Handling Expected Polynomial-Time Strategies in Simulation-Based Security Proofs*

Jonathan Katz† Yehuda Lindell‡

July 20, 2006

Abstract

The standard class of adversaries considered in cryptography is that of *strict* polynomial-time probabilistic machines. However, *expected* polynomial-time machines are often also considered. For example, there are many zero-knowledge protocols for which the only known simulation techniques run in expected (and not strict) polynomial time. In addition, it has been shown that expected polynomial-time simulation is *essential* for achieving constant-round black-box zero-knowledge protocols. This reliance on expected polynomial-time simulation introduces a number of conceptual and technical difficulties. In this paper, we develop techniques for dealing with expected polynomial-time adversaries in simulation-based security proofs.

Keywords: expected polynomial-time, black-box simulation, secure multiparty computation, zero-knowledge

1 Introduction

1.1 Background

Informally speaking, the simulation paradigm (introduced in [15]) states that a protocol is secure if the view of any adversary in a real protocol execution can be generated solely from the information the adversary legitimately possesses (i.e., its input and output). This is demonstrated by presenting a *simulator* that is given only the input and output of the adversarial (or "corrupted") party or parties, and generates a view that is indistinguishable from the view of the adversary in a real protocol execution. The implication is that the adversary learns nothing from the protocol execution, since it could anyway generate everything that it sees in such an execution by itself.

The simulation paradigm can be instantiated in a number of different ways, where the differences that we refer to here relate to the complexity of the adversary and the complexity of the simulator. The most straightforward way of instantiating this paradigm is to require that for every (strict) polynomial-time adversary there exists a (strict) polynomial-time simulator that generates the required view. However, in many cases it is not known how to construct such simulators. Often, it is

*An extended abstract of this work appeared in the *2nd Theory of Cryptography Conference* (TCC), 2005. This research was supported in part by Grant No. 2004240 from the United States-Israel Binational Science Foundation (BSF), Jerusalem, Israel.

†Department of Computer Science, University of Maryland. email: jkatz@cs.umd.edu.

‡Department of Computer Science, Bar-Ilan University. email: lindell@cs.biu.ac.il. Some of this work was carried out while the author was at IBM T.J.Watson.

shown instead that for every *strict* polynomial-time adversary there exists an *expected* polynomial-time simulator that generates the required view. In certain contexts a relaxation of this sort is actually *necessary* (see below); unfortunately, this reliance on expected polynomial-time simulation is problematic for the following reasons:

1. Conceptual considerations: The intuition behind the simulation paradigm is that anything an adversary can learn from its interaction in a real protocol execution, it could also learn given only its input and output. This (seemingly) follows because the adversary can run the simulator itself and thus obtain a view that is indistinguishable from its view in a real execution. However, if an adversary runs in strict polynomial time while the simulator runs in expected polynomial time, then the adversary *cannot* run the simulator. One immediate solution to this problem is to consider adversaries running in expected polynomial time as well. However, as we will see in Section 1.2, doing so is problematic for other reasons. The fact that the adversary and simulator are of different complexities is somewhat disturbing and contrary to the philosophy of the simulation paradigm. However, this discrepancy also has technical ramifications beyond the mere conceptual ones that we have mentioned. We describe a major one next.

2. Technical considerations (composition): Consider the case where a protocol π calls a sub-routine to compute some function f, and is proven secure in a setting where a trusted entity is used by the parties to compute f.[1] Let ρ be a protocol that securely computes f. We would like to claim that the composed protocol π^ρ (i.e., in which f is computed within π by having the parties run protocol ρ) is also secure. The typical way of proving that π^ρ is secure [5] is to incorporate the *simulator* that is guaranteed to exist for ρ into an adversary that attacks π (in the setting where a trusted party is used to compute f). The security of π can then be invoked, and the overall security of π^ρ is thus obtained. The important point to notice here is that this approach to proving security *fails* when security of π and ρ is proven by demonstrating the existence of an expected polynomial-time simulator for every strict polynomial-time adversary. The reason for this failure is that π is proven secure only for *strict* polynomial-time adversaries, while the adversary (attacking π) obtained by incorporating the simulator for ρ will run in *expected* polynomial time.

