler.

We stress that our security notion is not subject to the counterexample of [Can05a, pp. 65–66]. This counterexample exhibits a Canetti-PPT-protocol π that realizes an ideal functionality $\mathcal{F}$ that is not Canetti-PPT. The example then goes on to show that already two instances of π (in concurrency) do no longer realize two instances of $\mathcal{F}$. In our setting, however, π would not realize $\mathcal{F}$ in the first place. Namely, recall that we require the existence of a simulator $\mathcal{S}$ such that the ideal system $\phi \cup \{\mathcal{S}\}$ is reactively

polynomial-time. (I.e., it is the responsibility of the simulator to make ϕ polynomial-time.) However, in case of the example from [Can05a], the system $\mathcal{F} \cup \{\mathcal{S}\}$ is not reactively polynomial-time for *any* simulator $\mathcal{S}$.

On the assumption that ρ^π is reactively polynomial-time. An important point is the fact that we have to show that the composed protocol ρ^π is reactively polynomial-time before we can show that it is secure. This is an extra assumption compared, e.g., to the composition theorem of [Can05a]. In their setting, ρ^π is automatically polynomial-time as soon as ρ and π are. In our setting, this may not be the case (so in a certain sense, the definition of reactive polynomial time itself does not compose). However, we stress that in most practical situations, it is very easy to show that the composed protocol is reactively polynomial-time, while the security is the interesting property. We believe that this additional proof obligation is a necessary result of the high generality of our approach. In particular, one can easily derive a version of this composition theorem that does not have this condition: When restricting the protocols to some subclass of reactively polynomial-time protocols that is closed under composition (e.g., those studied in [DKMR05, Can05a]) one automatically gets a composition theorem without this condition as a corollary of Theorem 20 (and a universal composition theorem without this condition as corollary of Theorem 21).

Alternatively, the follow-up work [HS11] gives a "dual" version of our composition theorems Theorem 20 and Theorem 21. Namely, the (universal) composition theorem in [HS11] assumes that the *ideal* composed protocol ρ^ϕ (and not the real composed protocol ρ^π) is polynomial-time. The composition theorem in [HS11] proceeds to show that then, the composed real protocol ρ^π is polynomial-time as well. This dual version of the composition theorem can be easier to work with in situations in which protocol design starts with an ideal protocol, which is then refined and implemented in several steps. The price to pay for this "dual version" of Theorem 20 and Theorem 21 in [HS11] is a modified notion of polynomial runtime, which needs to explicitly consider and restrict the volume of the *message flows* between machines.

Functionalities with code upload. In some situations, it is convenient to model a functionality that accepts a fragment of code from the adversary or simulator and executes that code. For example, consider a signature functionality $\mathcal{F}$. This functionality, given a message m from a user, returns a corresponding signature σ. Since the signature functionality should not depend on the signature scheme that is used to implement the functionality, the functionality does not know what a valid signature σ should look like. Thus, m is typically sent to the simulator which provides a corresponding signature σ. The drawback with this solution is that the simulator learns all messages that are signed, even if both the message and the corresponding signature are never sent over the network. This can be avoided by modeling $\mathcal{F}$ as follows: In the beginning of the execution, the simulator sends a program P to the functionality. When the functionality has to sign m, it simply produces the signature as $\sigma := P(m)$. (This approach was suggested, e.g., in [Can05a].)

The problem with this approach is that such a functionality $\mathcal{F}$ with code upload is not polynomial-time (not even reactively polynomial-time): Since $\mathcal{F}$ does not impose any

runtime bound on P, the execution can take arbitrary long. (Notice also that fixing a polynomial runtime bound is non-trivial because $\mathcal{F}$ cannot know which is the polynomial that should be used.)

In our setting, however, it turns out that functionalities with code upload can be modeled and used. For example, in the case of the signature functionality $\mathcal{F}$, we can specify a protocol π that emulates $\mathcal{F}$ using a fixed signature scheme $\mathfrak{S}$ (π just signs all messages locally). Notice that π is reactively polynomial-time because it does not use an adversary-provided algorithm for signing. $\mathcal{F}$ however is not reactively polynomial-time (but, together with the concrete simulator that we use to simulate ρ, it becomes reactively polynomial-time because that simulator uploads the signature scheme $\mathfrak{S}$). Now, assume a protocol σ that uses the signature functionality $\mathcal{F}$ to implement some functionality $\mathcal{G}$. (Here $\mathcal{G}$ describes the final goal of our protocol construction.) At the first glance, it may seem as though showing that $\sigma^{\mathcal{F}}$ emulates $\mathcal{G}$ will be impossible: $\sigma^{\mathcal{F}}$ contains $\mathcal{F}$ which is not reactively polynomial-time, and hence we might expect $\sigma^{\mathcal{F}}$ to run a superpolynomial number of steps which means that we cannot rely on any computational assumptions in the analysis of $\sigma^{\mathcal{F}}$. This argument, however, is fallacious. Indeed, in the definition of UC emulation (Definition 11) we quantify only over valid adversaries. Hence we have the guarantee that $\sigma^{\mathcal{F}}$ will never make more than a polynomial number of steps. Thus, we can use computational assumptions.

We have that π emulates $\mathcal{F}$. Furthermore, σ^{π} and π will be reactively polynomial-time (if constructed sensibly) since they do not contain $\mathcal{F}$. We can therefore apply the composition theorem to get that σ^{π} emulates $\sigma^{\mathcal{F}}$. Together with the fact that $\sigma^{\mathcal{F}}$ emulates $\mathcal{G}$[26] and the transitivity of UC emulation (Lemma 13), it follows that σ^{π} emulates $\mathcal{G}$. We have thus successfully used a functionality with code upload in the analysis of σ^{π}, and never needed the fact that that functionality was reactively polynomial-time.

The reason why functionalities with code upload can be handled in our setting is that in the definition of UC emulation, we consider only valid adversaries and simulators, i.e., adversaries and simulators that make sure that the real/ideal protocol runs in polynomial-time. This puts the burden of ensuring the correct runtime on the adversary/simulator. Hence a functionality that executes code sent from the adversary/simulator is permitted because the adversary/simulator will not be allowed to send code that runs too long (otherwise the adversary/simulator would not be valid).

Proof of the simple composition theorem. We now finally state the proof of the simple composition theorem:

Proof of Theorem 20. Let $\mathcal{A}$ be the dummy adversary. Since π is reactively polynomial-time, $\mathcal{A}$ is a valid adversary for π. Therefore there exists a simulator $\mathcal{S}$ that is valid for ϕ such that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable.

[26] Depending on the particular situation, proving that $\sigma^{\mathcal{F}}$ emulates $\mathcal{G}$ may be more or less complicated. The fact that $\mathcal{F}$ is not reactively polynomial-time does not exclude the use of complexity assumptions in the proof because a valid adversary will make $\mathcal{F}$ run in polynomial time. But we are restricted in what structural properties of UC we can use when proving that $\sigma^{\mathcal{F}}$ emulates $\mathcal{G}$: both the composition theorem (Theorem 21) and the completeness of the dummy adversary (Theorem 19) do not apply and cannot be used in this subproof. Whether or not these difficulties outweigh the advantages of code upload probably depends on the particular use case.

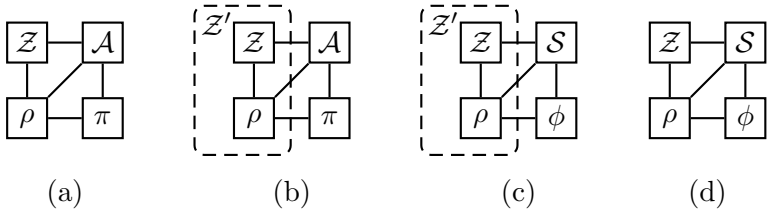

Figure 3: Networks appearing in the proof of the simple composition theorem

To show the composition theorem, by Theorem 19 it is sufficient to show that $\mathcal{S}$ is valid for ρ^π and that for any *a priori* polynomial-time environment $\mathcal{Z}$

$$\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \qquad \text{and} \qquad \mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}} \tag{5}$$

are computationally indistinguishable. These networks are depicted in Figures 3 (a) and (d).

Let therefore $\mathcal{Z}$ be an arbitrary *a priori* polynomial-time environment.

In the classical UC definitions, the proof would now continue by replacing $\mathcal{Z}$ and ρ by a machine $\mathcal{Z}'$ simulating these machines (Figure 3 (b)). Then $\mathcal{Z}'$ could be considered as an environment for π, and $\mathcal{A}$ would be an adversary for π. Since π emulates ϕ we could then replace π and $\mathcal{A}$ by ϕ and $\mathcal{S}$ (Figure 3 (c)) and finally replace $\mathcal{Z}'$ by $\mathcal{Z}$ and ρ (Figure 3 (d)). However, in our setting we have to be more careful. First, an adversary that is valid for ρ^π is not necessarily valid for π. Second, the resulting environment $\mathcal{Z}'$ is not necessarily *a priori* polynomial-time. And third, we further have to show that the simulator $\mathcal{S}$ is valid for ρ^π and not only for π.

The first point can be easily handled since we assumed $\mathcal{A}$ to be the dummy adversary. In this case, $\mathcal{A}$ is also valid for π so the problem does not occur. Note however that if $\mathcal{A}$ was an arbitrary adversary, this would not hold. Therefore the completeness of the dummy adversary is essential for our proof.

The second point can be solved by first replacing ρ by an *a priori* polynomial-time protocol with a sufficiently large polynomial runtime bound p and only then constructing an *a priori* polynomial-time environment $\mathcal{Z}'$ that simulates $\mathcal{Z}$ and the modified ρ. This will be shown in more detail in the following.

The third point is handled at the end of this proof, see below.

Since ρ^π is reactively polynomial-time, so is $\rho^\pi \cup \{\mathcal{A}\}$ (by Lemma 17). Hence for any *a priori* polynomial-time environment $\mathcal{Z}$ the network $\rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ is polynomial-time with overwhelming probability. In other words, there is a polynomial p such that $\mathrm{TIME}_{\rho^\pi,\mathcal{A},\mathcal{Z}}(k, z) \leq p(k)$ with overwhelming probability for all $z \in \{0,1\}^*$ and $k \in \mathbb{N}$.

We now construct the environment $\mathcal{Z}'$ as follows: $\mathcal{Z}'$ simulates the environment $\mathcal{Z}$ and all machines in ρ. However, when the total running time of all machines in ρ exceeds $p(k)$, then $\mathcal{Z}'$ terminates with a special output `beep` (we assume that $\mathcal{Z}$ never outputs `beep`). Since $\mathrm{TIME}_{\rho^\pi,\mathcal{A},\mathcal{Z}}(k, z) > p(k)$ only with negligible probability, the running time of ρ will exceed $p(k)$ only with negligible probability. Thus $\mathcal{Z}'$ terminates with output `beep` only with negligible probability (when running with π and $\mathcal{A}$, cf. Figure 3 (b)) and performs

a faithful simulation of $\mathcal{Z}$ and ρ otherwise. Therefore $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}'}$ are computationally indistinguishable (in fact even statistically indistinguishable).

Since $\mathcal{Z}$ is *a priori* polynomial-time, and since $\mathcal{Z}'$ enforces a polynomial runtime bound for the simulated machines in ρ, the resulting environment $\mathcal{Z}'$ is *a priori* polynomial-time, too.

Therefore by definition of $\mathcal{S}$, the simulator $\mathcal{S}$ is valid for ϕ, and the ensembles $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}'}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}'}$ are computationally indistinguishable. (Cf. Figures 3 (b) and (c).)

Since in an execution $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}'}$ the output `beep` occurs only with negligible probability, the probability of output `beep` is also negligible for the computationally indistinguishable $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}'}$. Since $\mathcal{Z}'$ faithfully simulates $\mathcal{Z}$ and ρ unless it gives output `beep`, we can again replace $\mathcal{Z}'$ by $\mathcal{Z}$ and ρ, resulting in the network $\rho^\pi \cup \{\mathcal{S}, \mathcal{Z}\}$ (cf. Figure 3 (d)). Thus the ensembles $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}'}$ and $\mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable (in fact even statistically indistinguishable).

Summarising, we have

$$\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}'} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}'} \approx \mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}}$$

where $\approx$ denotes computational indistinguishability. Thus $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable and (5) is shown.

It is left to show that $\mathcal{S}$ is valid for ρ^ϕ. Since $\mathcal{S}$ is by construction valid for ϕ, and since $\mathcal{Z}'$ is *a priori* polynomial-time, we have that $\phi \cup \{\mathcal{S}, \mathcal{Z}'\}$ is polynomial-time with overwhelming probability.

As seen above, the output $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}'}$ is `beep` only with negligible probability, and $\mathcal{Z}'$ faithfully simulates ρ and $\mathcal{Z}$ otherwise. Therefore, since the running time of $\phi \cup \{\mathcal{S}, \mathcal{Z}'\}$ is polynomial-time with overwhelming probability, so is that of the network $\rho^\phi \cup \{\mathcal{S}, \mathcal{Z}\}$ which results from replacing $\mathcal{Z}'$ by ρ and $\mathcal{Z}$.

Since this holds for every *a priori* polynomial-time $\mathcal{Z}$, it follows that $\rho^\phi \cup \{\mathcal{S}\}$ is reactively polynomial-time, so the simulator $\mathcal{S}$ is valid for ρ^ϕ. □

We now state our main result in this section, the universal composition theorem:

Theorem 21 (Universal Composition Theorem) *Let π, ϕ and ρ be protocols, such that π and ρ^π are reactively polynomial-time. The protocol ρ may call an arbitrary number of subprotocol instances. Assume that π emulates ϕ. Then ρ^π emulates ρ^ϕ.*

Proof sketch (of Theorem 21). Recall the original proof of the universal composition theorem reproduced in Section 2.1. In that proof, we have constructed a simulator $\mathcal{S}^\infty$ for ρ^ϕ from a simulator $\mathcal{S}$ for ϕ. Concretely, $\mathcal{S}^\infty$ was essentially a combination of many instances of $\mathcal{S}$. It is easy to see that $\mathcal{S}^\infty$ is *a priori* polynomial-time whenever $\mathcal{S}$ is. However, we do not know that $\mathcal{S}^\infty$ is *reactively polynomial-time* (when combined with the ideal protocol) whenever $\mathcal{S}$ is. (Recall that the combination of several reactively polynomial-time machines may not be reactively polynomial.)

Hence, we cannot apply the original reasoning of the universal composition theorem because we do not know if the constructed simulator $\mathcal{S}^\infty$ satisfies our polynomial-time

notion. Furthermore, the hybrid networks H_l from the analysis in Section 2.1 may or may not satisfy any polynomial runtime bounds (which is a prerequisite for applying the theorem assumption that π emulates ϕ). For example, it is possible to construct protocols π and ϕ such that k copies of π running concurrently as well as k copies of ϕ are reactively polynomial-time, but $\frac{k}{2}$ copies of π with $\frac{k}{2}$ copies of ϕ run in exponential time, *even though they cannot communicate directly.*[27] So even when we require both ρ^π and ρ^ϕ to be reactively polynomial-time, the hybrid network $H_{p/2}$ might not be.

We approach these issues by *inductively* proving that the networks H_j $(j = 1, \ldots, p)$ are reactively polynomial-time. Of course, since we apply an inductive step a polynomial number of times, we have to keep track of the concrete complexities and probabilities carefully. To prevent these concrete bounds from growing too quickly, we use the following approach.

Recall that the hybrid environment $\tilde{\mathcal{Z}}_l^*$ from the proof sketch of Theorem 6 mapped subprotocol invocations directly to instances of π, resp. ϕ (with the corresponding adversaries). Concretely, the first $l-1$ subprotocol instances are mapped to ϕ-instances, the l-th subprotocol instance is the challenge instance, and the remaining subprotocol instances are mapped to π-instances. (See also Figure 1(c).) For our purposes, we modify the $\tilde{\mathcal{Z}}_l^*$ into an environment $\mathcal{Z}_l^*$ as follows: Instead of directly mapping the subprotocol sessions invoked by ρ to instances of the real, resp. ideal protocol, our hybrid environment $\mathcal{Z}_l^*$ applies a random permutation to the instance indices $1, \ldots, l$. (In other words, $\mathcal{Z}_l^*$ proceeds like $\tilde{\mathcal{Z}}_l^*$, but randomly shuffles the subprotocol indices.) Assume that for some i we already know that $\mathcal{Z}_i^*$ with π runs in polynomial time with some overwhelming probability $1 - t_{i-1}$ (where t_{i-1} is some negligible function that will be inductively derived below). If we replace π by ϕ, by assumption the environment $\mathcal{Z}_i^*$ cannot distinguish the two cases, so in particular, we know that all $i-1$ internal instances of ϕ simulated by $\mathcal{Z}_i^*$ still run in polynomial time with probability $1 - t_{i-1}$ (up to a negligible error h). Now consider the probability t_i that one of the i internal or external instances of ϕ runs in superpolynomial time. Since the instances $1, \ldots, i$ of ϕ are randomly permuted, the instances of ϕ cannot "know" which of them is the external instance, so with probability $\frac{i-1}{i} t_i$ one of the *internal* instances will run in superpolynomial time, thus $t_i \leq \frac{i}{i-1} t_{i-1}$. Since $\prod_i \frac{i}{i-1}$ is polynomially bounded even for a polynomial number of factors, the probabilities t_i that the hybrid networks H_i run in superpolynomial time will stay negligible. This proves that all hybrid networks H_l are reactively polynomial-time.

Note that in this argument, to derive the runtime bounds of the hybrid networks H_l, we needed that two consecutive H_l are indistinguishable; and to show that indistinguishability, we need the polynomial runtime bound. Fortunately, for the indistinguishability of H_l and H_{l+1}, we need runtime bounds on H_l but not on H_{l+1}. Hence, we can de-

[27] As a rough sketch, assume that there are two puzzles A and B of variable hardness. When $\mathcal{Z}$ solves a puzzle of type A of hardness s for π, then π solves a puzzle of type B of hardness $2s$ for $\mathcal{Z}$. Similarly when $\mathcal{Z}$ solves puzzles of type B for ϕ of hardness s, then ϕ solves puzzles of type A and hardness $2s$ for $\mathcal{Z}$. Both π and ϕ are reactively polynomial-time, even when executed polynomially many times. But when $\mathcal{Z}$ relays the messages between k instances of π and ϕ, these instances will solve puzzles up to a hardness 2^k. Of course, these protocols can be easily distinguished by $\mathcal{Z}$; hence this particular example does not invalidate the proof of the composition theorem.

rive both the indistinguishability and the runtime bounds in one simultaneous induction. Of course, in the full proof we additionally have to keep track of the concrete runtime polynomials, and we have to ensure that the negligible error h is independent of i.

We remark that in the full proof, the hybrid network H_l is not constructed explicitly; instead, we directly analyze the networks $\pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}$ and $\phi \cup \{\mathcal{S}, \mathcal{Z}_{l-1}^*\}$ which simulate the machines in H_l.

The full proof. The rest of this section will be devoted to the proof of Theorem 21. In this, as usual, $k \in \mathbb{N}$ will always denote the security parameter, and $\mathcal{A}$ will always denote the dummy adversary. Furthermore, $\mathcal{S}$ will always denote a simulator such that $\phi \cup \{\mathcal{S}\}$ is reactively polynomial-time, and for every *a priori* polynomial-time $\mathcal{Z}$, we have

$$\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \tag{6}$$

The existence of such a good simulator[28] for ϕ and $\mathcal{A}$ follows from the fact that with π, also $\pi \cup \{\mathcal{A}\}$ is reactively polynomial-time (Lemma 17), and hence our security assumption that π emulates ϕ implies the existence of such an $\mathcal{S}$.

In analogy to existing composability proofs, a good simulator $\mathcal{S}^\infty$ for ρ^ϕ and $\mathcal{A}$ can be obtained by simply running many copies of the simulator $\mathcal{S}$ concurrently, one for each session of ϕ. The main difficulty in proving that $\mathcal{S}^\infty$ is good is to show that the network $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is reactively polynomial-time. This is also the main difference to existing proofs for (universal) composition theorems.

We start by defining a hybrid environment for our hybrid argument. This hybrid argument is, due to the absence of *a priori* and uniform runtime bounds, considerably more complicated than existing hybrid arguments for composition theorems in classical models (such as [Can01, Can05a]).

Definition 22 (Hybrid environment $\mathcal{Z}_{l,p}^*$) *Let π, ϕ, and ρ protocols, such that π and ρ^π are reactively polynomial-time. Let $\mathcal{A}$ be the dummy adversary and $\mathcal{S}$ be a simulator that is valid for ϕ such that* $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ *for all* a priori *polynomial-time $\mathcal{Z}$. Let $\mathcal{Z}$ be an* a priori *polynomial-time environment.*

Let furthermore $p = p(k)$ be a polynomial and $l \in \mathbb{N} \cup \{\infty\}$.

Then the environment $\mathcal{Z}_{l,p}^*$ (to be run either with π or with ϕ) proceeds as follows:

1. *Uniformly pick a random permutation Π on $\{1, \ldots, p(k)\}$. Define $\Pi(i) := i$ for $i > p(k)$.*
2. *Start a simulation of $\mathcal{Z}$ with protocol ρ and adversary $\mathcal{A}$. Note that ρ and $\mathcal{A}$ may invoke and communicate with subprotocol instances of π or ϕ. Denote the session-id of the i-th invoked instance by sid_i.*
3. *Calls to the i-th instance of π are answered as follows:*
   (a) *if $\Pi(i) < l$, then relay to a simulation of protocol ϕ with simulator $\mathcal{S}$,*

[28] By *good simulator $\mathcal{S}$ for ϕ and $\mathcal{A}$* we mean here and in the following that $\phi \cup \{\mathcal{S}\}$ is reactively polynomial-time and that $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ for every *a priori* polynomial-time $\mathcal{Z}$

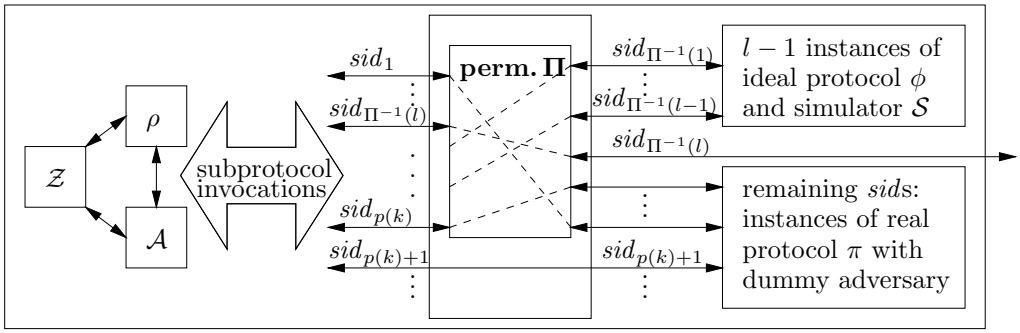

Figure 4: The hybrid environment $\mathcal{Z}^*_{l,p}$ internally simulates environment $\mathcal{Z}$ with dummy adversary $\mathcal{A}$ and protocol ρ. The subroutine calls of ρ (and $\mathcal{A}$) are translated as follows: $l-1$ subsessions are simulated inside $\mathcal{Z}^*_{l,p}$ as ideal instances of ϕ with simulator $\mathcal{S}$. The subsession with session-id $sid_{out} = sid_{\Pi^{-1}(l)}$ is relayed outside of $\mathcal{Z}^*_{l,p}$, i.e., to the adversary and protocol $\mathcal{Z}^*_{l,p}$ itself is running with. The remaining subsessions are simulated in $\mathcal{Z}^*_{l,p}$ as real instances of π together with the dummy adversary. Which subsessions are relayed where is governed by the permutation Π.

(b) *if* $\Pi(i) = l$*, then relay to outside of* $\mathcal{Z}^*_{l,p}$*, i.e., to the protocol and adversary that* $\mathcal{Z}^*_{l,p}$ *runs with,*

(c) *if* $\Pi(i) > l$*, then relay to a simulation of protocol* π *with dummy adversary.*

During this, the session-id sid_i *is removed from and added to the messages as necessary for interfacing to and from* ρ *and* $\mathcal{A}$*.*

4. *When* $\mathcal{Z}$ *terminates, terminate with the same output as* $\mathcal{Z}$*.*

It will be useful to abbreviate $out := \Pi^{-1}(l)$*, i.e., out is the index such that messages for session* sid_{out} *are relayed to the outside of* $\mathcal{Z}^*_{l,p}$*.*

Definition 23 (Hybrid environments $\mathcal{Z}^*_{R,p}, [\mathcal{Z}^*_{l,p}]_q, [\mathcal{Z}^*_{R,p}]_q$) *In the situation of the above Definition 22, and for a polynomial* $q = q(k)$*, define environments* $\mathcal{Z}^*_{R,p}$*,* $[\mathcal{Z}^*_{l,p}]_q$*, and* $[\mathcal{Z}^*_{R,p}]_q$ *just like* $\mathcal{Z}^*_{l,p}$*, only with the following exceptions:*

- $\mathcal{Z}^*_{R,p}$ *initially uniformly chooses* $l \in \{1, \ldots, p(k)\}$ *on its own,*
- $[\mathcal{Z}^*_{l,p}]_q$ *terminates with output* $(timeout, l)$ *as soon as one of the following holds:*
  - *the internally simulated protocol* ρ *runs more than* $p(k)$ *steps, or*
  - *the internally simulated protocol* ρ *or the simulation of* $\mathcal{Z}$ *invokes more than* $p(k)$ *subprotocol sessions,*[29] *or*
  - *one internally simulated subprotocol session (where we count steps of the respective instances of* $\mathcal{S}$*,* π*, and* ϕ*, but* not *those of* $\mathcal{A}$*) runs more than* $q(k)$ *steps.*

[29] $\mathcal{Z}$ may invoke subprotocol sessions through the dummy adversary.

Without losing on generality, we assume that $\mathcal{Z}$ never outputs $(timeout, *)$ on its own (this can be enforced, e.g., by a different encoding of $\mathcal{Z}$'s own output). Hence, from a $(timeout, l)$ output of $[\mathcal{Z}^*_{l,p}]_q$, we can deduce that one of the preceding conditions is fulfilled.

- *$[\mathcal{Z}^*_{R,p}]_q$ is defined as $[\mathcal{Z}^*_{l,p}]_q$, but initially uniformly chooses $l \in \{1, \ldots, p(k)\}$ on its own.*

Note that the environments $[\mathcal{Z}^*_{l,p}]_q$ and $[\mathcal{Z}^*_{R,p}]_q$ stop execution as soon as one of the internally simulated non-$\mathcal{A}$ machines run more than a polynomial number of steps (or if more than polynomially many of those internal simulations are started). By construction of the dummy adversary $\mathcal{A}$, this makes $[\mathcal{Z}^*_{l,p}]_q$ and $[\mathcal{Z}^*_{R,p}]_q$ *a priori* polynomial-time, whereas $\mathcal{Z}^*_{l,p}$ and $\mathcal{Z}^*_{R,p}$ might not be.

The next definition will be useful in the analysis of the environments defined above. It defines events that are fulfilled when certain complexity bounds are surpassed.

Definition 24 (Events $B^i_q, B^\rho_p, B_{p,q}, B^{\neq i}_{p,q}$) *Assume a network of the form $\rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$. For $i \in \mathbb{N}$, denote by B^i_q the event that the machines associated with the i-th session-id sid_i of π run more than $q(k)$ overall steps. Denote by B^ρ_p the event that either the machines from protocol ρ (not counting machines from π) run more than $p(k)$ overall steps, or that ρ and $\mathcal{Z}$ have invoked in total more than $p(k)$ sessions of π.*

Furthermore, let

$$B_{p,q} := B^\rho_p \vee \bigvee_{i \in \mathbb{N}} B^i_q$$
$$B^{\neq i}_{p,q} := B^\rho_p \vee \bigvee_{i' \neq i} B^{i'}_q.$$

For networks of the form $\pi \cup \{\mathcal{A}, \mathcal{Z}^*\}$ with $\mathcal{Z}^* = \mathcal{Z}^*_{l,p}$ or one of its variants, define B^i_q, B^ρ_p, $B_{p,q}$, and $B^{\neq i}_{p,q}$ analogously. As usual, the machines associated with session-id sid_i include a possible copy of $\mathcal{S}$, but not *a possible copy of $\mathcal{A}$.*[30]

We write "B^i_q in N" etc. to emphasize the specific network N in which the event is considered (e.g., "B^ρ_p in $\pi \cup \{\mathcal{A}, \mathcal{Z}^*_{R,p}\}$").

Note that we have defined $B^{\neq out}_{p,q}$ such that $[\mathcal{Z}^*_{l,p}]_q$ gives output $(timeout, l)$ if and only if the event $B^{\neq out}_{p,q}$ occurs.

The following simple observations will prove substantial for the later arguments.