We remark that – seemingly due, at least in part, to this difficulty – all previous simulation-based composition theorems of which we are aware (e.g., [18, 5, 6]) deal only with the case of protocols proven secure via *strict* polynomial-time simulation. We also remark that, as with the aforementioned conceptual considerations, this problem would be solved if all protocols were proven secure relative to adversaries that run in expected polynomial time. As we have mentioned, this results in other difficulties, and anyway may not be necessary for obtaining the desired composition.

1.2 Potential Ways of Resolving the Difficulties

There are at least two possible ways of dealing with the difficulties raised above:

1. Require simulators to be "no more powerful" than adversaries: One way of resolving the above difficulties is to require simulators and adversaries to lie in the same complexity class. This approach addresses not only the conceptual difficulty raised above, but also the issue of composition. (This is due to the fact that once the simulator lies in the same class as the adversary, the general strategy for proving secure composition, as sketched above, is applicable.) Here, there are two natural choices: **(a)** require both the adversary and the simulator to run in STRICT polynomial time, or **(b)** allow both the adversary and the simulator to run in EXPECTED polynomial-time.

[1] More formally, and using more technical terminology, here we consider a protocol π that has been proven secure in the f-hybrid model [5, 6]. See Section 4.1 for formal definitions.

Some limitations of the first choice (requiring STRICT polynomial time for both the adversary and the simulator) were demonstrated in [3], where it was shown that there do not exist *constant-round* zero-knowledge protocols[2] with black-box simulators running in strict polynomial time. Constant-round zero-knowledge protocols with *non black-box* simulation strategies running in strict polynomial time are, however, known to exist [1, 2, 3].

Before considering the second choice, where both simulators and adversaries run in EXPECTED polynomial time, we briefly address the issue of how to define expected polynomial-time adversaries for *interactive* settings (this is treated more formally in Section 2). Loosely speaking, Feige [8] defined that an adversary $\mathcal{A}$ attacking a protocol π runs in expected polynomial time if it runs in expected polynomial time when interacting with the *honest parties running* π. (We refer to this notion as expected polynomial time with respect to the protocol π.) Under this definition, $\mathcal{A}$ may possibly run for a much longer amount of time when interacting with other machines.[3] The justification for this definition is that the goal of an adversary is to attack honest parties; therefore, any strategy that is "efficient" when interacting with honest parties should be considered "feasible." A more stringent definition, advocated by Goldreich [10], requires the adversary to run in expected polynomial time when interacting with *any* interactive machine. (We call this notion expected polynomial time in any interaction.) Clearly, any machine that is expected polynomial time in any interaction is also expected polynomial time with respect to any protocol π; it is also not hard to see that the converse is not true. Thus, the second notion defines a strictly smaller class of adversaries than the first. The justification for this latter definition is that it seems more intuitively appealing as a "complexity class" of machines.

We now discuss the implementation of the simulation paradigm in which both the adversary and the simulator run in EXPECTED polynomial time. Feige [8] showed that known simulators for computational zero-knowledge protocols fail when considering adversaries that run in expected polynomial time *with respect to the honest prover*. In contrast, it was shown in [20, Appendix A.1] that the Feige-Shamir zero-knowledge argument[4] system [8, 9] remains both zero-knowledge and an argument of knowledge even when the adversarial party runs in expected polynomial time *in any interaction*. (We stress that this positive result of [20] does *not* hold for adversaries that run in expected polynomial time with respect to the honest prover.) It was further demonstrated in [20, Appendix A.2] that the known simulator for the Goldreich-Kahan zero-knowledge proof system [12] does *not* work for adversaries running in expected polynomial time in any interaction, and so likewise for expected polynomial time with respect to the protocol (for the sake of self containment, we duplicate this negative result of [20] in Appendix A). In fact, prior to our results (see Section 1.3), there was no known *proof system* for $\mathcal{NP}$ that was proven zero-knowledge for adversaries that run in expected polynomial time (even under the restricted definition of [10]). We conclude that allowing the adversary to run in EXPECTED polynomial time is problematic because, prior to our results, in many cases *it was simply not known how to construct simulators for such adversaries*. (This is in contrast to the case where both the adversary and the simulator run in strict polynomial time which, as we have mentioned, suffers from certain *inherent* limitations.)