[30] This asymmetry is to ensure that we can compare "timeout events" in systems of the form $\rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ and $\pi \cup \{\mathcal{A}, [\mathcal{Z}^*]\}$ where the dummy adversary relays a different set of connections. Intuitively, this is justified by the fact that the dummy adversary can be considered just as being a set of connections and not participating actively in the computation.

Lemma 25 *In the situation of Definition 22, for arbitrary p, and $l \in \mathbb{N}$, the network equivalences*

$$\phi \cup \{\mathcal{S}, \mathcal{Z}^*_{l,p}\} = \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{l+1,p}\} \tag{7}$$

$$\rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\} = \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\} \tag{8}$$

$$\rho^\phi \cup \{\mathcal{S}, \mathcal{Z}\} = \phi \cup \{\mathcal{S}, \mathcal{Z}^*_{\infty,p}\} = \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{\infty,p}\} \tag{9}$$

hold in the following sense. For each equivalence, the common distribution of the view of all machines (simulated and non-simulated, but excluding instances of the dummy adversary $\mathcal{A}$*)*[31] *on the left-hand-side is identical to common distribution of the view of all machines on the right-hand-side.*

Here we do not *count the view of $\mathcal{Z}^*_{l,p}$ itself, but only the common view all of its submachines, except for $\mathcal{A}$-instances.*

Proof. For Equation 7, this is clear since in both networks, precisely l ideal protocol instances are present, in both cases with the session-ids $(sid_{\Pi^{-1}(1)}, \ldots, sid_{\Pi^{-1}(l)})$.

Similarly, in the networks from Equation 8, only real instances are run, and in Equation 9, only ideal instances are run. (Note that $\mathcal{Z}^*_{\infty,p}$'s execution does not depend on the network it runs in, since $\mathcal{Z}^*_{\infty,p}$ never activates the network it runs with.) □

The following lemma will not only act as a "base case" in the upcoming inductive argument. It will also be useful to derive the existence of some concrete complexity bounds.

Lemma 26 *In the situation of Definition 22, there exist polynomials $p = p(k)$ and $q = q(k)$, and a negligible function $\mu = \mu(k)$ such that for all $k \in \mathbb{N}$ and all auxiliary inputs $z \in \{0,1\}^*$ for $\mathcal{Z}$, the following holds. We have that $\Pr[B_{p,q}] \leq \mu(k)$, both in $\pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}$ and in $\phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}$.*

Proof. By assumption, ρ^π is reactively polynomial-time. So by Lemma 17, also the network $\rho^\pi \cup \{\mathcal{A}\}$ is reactively polynomial-time. Since the original environment $\mathcal{Z}$ is *a priori* polynomial-time, $\rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ is polynomial-time with overwhelming probability. Hence, there is a polynomial $p = p(k)$ and a negligible function $\mu_1 = \mu_1(k)$, such that

$$\Pr[B_{p,p} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}] \overset{(8)}{=} \Pr[B_{p,p} \text{ in } \rho^\pi \cup \{\mathcal{A}, \mathcal{Z}\}] \leq \Pr[\mathrm{TIME}_{\rho^\pi, \mathcal{A}, \mathcal{Z}} > p(k)] \leq \mu_1(k). \tag{10}$$

As discussed above, by construction, $[\mathcal{Z}^*_{1,p}]_p$ is *a priori* polynomially bounded and outputs $(timeout, 1)$ iff $B^{\neq out}_{p,p}$ occurs. Since $\mathcal{S}$ is a good simulator for ϕ, this implies

$$\begin{aligned}\Pr[B^{\neq out}_{p,p} \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}] &\overset{(*)}{=} \Pr[B^{\neq out}_{p,p} \text{ in } \phi \cup \{\mathcal{S}, [\mathcal{Z}^*_{1,p}]_p\}] \\ &\overset{(**)}{\leq} \Pr[B^{\neq out}_{p,p} \text{ in } \pi \cup \{\mathcal{A}, [\mathcal{Z}^*_{1,p}]_p\}] + \mu_2(k) \overset{(*)}{=} \Pr[B^{\neq out}_{p,p} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}] + \mu_2(k) \\ &\overset{(10)}{\leq} \mu_1(k) + \mu_2(k). \end{aligned} \tag{11}$$

[31] See footnote 30.

for some negligible $\mu_2 = \mu_2(k)$. Here $(*)$ uses that $[\mathcal{Z}^*_{1,p}]_p$ behaves like $\mathcal{Z}^*_{1,p}$ until $B^{\neq out}_{p,p}$ occurs. And $(**)$ uses that $B^{\neq out}_{p,p}$ can be efficiently computed from the output of $[\mathcal{Z}^*_{1,p}]_p$.

Now, since $[\mathcal{Z}^*_{1,p}]_p$ is *a priori* polynomial-time, $\phi \cup \{\mathcal{S}, [\mathcal{Z}^*_{1,p}]_p\}$ is polynomial-time with overwhelming probability. Hence, there is a polynomial $q = q(k)$ with $q > p$ and a negligible function $\mu_3 = \mu_3(k)$ with

$$\Pr[B^{out}_q \text{ in } \phi \cup \{\mathcal{S}, [\mathcal{Z}^*_{1,p}]_p\}] \;\leq\; \Pr[\mathrm{TIME}_{\phi,\mathcal{S},[\mathcal{Z}^*_{1,p}]_p} > q(k)] \;\leq\; \mu_3(k). \tag{12}$$

Since $[\mathcal{Z}^*_{1,p}]_p$ simulates $\mathcal{Z}^*_{1,p}$ until it outputs $(\mathit{timeout}, 1)$ which in turn happens with probability at most $\mu_1 + \mu_2$ in an execution with ϕ and $\mathcal{S}$ by (11), an execution of $\phi \cup \{\mathcal{S}, [\mathcal{Z}^*_{1,p}]_p\}$ and an execution of $\phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}$ differ with probability at most $\mu_1 + \mu_2$. Using (12) it follows that

$$\Pr[B^{out}_q \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}] \;\leq\; \mu_1(k) + \mu_2(k) + \mu_3(k). \tag{13}$$

Let $\mu := 2\mu_1 + 2\mu_2 + \mu_3$. Since $q > p$, we have $B_{p,q} = B^\rho_p \vee \bigvee_{i\in\mathbb{N}} B^i_q \Rightarrow B^\rho_p \vee \bigvee_{i\neq out} B^i_p \vee B^{out}_q = B^{\neq out}_{p,p} \vee B^{out}_q$. So

$$\Pr[B_{p,q} \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}] \leq \Pr[B^{\neq out}_{p,p} \vee B^{out}_q \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}] \overset{(11,13)}{\leq} \mu(k). \tag{14}$$

Finally, since $q > p$, we have $B_{p,q} = B^\rho_p \vee \bigvee_{i\in\mathbb{N}} B^i_q \Rightarrow B^\rho_p \vee \bigvee_{i\in\mathbb{N}} B^i_p = B_{p,p}$ and thus get

$$\Pr[B_{p,q} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}] \;\leq\; \Pr[B_{p,p} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}] \overset{(10)}{\leq} \mu_1(k) \leq \mu(k). \tag{15}$$

Equations (15) and (14) show the lemma. □

For the remainder of this section, fix p, q, and μ as given by Lemma 26. For readability, we will drop p and q from the notation of the hybrid environments and events. That is, we will abbreviate $\mathcal{Z}^*_i := \mathcal{Z}^*_{i,p}$, $\mathcal{Z}^*_R := \mathcal{Z}^*_{R,p}$, $[\mathcal{Z}^*_i] := [\mathcal{Z}^*_{i,p}]_q$, and $[\mathcal{Z}^*_R] := [\mathcal{Z}^*_{R,p}]_q$. Also, we will write $B := B_{p,q}$, $B^i := B^i_q$, $B^\rho := B^\rho_{p,q}$, and $B^{\neq i} := B^{\neq i}_q$

Lemma 27 *In the situation of Definition 22, there exists a negligible function $h = h(k)$ such that for all $k \in \mathbb{N}$, all $l \in \{1, \ldots, p(k)\}$, and all auxiliary inputs $z \in \{0,1\}^*$ for $\mathcal{Z}$, we have*

$$\Big|\Pr[B^{\neq out} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}^*_l\}] - \Pr[B^{\neq out} \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}^*_l\}]\Big| \leq h(k). \tag{16}$$

Note the universality of h; in particular it does not depend on l.

Proof. By construction, $[\mathcal{Z}^*_R]$ is *a priori* polynomial-time. Therefore, we have the computational indistinguishability $\mathrm{EXEC}_{\pi,\mathcal{A},[\mathcal{Z}^*_R]}(k,z) \approx \mathrm{EXEC}_{\phi,\mathcal{S},[\mathcal{Z}^*_R]}(k,z)$. Now let

$$\delta_l(k) := \max_{z\in\{0,1\}^*} \Big|\Pr[B^{\neq out} \text{ in } \pi \cup \{\mathcal{A}, [\mathcal{Z}^*_l]\}] - \Pr[B^{\neq out} \text{ in } \phi \cup \{\mathcal{S}, [\mathcal{Z}^*_l]\}]\Big|,$$

and let $l^*(k)$ be an index $l^* \in \{1, \ldots, p(k)\}$ that maximizes $\delta_{l^*}(k)$.[32]

[32] The maximum is reached because $[\mathcal{Z}^*_l]$ is *a priori* polynomial-time and hence considers only a finite prefix of z (the length depending only on the security parameter k). Hence one can assume that there are only finitely many different z for each k.

Let D be the non-uniform polynomial-time algorithm that upon input $(1^k, z, X)$ outputs 1 iff $X = (\mathit{timeout}, l^*(k))$. Since $[\mathcal{Z}_R]$ chooses a random $l \in \{1, \ldots, p(k)\}$ and then behaves like $[\mathcal{Z}_l]$, and thus in particular only outputs $(\mathit{timeout}, l^*(k))$ if $l = l^*(k)$, we have for all $k \in \mathbb{N}$ and $l \in \{1, \ldots, p(k)\}$ that

$$\begin{aligned}
h'(k) &:= \max_{z \in \{0,1\}^*} \Big| \Pr[D(1^k, z, \mathrm{EXEC}_{\pi, \mathcal{A}, [\mathcal{Z}_R^*]}(k, z)) = 1] \\
&\qquad - \Pr[D(1^k, z, \mathrm{EXEC}_{\phi, \mathcal{S}, [\mathcal{Z}_R^*]}(k, z)) = 1] \Big| \\
&= \max_{z \in \{0,1\}^*} \tfrac{1}{p(k)} \Big| \Pr[D(1^k, z, \mathrm{EXEC}_{\pi, \mathcal{A}, [\mathcal{Z}_{l^*(k)}^*]}(k, z)) = 1] \\
&\qquad - \Pr[D(1^k, z, \mathrm{EXEC}_{\phi, \mathcal{S}, [\mathcal{Z}_{l^*(k)}^*]}(k, z)) = 1] \Big| \\
&= \max_{z \in \{0,1\}^*} \tfrac{1}{p(k)} \Big| \Pr[B^{\neq out} \text{ in } \pi \cup \{\mathcal{A}, [\mathcal{Z}_{l^*(k)}^*]\}] - \Pr[B^{\neq out} \text{ in } \phi \cup \{\mathcal{S}, [\mathcal{Z}_{l^*(k)}^*]\}] \Big| \\
&= \tfrac{1}{p(k)} \delta_{l^*(k)}(k) \geq \tfrac{1}{p} \delta_l(k).
\end{aligned}$$

Since $\mathrm{EXEC}_{\pi, \mathcal{A}, [\mathcal{Z}_R^*]} \approx \mathrm{EXEC}_{\phi, \mathcal{S}, [\mathcal{Z}_R^*]}$, and D is non-uniform polynomial-time, we have that h' is negligible.[33] Therefore $h(k) := p(k)h'(k)$ is negligible, too, and $\delta_l(k) \leq h(k)$ for all k.

Now observe that for all l, the environment $[\mathcal{Z}_l^*]$ behaves by construction exactly like $\mathcal{Z}_l^*$ unless $B^{\neq out}$ occurs. The lemma follows. □

Lemma 28 *In the situation of Definition 22, there exists a negligible function ν such that for all $k \in \mathbb{N}$, all $l \in \mathbb{N} \cup \{\infty\}$, and all $z \in \{0,1\}^*$, the following holds. We have $\Pr[B] \leq \nu(k)$, in all of the following networks:*

$$\begin{array}{cccc}
\pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}, & \pi \cup \{\mathcal{A}, \mathcal{Z}_R^*\}, & \pi \cup \{\mathcal{A}, [\mathcal{Z}_l^*]\}, & \pi \cup \{\mathcal{A}, [\mathcal{Z}_R^*]\}, \\
\phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}, & \phi \cup \{\mathcal{S}, \mathcal{Z}_R^*\}, & \phi \cup \{\mathcal{S}, [\mathcal{Z}_l^*]\}, & \phi \cup \{\mathcal{S}, [\mathcal{Z}_R^*]\}.
\end{array}$$

Proof. Fix a security parameter $k \in \mathbb{N}$ and auxiliary input $z \in \{0,1\}^*$. For $l \in \{1, \ldots, p(k)\}$, define $t_l := \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}]$. Our goal will be to give a common negligible bound on all t_l. Now Lemma 26 shows that $t_1 \leq \mu(k)$ where μ is negligible. The bounds on t_l for $l > 1$ will now be derived inductively.

Fix some $l \in \{2, \ldots, p(k)\}$. Recall that in an execution of $\phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}$, the session-ids $(\mathit{sid}_{\Pi^{-1}(1)}, \ldots, \mathit{sid}_{\Pi^{-1}(l)})$ refer to l identical ideal instances of $\phi \cup \{\mathcal{S}\}$. The sessions with the first $l-1$ session-ids in the list are simulated inside $\mathcal{Z}_l^*$. Only the last ideal session in this list, the one with session-id $\mathit{sid}_{out} = \mathit{sid}_{\Pi^{-1}(l)}$, is relayed outside of $\mathcal{Z}_l^*$. By the uniform choice of Π, however, the distribution of this list of session-ids is invariant under any (fixed) permutation. Hence, for runs of $\phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}$, we have for any fixed $j < l$:

$$\Pr[\neg B^{\neq \Pi^{-1}(l)} \wedge B^{\Pi^{-1}(l)}] = \Pr[\neg B^{\neq \Pi^{-1}(j)} \wedge B^{\Pi^{-1}(j)}]. \tag{17}$$

[33] Here we use that we have defined computational indistinguishability with respect to non-uniform distinguishers. In case of uniform distinguishers, the lemma can be shown with a more complicated but uniform D that guesses l^* by sampling runs of $\pi \cup \{\mathcal{A}, [\mathcal{Z}^R]\}$ and approximating δ_l.

Thus,

$$\Pr[B^{\neq\Pi^{-1}(l)}] \;\geq\; \Pr[\exists j \leq l-1 : B^{\Pi^{-1}(j)}] \;\geq\; \Pr[\exists j \leq l-1 : \neg B^{\neq\Pi^{-1}(j)} \wedge B^{\Pi^{-1}(j)}]$$
$$\stackrel{(*)}{=} \sum_{j=1}^{l-1} \Pr[\neg B^{\neq\Pi^{-1}(j)} \wedge B^{\Pi^{-1}(j)}] \stackrel{(17)}{=} (l-1)\Pr[\neg B^{\neq\Pi^{-1}(l)} \wedge B^{\Pi^{-1}(l)}]. \quad (18)$$

Here $(*)$ uses the fact that the events $\neg B^{\neq\Pi^{-1}(j)} \wedge B^{\Pi^{-1}(j)}$ are mutually exclusive for different j. We obtain

$$\Pr[B] \stackrel{(*)}{=} \Pr[B^{\neq\Pi^{-1}(l)}] + \Pr[\neg B^{\neq\Pi^{-1}(l)} \wedge B^{\Pi^{-1}(l)}] \stackrel{(18)}{\leq} \Pr[B^{\neq\Pi^{-1}(l)}] + \tfrac{1}{l-1}\Pr[B^{\neq\Pi^{-1}(l)}]$$
$$= \tfrac{l}{l-1}\Pr[B^{\neq\Pi^{-1}(l)}] \;=\; \tfrac{l}{l-1}\Pr[B^{\neq out}]. \quad (19)$$

Here $(*)$ uses the fact that $B \Leftrightarrow B^{\neq\Pi^{-1}(l)} \vee B^{\Pi^{-1}(l)}$.

Therefore we have

$$t_l = \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}] \stackrel{(19)}{\leq} \tfrac{l}{l-1}\Pr[B^{\neq out} \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}]$$
$$\stackrel{(16)}{\leq} \tfrac{l}{l-1}\big(\Pr[B^{\neq out} \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}] + h(k)\big) \;\leq\; \tfrac{l}{l-1}\big(\Pr[B \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}] + h(k)\big)$$
$$\stackrel{(7)}{=} \tfrac{l}{l-1}\big(\Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_{l-1}^*\}] + h(k)\big) \;=\; \tfrac{l}{l-1}(t_{l-1} + h(k))$$

Hence for any $l \in \{1, \ldots, p(k)\}$ we have

$$t_l \;\leq\; \left(\prod_{\gamma=2}^{l} \frac{\gamma}{\gamma-1}\right) t_1 + \left(\sum_{j=2}^{l}\prod_{\gamma=j}^{l} \frac{\gamma}{\gamma-1}\right) h(k)$$
$$= \; lt_1 + \sum_{j=2}^{l} \tfrac{l}{j-1}\, h(k) \;\leq\; lt_1 + l^2 h(k) \leq p(k)\mu(k) + p(k)^2 h(k) \;=:\; \nu(k). \quad (20)$$

Since p is polynomial, and μ and h are negligible, ν is negligible as well. Note that the construction of ν does not depend on k, l, or z.

For bounding t_l in case $l > p(k)$ (this includes the case $l = \infty$), consider executions of $\mathcal{Z}_l^*$. Now if $l > p(k)$, then $\mathcal{Z}_l^*$ runs the first $p(k)$ subprotocol sessions that ρ asks for internally as ideal instances, independently of the concrete value of l and $\mathcal{Z}_l^*$'s surrounding network. (Note that only the $\Pi^{-1}(l)$-th invoked session gets relayed outside, and that $\Pi^{-1}(l) = l > p(k)$ for $l > p(k)$.) Since the invocation of more than $p(k)$ sessions causes B^ρ and thus B, this implies that for $l > p(k)$,

$$\Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}] \;=\; \Pr[B \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}] \quad (21)$$
$$\Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}] \;=\; \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_{p(k)+1}^*\}]. \quad (22)$$

We get for $l > p(k)$:

$$t_l \;=\; \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}] \stackrel{(22)}{=} \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_{p(k)+1}^*\}]$$
$$\stackrel{(21)}{=} \Pr[B \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_{p(k)+1}^*\}] \stackrel{(7)}{=} \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_{p(k)}^*\}] \;=\; t_{p(k)} \stackrel{(20)}{\leq} \nu(k).$$

Combining this with (20), we see that

$$\forall l \in \mathbb{N} \cup \{\infty\} : \quad \Pr[B \text{ in } \phi \cup \{\mathcal{S}, \mathcal{Z}_l^*\}] \leq \nu(k). \tag{23}$$

With Equation 7 for the case $l > 1$ and Lemma 26 for the case $l = 1$ (using that $\mu \leq \nu$ by construction), we also obtain

$$\forall l \in \mathbb{N} \cup \{\infty\} : \quad \Pr[B \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_l^*\}] \leq \nu(k) \tag{24}$$

for the same ν. The remaining bounds from the lemma statement can be derived from Equation 23 and Equation 24 by using that

- $\mathcal{Z}_l^*$ and $[\mathcal{Z}_l^*]$ proceed identically unless B occurs (since B is implied by $B^{\neq out}$), so $\Pr[B]$ is identical with these environments,
- $\mathcal{Z}_R^*$ first picks $l \in \{1, \ldots, p(k)\}$ and then runs $\mathcal{Z}_l^*$, so any bound on $\Pr[B]$ that holds for *all* $\mathcal{Z}_l^*$ also holds for $\mathcal{Z}_R^*$. □

Lemma 29 *In the situation of Definition 22, we have the computational indistinguishability* $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_1^*}(k,z) \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_\infty^*}(k,z)$.

Proof. First, we have the following chain of computational indistinguishabilities:

$$\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_R^*} \approx \mathrm{EXEC}_{\pi,\mathcal{A},[\mathcal{Z}_R^*]} \approx \mathrm{EXEC}_{\phi,\mathcal{S},[\mathcal{Z}_R^*]} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_R^*}. \tag{25}$$

The first and third indistinguishability hold because $\mathcal{Z}_R^*$ and $[\mathcal{Z}_R^*]$ behave identically unless B occurs, and Lemma 28 bounds $\Pr[B]$ by a negligible function in these networks. The second indistinguishability holds since $\mathcal{S}$ is a good simulator, and $[\mathcal{Z}_R^*]$ is *a priori* polynomial-time.

Thus, for any non-uniform polynomial-time distinguisher D, the following is negligible:

$$\begin{aligned}
&\Big|\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_R^*}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_R^*}(k,z)) = 1]\Big| \\
&= \frac{1}{p(k)}\Big|\sum_{l=1}^{p(k)}\big(\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_l^*}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}_l^*}(k,z)) = 1]\big)\Big| \\
&\overset{(7)}{=} \frac{1}{p(k)}\Big|\sum_{l=1}^{p(k)}\big(\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_l^*}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_{l+1}^*}(k,z)) = 1]\big)\Big| \\
&= \frac{1}{p(k)}\Big|\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_1^*}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_{p(k)+1}^*}(k,z)) = 1]\Big| \\
&\overset{(*)}{\geq} \frac{1}{p(k)}\left(\Big|\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_1^*}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}_\infty^*}(k,z)) = 1]\Big| + \nu(k)\right).
\end{aligned} \tag{26}$$

Here $(*)$ uses Lemma 28 and the fact that $\mathcal{Z}^*_{p(k)+1}$ and $\mathcal{Z}^*_\infty$ behave identically unless B occurs.

Thus $\left|\Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_1}(k,z)) = 1] - \Pr[D(1^k, z, \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_\infty}(k,z)) = 1]\right|$ is negligible and and hence

$$\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_1} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_\infty} \qquad \square$$

We can finally proceed to prove the main result.

Proof of Theorem 21. Recall that $\mathcal{A}$ always denotes the dummy adversary. As in Definition 22 and all the preceding helping lemmas, let $\mathcal{S}$ be a simulator for a single instance of ϕ, such that for all *a priori* polynomial-time $\mathcal{Z}$, we have $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$. Now we construct a good simulator $\mathcal{S}^\infty$ for ρ^ϕ, such that $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is reactively polynomial-time, and such that $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}$ for every *a priori* polynomial-time $\mathcal{Z}$.

This construction of $\mathcal{S}^\infty$ is actually the same as in previous proofs of universal composability (e.g., as in the setting of [Can01]) and conceptually simple: $\mathcal{S}^\infty$ internally simulates a copy of the dummy adversary $\mathcal{A}$ for attacking ρ itself, and as many instances of $\mathcal{S}$ as needed, one for each session that the simulation of $\mathcal{A}$ or the protocol ρ asks for. Messages between $\mathcal{A}$ and instances of π are rerouted to the corresponding instances of $\mathcal{S}$. Messages between the instances of $\mathcal{S}$ and instances of protocol ϕ are directly relayed to $\mathcal{S}^\infty$'s outside, i.e., to the ϕ-hybrid setting in which $\mathcal{S}^\infty$ is executed. Informally, we get the situation depicted in Figure 5 when $\mathcal{S}^\infty$ is run with an environment $\mathcal{Z}$ and protocol ρ^ϕ. Note that the only difference to the hybrid simulator from the proof the composition theorem in the classical UC setting is that $\mathcal{S}^\infty$ has no upper bound on the number of instances of $\mathcal{S}$ it simulates. In particular, $\mathcal{S}^\infty$ is not *a priori* polynomial-time even if $\mathcal{S}$ is.

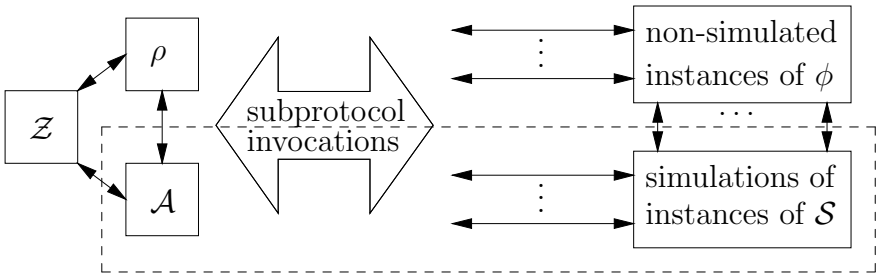

Figure 5: The dashed box surrounds simulator $\mathcal{S}^\infty$, running with environment $\mathcal{Z}$ and protocol ρ^ϕ (i.e., with protocol ρ in the ϕ-hybrid model). $\mathcal{S}$ internally simulates the dummy adversary $\mathcal{A}$ and instances of simulator $\mathcal{S}$.

Now we make the following claim of execution equalities: for all environments $\mathcal{Z}$, auxiliary inputs z and security parameters k, we claim

$$\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}}(k,z) = \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_1}(k,z) \tag{27}$$

$$\mathrm{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}(k,z) = \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}^*_\infty}(k,z). \tag{28}$$

Equation 27 follows from Equation 8. For Equation 28, note that the permutation Π in the definition of $\mathcal{Z}_l^*$ dictates which subsession instance queries are relayed where, but since all subsessions in $\phi \cup \{\mathcal{S}, \mathcal{Z}_\infty^*\}$ are ideal instances, this does not have any impact. (This has already been exploited in the proof of Equation 9.) Note also that $\mathcal{Z}_\infty^*$ never invokes the external machines $\mathcal{A}$ and $\mathcal{Z}_\infty^*$, but relays all session-ids to the unbounded number of internal instances of ρ and $\mathcal{S}$.

Combining Equation 27 and Equation 28 with Lemma 29 shows the indistinguishability $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}$.

It remains to show that $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is reactively polynomial-time (and thus $\mathcal{S}^\infty$ is valid for ρ^ϕ). Fix any *a priori* polynomial-time $\mathcal{Z}$ to run with $\rho^\phi \cup \{\mathcal{S}^\infty\}$. The above argument for Equation 28 shows that

$$\Pr[B \text{ in } \rho^\phi \cup \{\mathcal{S}^\infty, \mathcal{Z}\}] = \Pr[B \text{ in } \pi \cup \{\mathcal{A}, \mathcal{Z}_\infty^*\}].$$

Now the right hand side of this equation is negligible by Lemma 28. Hence $\Pr[B]$ is negligible in $\rho^\phi \cup \{\mathcal{S}^\infty, \mathcal{Z}\}$. Since in this network, event B occurs already if *any* machine exceeds a certain fixed polynomial runtime bound (or if more than a fixed polynomial number of machines are invoked), $\rho^\phi \cup \{\mathcal{S}^\infty, \mathcal{Z}\}$ is polynomial-time with overwhelming probability. Hence $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is reactively polynomial-time. □

8 Example: Secure Message Transmission

In this section we will use a toy example to show how using UC with respect to reactive polynomial time differs from using classical UC. In particular, we will demonstrate that for using our notion, one does not have to perform more complicated checks whether a protocol is polynomial time than one would have to do using the classical UC notion anyway. For this, we will consider an implementation of the functionality $\mathcal{F}_{\mathrm{SMT}}$ for secure message transmission. The functionality $\mathcal{F}_{\mathrm{SMT}}$ is defined as follows:

Functionality $\mathcal{F}_{\mathrm{SMT}}$

The functionality $\mathcal{F}_{\mathrm{SMT}}$ proceeds as follows:

- When receiving an input $(\texttt{Send}, m)$ from party P_1, then send $(\texttt{Sent}, |m|)$ to the adversary, and send a delayed message $(\texttt{Sent}, m)$ to P_2.[34]

Note that this functionality does not impose any bounds on the number or length of the transmitted messages. Yet it is easy to see that it is reactively polynomial-time, because the running time of $\mathcal{F}_{\mathrm{SMT}}$ is linear in the length of the inputs from the environment and the simulator. We will realise $\mathcal{F}_{\mathrm{SMT}}$ in the authenticated channel model in the case of static corruption and make use of an ideal key exchange functionality $\mathcal{F}_{\mathrm{KE}}$. The functionality $\mathcal{F}_{\mathrm{KE}}$ is defined as follows:

[34] By delayed we mean that the adversary may schedule the delivery of that message. That is, the functionality queues the message and only sends it upon an explicit request from the adversary. See [Can05a] for details.