The situation for zero-knowledge protocols (prior to our results) is summarized in Table 1. We stress that this refers only to *computational* zero-knowledge; *perfect* zero-knowledge proofs and

[2] In this paper, whenever we refer to zero-knowledge protocols we mean those having negligible soundness error.

[3] As a "silly" example to illustrate what we mean: assume protocol π requires parties to preface their messages with a "1," and consider an adversary that runs in polynomial time when receiving messages beginning with "1" but runs in exponential time when receiving messages beginning with "0." Under Feige's definition, such an adversary is considered to run in expected polynomial time.

[4] Recall that in a proof system soundness holds even for all-powerful provers, whereas in an argument system it is required to hold only for polynomial-time provers.

arguments appear to remain zero-knowledge even when the verifier runs in expected polynomial time (see [8, Sects. 3.3, 3.4]). A brief explanation for this is given in Section 1.3.

Type of verifier	**ZK proofs**	**ZK arguments**
Expected poly-time in any interaction	Unknown	Achieved by [9]
Expected poly-time w.r.t. the honest prover	Unknown	Unknown

Table 1: Prior state of affairs regarding the existence of *computational* zero-knowledge proofs/arguments for expected polynomial-time verifiers.

2. Prove a direct composition theorem for expected polynomial-time simulation: A second and incomparable approach for dealing with the problem of expected polynomial-time simulation addresses the technical issue of protocol composition, but does not deal with the above-mentioned conceptual considerations. In this approach, the aim is to show that if two protocols π and ρ are proven secure in the sense of admitting expected polynomial-time simulation for strict polynomial-time adversaries, then the composed protocol π^ρ is also secure in the same sense (i.e., that there exists an expected polynomial-time simulator for every strict polynomial-time adversary attacking π^ρ).[5] Note that in our earlier discussion regarding "technical considerations" we showed that the current *proof technique* for proving composition fails. This still leaves open the possibility of finding a different proof technique that can be used to show that π^ρ is secure.

An advantage of this approach is that many existing efficient protocols have already been proven secure using expected polynomial-time simulation for strict polynomial-time adversaries. A composition theorem as described above means we can use these protocols as building blocks without reproving their security.

1.3 Our Results

The main focus of this paper is to develop techniques for working with expected polynomial-time adversaries and simulation. We take first steps in this direction and present two incomparable results, corresponding to the two approaches discussed in the previous section.

1. Simulation for expected polynomial-time adversaries. Our first result focuses on achieving expected polynomial-time simulation for expected polynomial-time adversaries. Before describing the result, we illustrate by way of example the central technical problem that arises when attempting to simulate an expected polynomial-time adversary. Consider the abstract case of an execution of a polynomial-time oracle machine A with an expected polynomial-time oracle B (in our eventual application, the oracle machine A will be a black-box simulator and the oracle B will be an adversary), where A and B act as follows:

1. Upon input 1^k, oracle machine A queries its oracle with the message 1^k and receives back a message x. Next, A queries its oracle with x and halts.

2. Machine B receives an input q. If q equals the first k bits of its random tape, denoted r, then B runs for 2^k steps and halts. Otherwise, it replies with r and halts.

[5] More precisely (cf. Section 4.1 for formal definitions), say we take as our notion of protocol security the existence of an appropriate expected polynomial-time simulator for every strict polynomial-time adversary. Then we would like to claim that if π securely computes some functionality g in the f-hybrid model, and ρ securely computes f, then π^ρ securely computes g.