Functionality $\mathcal{F}_{\mathrm{KE}}$

The functionality $\mathcal{F}_{\mathrm{KE}}$ proceeds as follows (on security parameter k):

- When receiving an input $(\mathtt{Key})$ from party P_1, then choose a random key $K \in \{0,1\}^k$, send $(\mathtt{Key})$ to the adversary, and send $(\mathtt{Key}, K)$ as delayed messages to P_1 and P_2.

Let (E, D) be an IND-CPA secure encryption scheme (we assume for simplicity that the keys for (E, D) are uniformly distributed keys of length k). Note, that this encryption scheme is not *a priori* polynomial-time, but polynomial-time in its input. Next, we implement $\mathcal{F}_{\mathrm{SMT}}$ using the following (unsurprising) protocol.

Protocol SMT

- Whenever P_1 receives $(\mathtt{Send}, m)$ from the environment, it invokes a new instance of $\mathcal{F}_{\mathrm{KE}}$. Let K be the key that is sent to P_1 and P_2 by $\mathcal{F}_{\mathrm{KE}}$.
- Then P_1 sends $c := E_K(m)$ to P_2 over an authenticated channel.[35]
- Upon receipt of a message c from P_1, P_2 computes $m := D_K(c)$ and sends $(\mathtt{Sent}, m)$ to the environment.

For simplicity, we only elaborate on the case that no party is corrupted.[36] First, we verify that SMT is indeed reactively polynomial-time. For each input $(\mathtt{Send}, m)$ from the environment, one instance of the functionality $\mathcal{F}_{\mathrm{KE}}$ is invoked, and one encryption and one decryption is performed, whose complexity is polynomial in the length of m. So the total complexity of SMT is polynomial in the total length of all messages m received from the environment, so SMT is reactively polynomial-time.

We now examine whether SMT emulates $\mathcal{F}_{\mathrm{SMT}}$. By Theorem 19, it is sufficient to give a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$. The simulator $\mathcal{S}$ for the protocol SMT is straightforward: Whenever the simulator receives $(\mathtt{Sent}, l)$ from $\mathcal{F}_{\mathrm{SMT}}$, it informs the environment that an instance of $\mathcal{F}_{\mathrm{KE}}$ has been invoked. When the environment tells $\mathcal{S}$ to deliver the key to P_1, the simulator chooses an arbitrary message $\tilde{m}$ of length l and a random key K and informs the environment that the message $E_K(\tilde{m})$ has been transmitted over the authenticated channel.

To show that SMT emulates $\mathcal{F}_{\mathrm{SMT}}$, we show that $\mathrm{EXEC}_{\mathsf{SMT},\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\mathcal{F}_{\mathrm{SMT}},\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable for any *a priori* polynomial-time environment $\mathcal{Z}$, and that $\mathcal{S}$ is a valid simulator for ϕ. The computational indistinguishability follows from the fact that (E, D) is IND-CPA and therefore the environment cannot distinguish between $E_K(\tilde{m})$ and $E_K(m)$. We will not go into details, since this part of the proof is standard and does not differ from the analogous proof in the classical UC setting. To see that $\mathcal{S}$ is valid, we have to see that $\{\mathcal{F}_{\mathrm{SMT}}, \mathcal{S}\}$ is reactively polynomial-time. For each message m that is sent, the machines in $\{\mathcal{F}_{\mathrm{SMT}}, \mathcal{S}\}$ will only send messages that are polynomial-time in the length of m (most notably the encryption $E_K(\tilde{m})$). Since computing these messages also takes only polynomial time in $|m|$, the overall complexity

[35] We assume an authenticated channel where the adversary can reorder and drop, but not replay messages.

[36] For secure message transmission, this is actually the interesting case.

of $\{\mathcal{F}_{\mathrm{SMT}}, \mathcal{S}\}$ is polynomially bounded in the total length of the messages m. Thus $\mathcal{S}$ is valid. Note that interestingly, the simulator $\mathcal{S}$ by itself is not reactively polynomial-time. When receiving $(\mathtt{Sent}, l)$ he chooses a random message $\tilde{m}$ of length l, and the integer l is exponential in the length $|l|$ of its representation. However, the fact that $\mathcal{F}_{\mathrm{SMT}}$ would never send $(\mathtt{Sent}, l)$ without receiving a message of length l guarantees that the overall network is reactively polynomial-time. This, too, shows the flexibility of our approach; many earlier models of polynomial time in the UC setting would require $\mathcal{F}_{\mathrm{SMT}}$ to send $(\mathtt{Sent}, 1^\ell)$ instead of $(\mathtt{Sent}, \ell)$ to ensure that the running time of the simulator is bounded in the length of its input 1^ℓ.

We have seen that SMT emulates $\mathcal{F}_{\mathrm{SMT}}$ in the $\mathcal{F}_{\mathrm{KE}}$-hybrid model. Assume now that we want to implement $\mathcal{F}_{\mathrm{SMT}}$ without using an ideal key exchange. Let therefore DH be a Diffie-Hellman key exchange. Under the decisional Diffie-Hellman assumption, it is not hard to see that DH emulates $\mathcal{F}_{\mathrm{KE}}$ (in the case of static corruption at least). To see that $\mathsf{SMT}^{\mathsf{DH}}$ (i.e., the protocol SMT using DH as subprotocol) emulates $\mathcal{F}_{\mathrm{SMT}}$, we have to apply the Universal Composition Theorem 21. The protocol DH is *a priori* polynomial-time (since it generate only a *single* key of fixed length), so in particular it is reactively polynomial-time. Furthermore, we have to see that $\mathsf{SMT}^{\mathsf{DH}}$ is reactively polynomial-time. Analogous to the above, we count the number of steps occurring when a message m is transmitted and see that the complexity of $\mathsf{SMT}^{\mathsf{DH}}$ is polynomial-time in the total length of the messages transmitted. So $\mathsf{SMT}^{\mathsf{DH}}$ is reactively polynomial-time, too. Therefore Theorem 21 applies, and $\mathsf{SMT}^{\mathsf{DH}}$ emulates $\mathcal{F}_{\mathrm{SMT}}$.

9 Variants of our approach

In this section, we present two variants of our notion of polynomial time and of the corresponding security notion. The goal is to give the reader the possibility to better understand which of our design choices are necessary and which are just a matter of taste.

In Section 9.1, we introduce a simplification of the definition of reactive polynomial time, strong reactive polynomial time. Strong reactive polynomial time requires that the overall system (including $\mathcal{Z}$) runs in polynomial time with probability 1 (instead of just overwhelming probability as in Definition 10). We show that this variant is not viable because the composition theorem does not hold.

In Section 9.2, we introduce the notion of uniform reactive polynomial time. Recall that in Definition 10, we required that for any reactively polynomial-time system S and any *a priori* polynomial-time ITM $\mathcal{Z}$, the complexity of $S \cup \{\mathcal{Z}\}$ is polynomial-time with overwhelming probability. However, no requirement was made as to how the polynomial bounding the running time of $S \cup \{\mathcal{Z}\}$ depends on the polynomial bounding the running time of $\mathcal{Z}$. In contrast, in the case of uniform reactive polynomial time we require that these two polynomials are polynomially related. We show that the choice between reactive polynomial time in the sense of Definition 10 and uniform reactive polynomial time is largely a matter of choice and that all our results also apply to uniform polynomial time.

9.1 Strong reactive polynomial time

In Section 4 we have introduced the notion of a reactively polynomial-time network S roughly as follows: For any ITM $\mathcal{Z}$, the network $S \cup \{\mathcal{Z}\}$ is polynomial-time *with overwhelming probability*. However, the reader might question whether the additional generality of allowing networks that run in superpolynomial time with negligible probability is not offset by the added complexity. Might not the following notion of *strong reactive polynomial time* be more suitable for defining our security notion:

Definition 30 (Strong reactive polynomial time) *A system S of ITMs runs in* strong reactive polynomial time *if for any* a priori *polynomial time ITM $\mathcal{Z}$ the system $S \cup \{\mathcal{Z}\}$ runs in* a priori *polynomial time (i.e., $S \cup \{\mathcal{Z}\}$* always *terminates after a polynomial number of steps).*

For example, it is not difficult to see that strong reactive polynomial time has the following simple characterisation: For any sequence of incoming messages such that the total length is polynomially-bounded, the system S runs a polynomial number of steps.[37]

Based on the notion of strong reactive polynomial time, we can now define security analogous to Definition 11:

Definition 31 (UC with respect to strong reactive polynomial time) *We say an ITM M is* strongly valid *for π (or ϕ) if $\pi \cup \{M\}$ (or $\phi \cup \{M\}$) runs in strong reactive polynomial time.*

Then π emulates *ϕ* with respect to strong reactive polynomial time *if for any ITM $\mathcal{A}$ that is strongly valid for π, there is an ITM $\mathcal{S}$ that is strongly valid for ϕ such that for every* a priori *polynomial-time ITM $\mathcal{Z}$ the following families of random variables are computationally indistinguishable:*

$$\left\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*} \qquad \textit{and} \qquad \left\{\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*}$$

Although this definition looks very similar to Definition 11, it turns out that it is not a suitable security definition, since not even the Universal Composition Theorem 21 holds (not even its restricted variant Theorem 20):

Theorem 32 *There are protocols π, ϕ and ρ such that*

- *The protocol ρ calls only one instance of its subprotocol.*
- *The protocols π, ϕ, ρ, ρ^π, and ρ^ϕ are strongly reactively polynomial-time.*
- *The protocol π emulates ϕ with respect to strong reactive polynomial time.*
- *But ρ^π does not emulate ρ^ϕ with respect to strong reactive polynomial time.*

Proof. In this proof, we say "emulate" for "emulate with respect to strong reactive polynomial time".

[37] To see this, consider a polynomial-time ITM $\mathcal{Z}$ that sends random messages. Any sequence of message of polynomial length is sent by this ITM with nonzero probability.

We first describe the protocols π and ϕ. The protocol π expects a pair of the form $(1^t, s, b)$ with $t \in \mathbb{N}$, $s \in \mathbb{N}$, and $b \in \{0, 1\}$ from the environment (or the embedding protocol). When $b = 1$, it sends s to the adversary. Otherwise, the message is ignored.

The protocol ϕ also expects a pair of the form $(1^t, s, b)$. If $b = 1$, it sends s to the adversary. If $b = 0$, it sends s to the adversary with probability $\gamma(k) := 2^{-k}$ where k is the security parameter.

Both protocols accept only one message from the environment. Further messages are ignored.

It is easy to see that π and ϕ are both strongly reactively polynomial-time.

We will now show that π emulates ϕ. Let a strongly valid adversary $\mathcal{A}$ be given.[38] We set $\mathcal{S} := \mathcal{A}$. Since ϕ deviates from the program of π with probability at most $\gamma(k)$, the ensembles $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are statistically indistinguishable for any environment $\mathcal{Z}$. To show that π emulates ϕ we therefore only have to show that $\mathcal{S} = \mathcal{A}$ is strongly valid for ϕ. Let an *a priori* polynomial-time ITM $\mathcal{Z}$ be given. Let $\mathcal{Z}'$ be the ITM that simulates $\mathcal{Z}$ with the following modification: When $\mathcal{Z}$ would send a message $(1^t, s, 0)$ to the protocol, $\mathcal{Z}'$ sends with probability $\gamma(k)$ the message $(1^t, s, 1)$ and with probability $1 - \gamma(k)$ the message $(1^t, s, 0)$. Then $\mathrm{TIME}_{\pi,\mathcal{A},\mathcal{Z}'}$ and $\mathrm{TIME}_{\phi,\mathcal{S},\mathcal{Z}}$ have the same distribution and $\mathcal{Z}'$ is *a priori* polynomial-time. Therefore if there is an *a priori* polynomial-time ITM $\mathcal{Z}$ such that $\phi \cup \{\mathcal{S}, \mathcal{Z}\}$ is not *a priori* polynomial-time then there is an *a priori* polynomial-time ITM $\mathcal{Z}'$ such that $\pi \cup \{\mathcal{A}, \mathcal{Z}\}$ is not *a priori* polynomial-time. The latter is a contradiction to the strong validity of $\mathcal{A}$. Thus $\phi \cup \{\mathcal{S}, \mathcal{Z}\}$ is *a priori* polynomial-time and $\mathcal{S}$ is strongly valid. Therefore π emulates ϕ.

We now introduce the protocol ρ. This protocol expects a message $(1^t, s)$ from the environment. Then it sets $b := 1$ if and only if $t = s$ and $b := 0$ otherwise. Finally, it sends $(1^t, s, b)$ to its subprotocol. As did π and ϕ, this protocol accepts only a single message from the environment.

It is straightforward to check that ρ, ρ^π and ρ^ϕ are strongly reactively polynomial-time.

We proceed to show that ρ^π does not emulate ρ^ϕ. Consider the following adversary $\mathcal{A}$. When receiving a message s from the subprotocol π, it sends 1^s to the environment. We first check that $\mathcal{A}$ is strongly valid for ρ^π. The critical point is the fact that $\mathcal{A}$ receives an s in binary representation and outputs 1^s which takes time linear in s, i.e., exponential in the length of s. However, it turns out that $\rho^\phi \cup \{\mathcal{A}\}$ is *a priori* polynomial-time nevertheless. To see this, consider an *a priori* polynomial time ITM $\mathcal{Z}$. Whenever the ITM $\mathcal{Z}$ sends a message $(1^t, s)$ to ρ with $t \neq s$, ρ sends $(1^t, s, 0)$ to π. The message $(1^t, s, 0)$ is ignored by π. So π only outputs s if $\mathcal{Z}$ sends a message $(1^t, s)$ with $s = t$. Since $\mathcal{Z}$ is *a priori* polynomial-time, t is polynomially bounded in the security parameter. Therefore the message s received by the adversary $\mathcal{A}$ is guaranteed to be polynomially bounded, too, so the running time spent by $\mathcal{A}$ for outputting 1^s is polynomially bounded in the security parameter. Hence $\mathcal{A}$ is strongly valid for π.

[38] In the context of UC with respect to strong reactive polynomial time, by strongly valid we mean of course that $\pi \cup \{\mathcal{A}\}$ is *strongly* reactively polynomial-time. The same applies to strongly valid simulators.

Now assume a simulator $\mathcal{S}$ for $\mathcal{A}$. Without loss of generality, we may assume that $\mathcal{S}$ expects a message s from the subprotocol ϕ and then either ignores that message or sends a single message m to the environment. Let $P(k,s)$ denote the probability that the simulator $\mathcal{S}$ sends a message $m = 1^s$ upon receiving s when running with security parameter k. Let $L(k)$ be the largest nonnegative integer such that $P(k,s) \geq \frac{1}{2}$ for all $s \leq L(k)$. (We set $L(k) := \infty$ if $P(k,s) \geq \frac{1}{2}$ for all s.)

We distinguish two cases. First, consider the case that $L(k)$ is polynomially-bounded in k for sufficiently large k. Then we construct an environment $\mathcal{Z}$ that upon security parameter k sends $(1^t, s)$ to ρ with $t := s := L(k) + 1$ and outputs 1 if it receives the message 1^s from the simulator.[39] Obviously, $\mathcal{Z}$ is *a priori* polynomial-time. (In case $L(k)$ is not efficiently computable, we can assume that $\mathcal{Z}$ extracts $L(k)$ from its auxiliary input.) By construction of ρ, π and $\mathcal{A}$, we then have $\Pr\big[\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} = 1\big] = 1$ for sufficiently large k. On the other hand, by definition of $P(k,s)$ we have $\Pr\big[\mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}} = 1\big] = P(k,s) = P(k, L(k)+1) < \frac{1}{2}$ for sufficiently large k (namely whenever $L(k) \neq \infty$). Thus $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}}$ are computationally distinguishable.

In case that $L(k)$ is not polynomially bounded, we construct an ITM $\mathcal{Z}$ that chooses $t := 0$ and $s := \min\{L(k), 2^k\}$ and sends $(1^t, s)$ to ρ. Again, $\mathcal{Z}$ is *a priori* polynomial-time. However, we have $\Pr\big[\mathrm{TIME}_{\rho^\phi,\mathcal{S},\mathcal{Z}} > \min\{L(k),2^k\}\big] \geq \Pr\big[\mathcal{S} \text{ sends } 1^{\min\{L(k),2^k\}}\big] \overset{(*)}{\geq} \gamma(k)P(k,\min\{L(k),2^k\}) \geq \gamma(k)\frac{1}{2} > 0$. Here $(*)$ uses the fact that even in the case $b = 0$, the subprotocol ϕ sends s to the simulator with probability $\gamma(k)$. Thus $\rho^\phi \cup \{\mathcal{S},\mathcal{Z}\}$ does not run in *a priori* polynomial time, so $\mathcal{S}$ is not strongly valid for ϕ. So summarising, there is no strongly valid simulator $\mathcal{S}$ such that $\mathrm{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\rho^\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable for all *a priori* polynomial-time $\mathcal{Z}$. Hence ρ^π does not emulate ρ^ϕ. □

An interesting question at this point is whether this counterexample still holds (possibly with a different choice for γ) if we allow $S \cup \{\mathcal{Z}\}$ to run in *expected* polynomial time in Definition 30. However, in this case consider the simulator $\mathcal{S}$ that accepts any s, but aborts after $1/\gamma(k)$ steps. This simulator produces a good simulation: Since $1/\gamma(k)$ is superpolynomial, the abort occurs only for $t \neq s$. In this case no output is expected from the real adversary either, so the real and ideal views are indistinguishable. And this simulator is strongly valid (w.r.t. expected polynomial time): In the case $t \neq s$, it runs $1/\gamma(k)$ steps with probability $\gamma(k)$.

So at least this counterexample does not apply to a notion using expected polynomial time. However, it demonstrates that the simulator may have to explicitly bound its running time by the inverse of some probability γ, where γ is—intuitively—the probability that a naive simulator would run superpolynomial time. Since it is not clear whether such a bound γ can always be explicitly constructed or efficiently computed, we might expect that, even if it holds, the proof of even the simple composition theorem will be much harder in the case of expected polynomial time. Nevertheless, it would be an interesting question to see how a notion of reactive polynomial time based on expected

[39] Strictly speaking, this definition does not make sense for $L(k) = \infty$. However, this only happens for finitely many k, so we can assume that $\mathcal{Z}$ just aborts in these cases.

polynomial time behaves and what techniques would be used in the proofs.

9.2 Uniform reactive polynomial time

In Definition 10, we allow a reactively polynomial-time network S to run in time $p(k+q)$ where q is the runtime of the ITM $\mathcal{Z}$ and p is some polynomial *that may depend on* $\mathcal{Z}$. As mentioned on page 29, we might also require that p does not depend on $\mathcal{Z}$, leading to a stricter notion of *uniform reactive polynomial time*. In this appendix, we define this alternative notion and show that the properties we proved Sections 5–7 also hold for this somewhat stricter notion. Thus the choice which notion to use is more a matter of personal preference than of formal necessity. However, it should be noted that with uniform reactive polynomial time, some arguments are a slightly more awkward since one has to keep track that p is independent of $\mathcal{Z}$. (This is somewhat reminiscent of the difference between UC and specialised-simulator UC [Lin03].)

Definition 33 (Uniform reactive polynomial time) *A system S of ITMs runs in uniform reactive polynomial time if there exists a polynomial p such that for any* a priori *polynomial time ITM $\mathcal{Z}$ and any polynomial q bounding the running time of $\mathcal{Z}$ (cf. Definition 8), there is a negligible function μ such that for all $k \in \mathbb{N}$ and $z \in \{0,1\}^*$ we have that $\mathrm{TIME}_{S\cup\{\mathcal{Z}\}}(k,z) > p(k+q(k))$ with probability at most $\mu(k)$.*

Definition 34 (UC with respect to uniform reactive polynomial time) *We say an ITM M is* uniformly valid *for π (or ϕ) if $\pi \cup \{M\}$ (or $\phi \cup \{M\}$) runs in uniform reactive polynomial time.*

Then π emulates ϕ (with respect to uniform reactive polynomial time) *if for any ITM $\mathcal{A}$ that is uniformly valid for π, there is an ITM $\mathcal{S}$ that is uniformly valid for ϕ such that for every* a priori *polynomial-time ITM $\mathcal{Z}$ the following families of random variables are computationally indistinguishable:*

$$\left\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*} \qquad \textit{and} \qquad \left\{\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}(k,z)\right\}_{k\in\mathbb{N},z\in\{0,1\}^*}$$

In the following sections, we show that the properties we proved in Sections 5–7 still hold for the alternative notion in Definitions 33 and 34.

Basic properties. Lemma 12 still holds because uniform reactive polynomial time implies reactive polynomial time. So the conditions of Lemma 12 also hold the present setting. Lemmas 13 and 15 holds with identical proofs since these proofs do not use the definition of validity at all. Thus all results from Section 5 still hold for uniform reactive polynomial time.

Dummy-Adversary. All our results concerning the dummy adversary carry over to the case of uniform reactively polynomial time.

Lemma 35 (Uniform validity of the dummy adversary) *If π is a uniformly reactively polynomial-time protocol, then the dummy adversary is uniformly valid for π.*

Proof. Let $\mathcal{Z}$ be an ITM with runtime polynomial q and consider the system $\{\mathcal{Z}, \tilde{\mathcal{A}}\} \cup \pi$. Since $\tilde{\mathcal{A}}$ only forwards messages between $\mathcal{Z}$ and π, we can construct an *a priori* polynomial-time ITM $\mathcal{Z}'$ that directly sends and receives those messages to and from π. Then, assuming the same random tapes in both networks, $\mathrm{TIME}_{\{\mathcal{Z},\tilde{\mathcal{A}}\}\cup\pi}(k,z) \leq c \cdot \mathrm{TIME}_{\{\mathcal{Z}'\}\cup\pi}(k,z)$ for some fixed $c > 0$ (independent of $\mathcal{Z}$). Since π is uniformly reactively polynomial-time, we have that $\mathrm{TIME}_{\{\mathcal{Z}'\}\cup\pi}(k,z) \leq p(k+q(k))$ with overwhelming probability in k for some polynomial p which is independent of $\mathcal{Z}'$. Thus $\mathrm{TIME}_{\{\mathcal{Z},\tilde{\mathcal{A}}\}\cup\pi}(k,z) \leq c \cdot p(k+q(k))$ with overwhelming probability. Since this holds for all $\mathcal{Z}$ (and the polynomial $c \cdot p$ does not depend on $\mathcal{Z}$), it follows that $\{\tilde{\mathcal{A}}\} \cup \pi$ is uniformly reactively polynomial-time and thus $\tilde{\mathcal{A}}$ is uniformly valid for π. □

Definition 36 (Uniform emulation with respect to the dummy adversary) *We say* π emulates ϕ with respect to the dummy adversary and uniform reactive polynomial time *if for the dummy adversary* $\tilde{\mathcal{A}}$ *there is an ITM* $\tilde{\mathcal{S}}$ *that is uniformly valid for* ϕ *such that for every* a priori *polynomial-time ITM* $\mathcal{Z}$ *the ensembles* $\mathrm{EXEC}_{\pi,\tilde{\mathcal{A}},\mathcal{Z}}$ *and* $\mathrm{EXEC}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}}$ *are computationally indistinguishable.*

Theorem 37 (Completeness of the dummy adversary) *Assume that* π *is uniformly reactively polynomial-time. Then* π *emulates* ϕ *with respect to uniform reactive polynomial time if and only if* π *emulates* ϕ *with respect to the dummy adversary and uniform reactive polynomial time.*

Proof. We describe the changes that must be applied to the proof of Theorem 19. First, consider the construction of the polynomial p that bounds $\mathrm{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ with overwhelming probability. In the present case we can achieve a stronger condition: We can choose p such that $p(k) \leq \tilde{p}(k+q(k))$ for any polynomial q bounding the running time of $\mathcal{Z}$ where $\tilde{p}$ is a fixed polynomial independent of $\mathcal{Z}$ and q. Then, the construction of the simulator $\mathcal{S}$ and the proof that $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable is unchanged. (It does not use the definition of validity, only the property that p bounds $\mathrm{TIME}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ with overwhelming probability.) Thus it is only left to show that $\mathcal{S}$ is uniformly valid.

Since $\mathcal{Z}'_p$ simulates $\mathcal{Z}$ and $\mathcal{A}$, but $\mathcal{A}$ for at most p steps, we have that the running time of $\mathcal{Z}'_p$ is bounded by $q'(k) := c_1 \cdot (q(k) + p(k))$ for some constant c_1 (in the sense of Definition 8). The constant c_1 reflects a possible simulation overhead and is independent of $\mathcal{Z}$ and q. Since $\tilde{\mathcal{S}}$ is uniformly valid for ϕ, it follows that $\mathrm{TIME}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'_p} \leq p_1(k + q'(k))$ with overwhelming probability. Again, p_1 is independent of $\mathcal{Z}$ and q. Then, since the network $\phi \cup \{\mathcal{Z}'_p, \tilde{\mathcal{S}}\}$ behaves differently from $\phi \cup \{\mathcal{Z}', \tilde{\mathcal{S}}\}$ only if $\mathcal{Z}'_p$ outputs `beep` which happens with negligible probability, it follows that $\mathrm{TIME}_{\phi,\tilde{\mathcal{S}},\mathcal{Z}'} \leq c_2 \cdot p_1(k + q'(k))$ with overwhelming probability. Here c_2 again represents some simulation overhead independent of $\mathcal{Z}$ and q. Then we also have $\mathrm{TIME}_{\phi,\mathcal{S},\mathcal{Z}} \leq c_3 c_2 \cdot p_1(k + q'(k))$ with overwhelming probability with some overhead c_3 independent of $\mathcal{Z}$ and q. Substituting the definitions of q' and p, we get that $\mathrm{TIME}_{\phi,\mathcal{S},\mathcal{Z}} \leq c_3 c_2 \cdot p_1(k + c_1 \cdot (q(k) + \tilde{p}(k + q(k))))$ where $c_1, c_2, c_3, p_1, \tilde{p}$ are independent of $\mathcal{Z}$ and q. Thus we can choose some polynomial

p^* independent of $\mathcal{Z}$ and q such that $\text{TIME}_{\phi,\mathcal{S},\mathcal{Z}} \leq p^*(k + q(k))$. Since this holds for every *a priori* polynomial-time $\mathcal{Z}$ and any q bounding the running time of $\mathcal{Z}$, it follows that $\phi \cup \{\mathcal{S}\}$ is uniformly reactively polynomial-time and thus $\mathcal{S}$ uniformly valid for ϕ.□

Universal Composition Theorem. Since the Simple Composition Theorem is a direct consequence of the Universal Composition Theorem, it is sufficient to show that the Universal Composition Theorem 21 holds for uniform reactive polynomial time.

Theorem 38 (Universal Composition Theorem for uniform reactive polynomial time) *Let π, ϕ and ρ be protocols, such that π and ρ^π are uniformly reactively polynomial-time. The protocol ρ may call an arbitrary number of subprotocol instances. Assume that π emulates ϕ. Then ρ^π emulates ρ^ϕ.*

We will now sketch the modifications that need to be applied to the proof of Theorem 21 in order to prove Theorem 38. We assume the notation used in the proof of Theorem 21. Similar to that proof, we here let $\mathcal{A}$ denote the dummy adversary and choose a fixed simulator $\mathcal{S}$ such that $\phi \cup \{\mathcal{S}\}$ is uniformly reactively polynomial-time, and that for every *a priori* polynomial-time $\mathcal{Z}$ we have that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable. Additionally, by $r_{\mathcal{Z}}$ we denote a polynomial bounding the running time of $\mathcal{Z}$ (in the sense of Definition 8).

Then, for the new proof Definitions 22, 23 and 24 and Lemmas 25, 27, 28, and 29 remain unchanged. These lemmas were shown to hold under the assumption that π, ρ^π, and $\{\mathcal{S}\} \cup \phi$ are reactively polynomial-time, that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ for all *a priori* polynomial-time $\mathcal{Z}$, and that $\mathcal{Z}$ is an *a priori* polynomial-time environment. Then the lemmas in particular hold under the stronger condition of the present proof that π, ρ^π, and $\{\mathcal{S}\} \cup \phi$ are uniformly reactively polynomial-time, that $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ for all *a priori* polynomial-time $\mathcal{Z}$, and that $\mathcal{Z}$ is an *a priori* polynomial-time environment. The same holds for Lemma 26, but we need to somewhat strengthen Lemma 26:

Lemma 39 *In the situation of Definition 22, there exist polynomials $p = p(k)$ and $q = q(k)$, and a negligible function $\mu = \mu(k)$ such that for all $k \in \mathbb{N}$ and all auxiliary inputs $z \in \{0,1\}^*$ for $\mathcal{Z}$, the following holds. We have that $\Pr[B_{p,q}] \leq \mu(k)$, both in $\pi \cup \{\mathcal{A}, \mathcal{Z}^*_{1,p}\}$ and in $\phi \cup \{\mathcal{S}, \mathcal{Z}^*_{1,p}\}$.*

Moreover, we can write p and q as $p(k) = \tilde{p}(k + r_{\mathcal{Z}}(k))$ and $q(k) = \tilde{q}(k + r_{\mathcal{Z}}(k))$ where $\tilde{p}$ and $\tilde{q}$ do not depends on $\mathcal{Z}$ and $r_{\mathcal{Z}}$.

(Note that only the part after *moreover* is changed with respect to Lemma 26.)
Proof. To show Lemma 39, we have to show that in the proof of Lemma 26 we can choose p and q such that they additionally satisfy the conditions $p(k) = \tilde{p}(k + r_{\mathcal{Z}}(k))$ and $q(k) = \tilde{q}(k + r_{\mathcal{Z}}(k))$.

For p this is straightforward: p was chosen as a polynomial such that $\text{TIME}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \leq p(k)$ with overwhelming probability. Since in our setting, $\rho^\pi \cup \{\mathcal{A}\}$ is uniformly reactively polynomial-time, and since $r_{\mathcal{Z}}$ bounds the running time of $\mathcal{Z}$, we can therefore choose p with $p(k) = \tilde{p}(k + r_{\mathcal{Z}}(k))$ where $\tilde{p}$ is independent of $r_{\mathcal{Z}}$ and $\mathcal{Z}$.

For q the situation is slightly more complicated. The polynomial q was chosen such that $\text{TIME}_{\phi,\mathcal{S},[\mathcal{Z}^*_{1,p}]_p} \leq q(k)$ with overwhelming probability. To show that q can fulfill the additional constraint, we first have to analyze the runtime bound of $[\mathcal{Z}^*_{1,p}]_p$. By construction, $[\mathcal{Z}^*_{1,p}]_p$ simulates $\mathcal{Z}$, ρ and at most p instances of the dummy adversary and π. Furthermore, ρ and each instance of π is executed for at most p steps. Therefore the running time of $[\mathcal{Z}^*_{1,p}]_p$ is bounded by $s_1(k) := s_2(k + r_{\mathcal{Z}}(k) + p(k))$ for some polynomial s_2 that does not depend on $\mathcal{Z}$ and $r_{\mathcal{Z}}$. Since $\phi \cup \{\mathcal{S}\}$ is uniformly reactively polynomial-time by assumption, it follows that $\text{TIME}_{\phi,\mathcal{S},[\mathcal{Z}^*_{1,p}]_p} \leq s_3(k + s_1(k))$ with overwhelming probability where the polynomial s_3 does not depend on $\mathcal{Z}$ and $r_{\mathcal{Z}}$. We can therefore choose a polynomial $\tilde{q}$ with $\tilde{q}(k + r_{\mathcal{Z}}(k)) \geq s_3(k + s_2(k + r_{\mathcal{Z}}(k) + \tilde{p}(k + r_{\mathcal{Z}}(k)))) = s_3(k + s_1(k))$ such that $\tilde{q}$ does not depend on $\mathcal{Z}$ and $r_{\mathcal{Z}}$. Then $\text{TIME}_{\phi,\mathcal{S},[\mathcal{Z}^*_{1,p}]_p} \leq \tilde{q}(k + r_{\mathcal{Z}}(k)) =: q(k)$ with overwhelming probability, so we have shown that we can choose q satisfying the additional constraint $q(k) = \tilde{q}(k + r_{\mathcal{Z}}(k))$. □

We are now ready to prove Theorem 38. The construction of the simulator $\mathcal{S}^\infty$ and the proof that $\text{EXEC}_{\rho^\pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}$ are as in the proof of Theorem 21. However, to prove Theorem 38, we need to additionally show that $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is uniformly reactively polynomial-time. To achieve this, we first show as for Theorem 21 that $\Pr[B]$ is negligible in $\rho^\phi \cup \{\mathcal{S}^\infty, \mathcal{Z}\}$. Furthermore, note that by construction of $\mathcal{S}^\infty$ there is a fixed polynomial s (not depending on $\mathcal{Z}$ or $r_{\mathcal{Z}}$) such that $\text{TIME}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}(k, z) \leq s(k + R_{k,z} + P^1_{k,z} + P^2_{k,z} + Q_{k,z})$ where the random variable $R_{k,z}$ denotes the number of steps $\mathcal{Z}$ runs, $P^1_{k,z}$ denotes the number of steps the machines from ρ run, $P^2_{k,z}$ denotes the number of sessions of π invoked, and $Q_{k,z}$ the maximum number of steps any of the instances of π runs. By definition of $r_{\mathcal{Z}}$ we have $R_{k,z} \leq r_{\mathcal{Z}}(k)$ with probability 1, and by definition of $B = B_{p,q}$, the fact that $\Pr[B]$ is negligible implies that $P^1_{k,z} \leq p(k)$, $P^2_{k,z} \leq p(k)$, and $Q_{k,z} \leq p(k)$ holds with overwhelming probability. Thus with overwhelming probability we have $\text{TIME}_{\rho^\phi,\mathcal{S}^\infty,\mathcal{Z}}(k, z) \leq s(k + r_{\mathcal{Z}}(k) + 2p(k) + q(k)) \overset{(*)}{=} s(k + r_{\mathcal{Z}}(k) + 2\tilde{p}(k + r_{\mathcal{Z}}(k)) + \tilde{q}(k + r_{\mathcal{Z}}(k))) \leq \tilde{s}(k + r_{\mathcal{Z}}(k))$ for a suitable polynomial $\tilde{s}$ that does not depend on $\mathcal{Z}$ or $r_{\mathcal{Z}}$. (Here $(*)$ uses Lemma 39.) Since this holds for any *a priori* polynomial-time $\mathcal{Z}$, we have that $\rho^\phi \cup \{\mathcal{S}^\infty\}$ is uniformly reactively polynomial-time and Theorem 38 follows. □

10 Relation to classical notions

In this section we investigate in what relation our notion stands to the classical UC definitions. Since the classical definitions are not meaningful for protocols that are not *a priori* polynomial-time, we are interested in the case that π and ϕ are *a priori* polynomial-time protocols. In this case, it turns out that UC with respect to reactive polynomial time lies strictly between two common classical definitions: UC and specialized-simulator UC[40]. To show the strictness of these implications, we need the following complexity

[40]Specialized-simulator UC is defined like UC, with the difference that the simulator may depend on the environment [Lin03]. We stress that we consider the specialized-simulator UC notion as defined in [Lin03], which is *not* equivalent to the UC notion from [Can05a]. There also exists a specialized-simulator UC variant in [Can05a] that *is* equivalent to standard UC (see [Can05a, Claim 12]).

assumption:

Definition 40 (Time-lock puzzle) *A* time-lock puzzle *consists of an ITM $\mathcal{V}$ (the* verifier*) and an ITM $\mathcal{P}$ (the* prover*) such that*

- *Given arguments $(1^k, s)$, the ITM $\mathcal{V}$ runs in polynomial time in k. Given arguments $(1^k, s)$, the ITM $\mathcal{P}$ runs in polynomial time in $k + s$.*
- Easiness. *For any polynomial p we have that*

$$\min_{s \leq p(k)} P\big(\langle \mathcal{P}(1^k, s), \mathcal{V}(1^k, s)\rangle = 1\big)$$

  is overwhelming in k. (We call s the hardness of the puzzle.)
- Hardness. *For any ITM B running in polynomial time in the length of its first argument there exists a polynomial p, such that*

$$\sup_{\substack{s \geq p(k) \\ z \in \{0,1\}^*}} P\big(\langle B(1^k, s, z), \mathcal{V}(1^k, s)\rangle = 1\big)$$

  is negligible in k.

In this definition $\langle \mathcal{P}, \mathcal{V}\rangle$ denotes the distribution of the output of $\mathcal{V}$ after an interaction with $\mathcal{P}$.

Note the following differences between our definition and that of [HU05, HU06]: First, following [Unr06], we allow interactive time-lock puzzles, while [HU05] used the stronger assumption of non-interactive ones. However, all results of [HU06] were shown to hold also for interactive time-lock puzzles [Unr06]. Further, [HU05, HU06, Unr06] allow the prover to depend of the polynomial p in the easiness condition while we require the same prover for any p, i.e., we impose a uniformity requirement on honest prover. All constructions known to the authors (in particular those from [RSW96, Unr06]) fulfill this additional requirement.

We can now state the relations between our model and classical notions for the case of *a priori* polynomial-time protocols. Note that we have included another notion besides classical UC and classical specialized-simulator UC, namely general composability. Intuitively, general composability is the weakest security notion that still fulfils the Universal Composition Theorem 21. Although no workable characterisation for this notion is known, it is insofar an important notion that is specifies the minimum properties we might expect from a UC-like security notion.

Theorem 41 *By classical UC we denote UC as defined in Definition 4, where polynomial time means* a priori *polynomial time. By classical specialized-simulator UC we denote the notion from [Lin03] which is defined like classical UC, except that the simulator may depend on the environment.*

A protocol π is said to emulate ϕ with respect to (polynomially-bounded) general composability if for every a priori *polynomial-time protocol ρ we have that ρ^π emulates ρ^ϕ in the stand-alone model (see [Lin03] for a detailed definition of general composability).*

Then for a priori *polynomial-time protocols π and ϕ, consider the following statements.*

(i) π *emulates* ϕ *with respect to classical UC.*
(ii) π *emulates* ϕ *with respect to reactive polynomial time.*
(iii) π *emulates* ϕ *with respect to general composability.*
(iv) π *emulates* ϕ *with respect to classical specialized-simulator UC.*
Then (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (iv).

If time-lock puzzles exist, all implications are strict in the sense that there is a pair of protocols π*,* ϕ *such that the implication does not hold.*

Proof. First we show (i) $\Rightarrow$ (ii), i.e., that if π emulates ϕ with respect to classical UC, then π emulates ϕ with respect to reactive polynomial time.

Let p a polynomial such that the running time of π upon security parameter k is bounded by $p(k)$.

Let $\tilde{\mathcal{A}}_p$ be defined like the dummy adversary, except that upon security parameter k, no message of length greater than $p(k)$ is sent or received to/from the protocol or environment, and at most $p(k)$ messages are sent to/from the environment and the protocol, respectively.

Then π emulates ϕ with respect to reactive polynomial time if and only if π emulates ϕ with respect to reactive polynomial time and the dummy adversary $\tilde{\mathcal{A}}_p$. This is shown analogous to Theorem 19, except that we additionally use that we can w.l.o.g. assume the environment not to sent more than $p(k)$ messages or messages of length greater than $p(k)$ through the dummy adversary since the protocol (having runtime bound $p(k)$) would not be able to read these superfluous messages.

Assume that π emulates ϕ with respect to classical UC. Since $\tilde{\mathcal{A}}_p$ is *a priori* polynomial-time, by definition of classical UC there is a *a priori* polynomial-time simulator $\tilde{\mathcal{S}}_p$ such that for all *a priori* polynomial-time environments $\mathcal{Z}$ the ensembles $\mathrm{EXEC}_{\pi,\tilde{\mathcal{A}}_p,\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\tilde{\mathcal{S}}_p,\mathcal{Z}}$ are computationally indistinguishable. Since $\tilde{\mathcal{S}}_p$ and π are *a priori* polynomial-time, the network $\pi \cup \{\tilde{\mathcal{S}}_p\}$ is *a priori* polynomial-time and therefore in particular reactively polynomial-time. So $\tilde{\mathcal{S}}_p$ is valid for ϕ. Thus π emulates ϕ with respect to reactive polynomial time and the dummy adversary $\tilde{\mathcal{A}}_p$. As seen above, this implies that π emulates ϕ with respect to reactive polynomial time. This shows (i) $\Rightarrow$ (ii).

Now we are going to show (ii) $\Rightarrow$ (iv), i.e., that if π emulates ϕ with respect to reactive polynomial time, then π emulates ϕ with respect to classical specialised-simulator UC. To prove this, let an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ be given, both *a priori* polynomial-time, and we have to show that there is an *a priori* polynomial-time simulator $\mathcal{S}$ such that $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable.

Since $\mathcal{A}$ and π are *a priori* polynomial-time, $\mathcal{A}$ is valid for π. By assumption, π emulates ϕ with respect to reactive polynomial time, so there is a valid simulator $\mathcal{S}'$ for ϕ such that the ensembles $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\phi,\mathcal{S}',\mathcal{Z}}$ are computationally indistinguishable. However, $\mathcal{S}'$ is not necessarily *a priori* polynomial-time. Since $\mathcal{S}'$ is valid, and $\mathcal{Z}$ is *a priori* polynomial-time, the network $\phi \cup \{\mathcal{S}', \mathcal{Z}\}$ is polynomial-time with overwhelming probability, so there is a polynomial p such that $\mathrm{TIME}_{\phi,\mathcal{S}',\mathcal{Z}}(k, z) \leq p(k)$ with overwhelming probability. So in particular $\mathcal{S}'$ runs at most $p(k)$ steps with overwhelming probability. Let $\mathcal{S}$ be as $\mathcal{S}'$, except that when running more than $p(k)$ steps $\mathcal{S}$

aborts. Since this happens only with negligible probability in an execution of $\phi \cup \{\mathcal{S}, \mathcal{Z}\}$, we have that $\text{EXEC}_{\phi,\mathcal{S}',\mathcal{Z}}$ and $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable (in fact even statistically indistinguishable). Summarising, $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\pi,\mathcal{S},\mathcal{Z}}$ are computationally indistinguishable, and $\mathcal{S}$ is *a priori* polynomial-time, thus π emulates ϕ with respect to classical specialised-simulator UC.

Now we show (ii) $\Rightarrow$ (iii), i.e., that if π emulates ϕ with respect to reactive polynomial time, then π emulates ϕ with respect to general composability. For any *a priori* polynomial-time protocol ρ, both ρ^π and ρ^ϕ are *a priori* polynomial-time and thus in particular reactively polynomial-time. Thus by Theorem 21 ρ^π emulates ρ^ϕ with respect to reactive polynomial time. Above we showed that for *a priori* polynomial-time protocols, reactive polynomial time UC implies classical specialised-simulator UC, so ρ^π emulates ρ^ϕ with respect to classical specialised-simulator UC. This again implies that ρ^π emulates ρ^ϕ in the stand-alone model (see [Lin03]). Since this holds for any *a priori* polynomial-time protocol ρ, we have that π emulates ϕ with respect to general composability.

In [Lin03] it was shown that (iii) $\Rightarrow$ (iv), so summarising we have (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (iv). So all implications are proven.

We are left to show that the implications are strict if time-lock puzzles exist.

First, we show that there are protocols π_1 and ϕ_1 such that π_1 emulates ϕ_1 with respect to general composability, but π_1 does not emulate ϕ_1 with respect to reactive polynomial time. For this purpose, we use a pair of protocols proposed in [HU05] to separate the notions of UC and specialised-simulator UC.[41] We give a short sketch of their construction. For this, we first review the definition of a time-lock puzzle. A time-lock puzzle is an interactive protocol where one party (the prover) tries to convince another party (the verifier) that he has a given amount of computational power. More exactly, the verifier gets a parameter $s \in \mathbb{N}$ (the strength of the puzzle) as input. The ensuing interaction we call the time-lock puzzle. If the verifier output 1 after that interaction, we say the prover solved the puzzle. For any polynomial p, there is an *a priori* polynomial-time prover P such that P solves time-lock puzzles with strength $s \leq p(k)$ with overwhelming probability. On the other hand, for any *a priori* polynomial-time prover B, there is a polynomial q such that B solves puzzles of strength $s \geq q(k)$ only with negligible probability. For a formal definition, see [HU05] (who only investigate the case of one-round time-lock puzzles) or [Unr06] (which generalises the results of [HU05]).

The protocols proposed in [HU05] are the following (called M_0 and M_1 there). Let k denote the security parameter. The protocol π_1 first randomly chooses a strength $s \in \{2^0, \ldots, 2^k\}$. Then it performs a time-lock puzzle of strength s with the environment as prover. After this, it performs a time-lock puzzle of strength s with the adversary as prover. After this, π_1 sends the message $b = 0$ to the environment.

The protocol ϕ_1 behaves identically to π_1, with the following difference: When the environment solves the time-lock puzzle and the simulator does not solve it, then ϕ_1

[41] Actually, [HU05] separated the corresponding notions in the Reactive Simulatability framework [PW01, BPW04b]. However, all their proof carry easily over to the UC framework. The same holds for [HU06].

sends the message $b = 1$ to the environment. Otherwise $b = 0$ is sent to the environment as would have done π_1.

Then π_1 does not emulate ϕ_1 with respect to classical UC due to the following reason: For any *a priori* polynomial-time simulator $\mathcal{S}$, there is a polynomial p such that $\mathcal{S}$ solves puzzles with strength $s \geq p(k)$ only with negligible probability. Furthermore there is an *a priori* polynomial-time environment that can solve puzzles of strength $s \leq 2p(k)$ with overwhelming probability. Since a puzzle of strength $p(k) \leq s \leq 2p(k)$ is asked by ϕ_1 with probability $\frac{1}{k}$, with noticeable probability the environment solves the puzzle while the simulator does not. Thus the environment gets message $b = 1$ with noticeable probability when running with ϕ_1 and $\mathcal{S}$, but gets only $b = 0$ when running with π_1 and some adversary; the environment can hence distinguish. Since for any simulator such a distinguishing simulator exists, π_1 does not emulate ϕ_1 with respect to classical UC.

On the other hand, if the simulator may depend on the environment, as in the case of classical specialised-simulator UC, let p be a polynomial such that the *a priori* polynomial-time environment $\mathcal{Z}$ solves puzzles of strength $s \geq p(k)$ only with negligible probability. Then we can construct an *a priori* polynomial-time simulator that solves all puzzles of strength $s \leq p(k)$. With overwhelming probability it then holds that if the environment solves the puzzle, the simulator does so, too. Thus the message sent by ϕ_1 will be $b = 0$ with overwhelming probability, so that the environment cannot distinguish ϕ_1 from π_1. Therefore π_1 emulates ϕ_1 with respect to classical specialised-simulator UC.

For detailed constructions and proofs we refer to [HU05]. The result can somewhat be strengthened: It is easy to see that the proof that π_1 emulates ϕ_1 with respect to classical specialised-simulator UC generalises to the case where a polynomial number of copies of π_1 and ϕ_1, respectively, run concurrently. From this it follows that π_1 emulates ϕ_1 with respect to general composability [Lin03]. This is detailed in [Unr06].

We now show that π_1 does not emulate ϕ_1 with respect to reactive polynomial time. From this it follows that the implication (ii) $\Rightarrow$ (iii) is strict.

Let $\mathcal{A}$ be the *a priori* polynomial-time adversary that solves time-lock puzzles given by π up to an (arbitrarily chosen) strength of $s = 1$. Since π_1 and $\mathcal{A}$ are *a priori* polynomial, $\mathcal{A}$ is valid for π_1. For a polynomial p, let $\mathcal{Z}_p$ be the *a priori* polynomial-time environment that solves time-lock puzzles given by π_1 or ϕ_1 of a strength of $s \leq p(k)$ with overwhelming probability. Let $\mathcal{S}$ be any simulator that is valid for ϕ_1. Then $\phi_1 \cup \{\mathcal{S}, \mathcal{Z}_0\}$ is polynomial-time with overwhelming probability, so there is a polynomial q bounding $\mathrm{TIME}_{\phi_1,\mathcal{S},\mathcal{Z}_0}$. Let $\mathcal{S}_q$ be the simulator that behaves as does $\mathcal{S}$, but aborts when running more than $q(k)$ steps. Then $\mathcal{S}_q$ is *a priori* polynomial-time, so there is a polynomial r such that in an execution of $\phi_1 \cup \{\mathcal{S}_q, \mathcal{Z}_0\}$ the simulator $\mathcal{S}_q$ solves time-lock puzzles of strength $s \geq r(k)$ only with negligible probability. Since $\mathcal{S}_q$ simulates $\mathcal{S}$ faithfully up to a negligible probability in an execution of $\phi_1 \cup \{\mathcal{S}_q, \mathcal{Z}_0\}$, it follows that also $\mathcal{S}$ solves time-lock puzzles of strength $s \geq r(k)$ only with negligible probability in an execution of $\phi_1 \cup \{\mathcal{S}, \mathcal{Z}_0\}$. Since the messages sent by ϕ_1 to $\mathcal{S}$ do not depend on whether the environment solves its puzzle or not, the probability that $\mathcal{S}$ solves time-lock puzzles of strength $s \geq r(k)$ in an execution of $\phi_1 \cup \{\mathcal{S}, \mathcal{Z}_{2p}\}$ is negligible, too. On the other hand, $\mathcal{Z}_{2p}$ solves puzzles with strength $s \leq 2p(k)$ with overwhelming probability.

Since ϕ_1 chooses $p(k) \leq s \leq 2p(k)$ with probability $\frac{1}{k}$, it follows that with noticeable probability the environment $\mathcal{Z}_p$ solves its puzzle while the simulator $\mathcal{S}$ does not. Then the message $b = 1$ is send to the environment by ϕ_1 so that the environment $\mathcal{Z}_p$ can distinguish between π_1 and ϕ_1. Therefore π_1 does not emulate ϕ_1 with respect to reactive polynomial time. Since π_1 does emulate ϕ_1 with respect to general composability (see above), the implication (ii) $\Rightarrow$ (iii) is strict.

We will now show that the implication (i) $\Rightarrow$ (ii) is strict. For this, we use a slight modification of the protocols given by [HU05]. We modify π_1 and ϕ_1 insofar that the time-lock puzzle is only given to the adversary/simulator if the environment beforehand solves its time-lock puzzle. We call the resulting protocols π_2 and ϕ_2. For these modified protocols the results from [HU05] still hold (with almost unmodified proofs), in particular π_2 does not emulate ϕ_2 with respect to classical UC. However, we will show that π_2 does emulate ϕ_2 with respect to reactive polynomial time. From this it follows that the implication (i) $\Rightarrow$ (ii) is strict.

By Theorem 19 it is sufficient to construct a simulator $\tilde{\mathcal{S}}$ for the dummy adversary $\tilde{\mathcal{A}}$. This simulator $\tilde{\mathcal{S}}$ behaves like the dummy adversary: It follows the instructions given by the environment (since the dummy adversary would do so, too) and forwards all messages from the protocol ϕ_2 to the environment. But whenever the environment instructs the simulator to send a given solution a for the time-lock puzzle to ϕ_2, the simulator runs the algorithm for solving the puzzle (which runs in polynomial-time in s and outputs a correct solution a' with overwhelming probability) and then sends that correct solution a' instead of a.[42] Since the simulator solves all puzzles with overwhelming probability, the message sent by ϕ_2 to the environment will be $b = 1$ with overwhelming probability, and therefore the environment cannot distinguish. It is left to show that $\mathcal{S}$ is valid for ϕ. The only critical point is the running time of the algorithm for solving the time-lock puzzle. Let an *a priori* polynomial-time environment $\mathcal{Z}$ be given. Then there exists a polynomial p such that the probability is negligible that $\mathcal{Z}$ solves puzzles with strength $s \geq p(k)$. Since by construction ϕ_2 give a puzzle of strength s to the simulator only if the environment previously solved a puzzle of that strength. Therefore ϕ_2 gives puzzles of strength $s \geq p(k)$ to $\mathcal{S}$ only with negligible probability. Since the running time needed by $\mathcal{S}$ for solving the puzzle is bounded by $q(s)$ for some polynomial s, it follows that when interacting with $\mathcal{Z}$ the running time needed by $\mathcal{S}$ for solving the puzzle is bounded by $q(p(k))$ with overwhelming probability. Thus $\pi \cup \{\mathcal{S}, \mathcal{Z}\}$ is polynomial-time with overwhelming probability, and since this holds for all *a priori* polynomial-time environments $\mathcal{Z}$, it follows that $\mathcal{S}$ is valid for ϕ_2. Thus π_2 emulates ϕ_2 with respect to reactive polynomial time time. Since π_2 does not emulate ϕ_2 with respect to classical UC, the implication (i) $\Rightarrow$ (ii) is strict.

We have shown that the implications (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) are strict. In [HU06] it was shown that the implication (iii) $\Rightarrow$ (iv) is strict, too, given the existence of time-lock puzzles. So all implications given in the theorem are strict. □

[42] This assume that the solution to the time-lock puzzle is a single message as in [HU05]. If the solution is an interaction as in [Unr06], the simulator will first solve (interactively) the puzzle given by ϕ_2 and then (interactively) give a new puzzle of the same strength to the environment.

Acknowledgements. Dominique Unruh was supported by European Social Fund's Doctoral Studies and Internationalisation Programme DoRa. Much of the work was done while Dominique Unruh was at the University of Karlsruhe and at the Saarland University. Part of the work was performed while Dennis Hofheinz was at CWI, Amsterdam, and supported by an NWO Veni grant.

References

[Bac02] Michael Backes. *Cryptographically Sound Analysis of Security Protocols*. PhD thesis, Universität des Saarlandes, 2002. Online available at `http://www.infsec.cs.uni-sb.de/~backes/papers/PhDthesis.ps.gz`.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In *10th ACM Conference on Computer and Communications Security, Proceedings of CCS 2003*, pages 220–230. ACM Press, 2003. Extended abstract, extended version online available at `http://eprint.iacr.org/2003/015.ps`.

[BPW04a] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In Moni Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer-Verlag, 2004. Online available at `http://www.zurich.ibm.com/security/publications/2004/BaPfWa2004MoreGeneralComposition.pdf`.

[BPW04b] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004. Online available at `http://eprint.iacr.org/2004/082.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[Can04a] Ran Canetti. On universally composable signature, certification and authentication. IACR ePrint 2003/239, June 2004. Version of 2004-06-26.

[Can04b] Ran Canetti. On universally composable signature, certification and authentication. IACR ePrint 2003/239, August 2004. Version of 2004-08-15.

[Can05a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint 2000/067, December 2005. Version of 2005-12-14.

[Can05b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint 2000/067, January 2005. Version of 2005-01-28.

[Can08] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Manuscript, 2008.

[CCK+06a] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Task-structured probabilistic I/O automata. Technical Report MIT-CSAIL-TR-2006-060, MIT CSAIL, September 2006. Online available at `http://dspace.mit.edu/handle/1721.1/33964`.

[CCK+06b] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In *DISC*, pages 238–253, 2006.

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Concurrent zero-knowledge requires $\tilde{\Omega}(\log n)$ rounds. In *33th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2001*, pages 570–579. ACM Press, 2001.

[DKMR05] Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. In Joe Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 476–494. Springer-Verlag, 2005. Online available at `http://www.ti.informatik.uni-kiel.de/~kuesters/publications_html/DattaKuestersMitchellRamanathan-TCC-2005.ps.gz`.

[Fei90] Uri Feige. *Alternative Models for Zero Knowledge Interactive Proofs*. PhD thesis, Weizmann Institute of Science, 1990.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9(3):167–190, 1996.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[Gol01] Oded Goldreich. *Foundations of Cryptography – Volume 1 (Basic Tools)*. Cambridge University Press, August 2001. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[Gol04] Oded Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications)*. Cambridge University Press, May 2004. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[Gol07] Oded Goldreich. On expected probabilistic polynomial-time adversaries: A suggestion for restricted definitions and their benefits. In Salil Vadhan, editor, *Theory of Cryptography, Proceedings of TCC 2007*, Lecture Notes in Computer Science, pages 174–193. Springer-Verlag, 2007. Online available at `http://eprint.iacr.org/2006/277.ps`.

[HMQU05] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Polynomial runtime in simulatability definitions. In *18th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2005*, pages 156–169. IEEE Computer Society, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05polynomial.html`.

[HS11] Dennis Hofheinz and Victor Shoup. GNUC: A new universal composability framework. IACR ePrint 2011/303, June 2011.

[HU05] Dennis Hofheinz and Dominique Unruh. Comparing two notions of simulatability. In Joe Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 86–103. Springer-Verlag, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05comparing.html`.

[HU06] Dennis Hofheinz and Dominique Unruh. Simulatable security and polynomially bounded concurrent composition. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '06*, pages 169–182. IEEE Computer Society, 2006. Full version online available at `http://eprint.iacr.org/2006/130.ps`.

[JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proc. 4th ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 61–70. ACM, 2005.

[Küs06] Ralf Küsters. Simulation-based security with inexhaustible interactive Turing machines. In *CSFW 2006, Computer Security Foundations Workshop*, pages 309–320. IEEE Computer Society, 2006. Long version available as IACR eprint 2006/151.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer Society, 2003. Online available at `http://eprint.iacr.org/2003/141`.

[MCC08] Andrew C. Myers, Michael Clarkson, and Stephen Chong. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368. IEEE, May 2008.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '01*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, Massachusetts Institute of Technology, February 1996. Online available at `http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.ps`.

[Unr06] Dominique Unruh. *Protokollkomposition und Komplexität.* PhD thesis, Universität Karlsruhe (TH), 2006. In German, online available at `http://www.infsec.cs.uni-sb.de/~unruh/publications/unruh06protokollkomposition.html`.