Machine A runs in strict polynomial time. Machine B runs in expected polynomial time because, for any input q of length k, the probability (over the choice of its random tape r) that r is equal to q is 2^{-k} (and thus B runs for 2^k steps with probability 2^{-k}). We may therefore expect that the overall expected number of steps made by both A and B in an execution of $A^B(1^k)$ would also be polynomial. However, the number of steps made by both A and B in an execution of $A^B(1^k)$ is actually always more than 2^k. This is due to the fact that in the execution of $A^B(1^k)$, machine A's second query to B is always the first k bits of B's random tape. We therefore conclude that the "composition" of a polynomial-time oracle machine with an expected polynomial-time oracle does not necessarily yield an expected polynomial-time computation (when counting the steps of both the oracle machine and its oracle). More technically, we call this type of composition of two machines A and B oracle composition. Furthermore, say a class of machines $\mathcal{C}$ is closed under oracle composition if for any oracle machine $A \in \mathcal{C}$ and any machine $B \in \mathcal{C}$, the composed machine A^B is also in $\mathcal{C}$.[6] This property of closure under oracle composition is important for black-box simulations (where A is the simulator and B is the adversary), and holds for the class of strict polynomial-time machines. However, the above example shows that the class of expected polynomial-time machines is *not* closed under oracle composition.

In the setting we consider in this paper, the oracle machine A is a black-box simulator, and the machine B is a real adversary (whose view A is simulating). The central difficulty that arises is due to the fact that the distribution of messages that the adversary sees in a simulated execution may be *very far* from the distribution of messages that it sees when interacting in a real execution with *any* real machine. If the adversary "notices" this difference, it can have a much higher complexity during simulation than in a real execution. Due to this, *the adversary may run in expected polynomial time in real interactions, but in expected* superpolynomial *time in simulated interactions.* This is the main problem we encounter. For example, when the above-described B interacts with a real machine, it almost never receives q such that $q = r$. In contrast, when it interacts with the simulator A this always happens. Thus, B's execution time under "simulation" is *much longer* than its execution time in a real interaction. We stress that this problem is not just hypothetical. Rather, as we have mentioned earlier, many concrete protocols and expected polynomial-time simulators suffer from this exact problem. A good example of this phenomenon (for the protocol of [12], and not a contrived protocol) is demonstrated in Appendix A; there, an expected polynomial-time verifier is described that tries to "detect" that it is in a simulated execution and not a real one. In case it does detect this, it runs for a very long time. The simulator of the zero-knowledge protocol of [12] fails for this verifier because the overall expected running time of the simulation is superpolynomial. Simple solutions to this problem (such as truncating the execution after some polynomial number of steps) do not work; see [3] for some discussion on this.

Ideally, we would like to present conditions under which closure under oracle composition can be achieved for expected polynomial-time machines. If we can then also construct an expected polynomial-time simulator fulfilling these conditions, we would then achieve simulation even when the adversary runs in expected polynomial time. Toward this goal, we prove a theorem that shows how to automatically modify a class of black-box simulators (characterized by a certain property) so that the resulting simulation remains expected polynomial time even if the adversary runs in expected polynomial time. Before describing this property, note that if we have a black-box simulator with the property that its oracle queries are always distributed *identically* to the messages sent in a real protocol (let us call this as a *perfect simulator*), then there would be no

[6] In measuring the time complexity of an oracle machine A, calls to its oracle are counted as a single step. However, when we measure the complexity of the composed machine A^B, the running time includes the steps of both A and B. See Section 2.

problem dealing with expected polynomial-time adversaries. This is because in such a case, if the adversary runs for a very long time in the simulation it would also have to run for a very long time in a real execution (and so it would not run in expected polynomial time). For this reason, known simulators for perfect zero-knowledge proofs and arguments seem not to have any difficulty dealing with expected polynomial-time verifiers.

We are now ready to describe a special property of a black-box simulator that will enable us to claim that simulation still runs in expected polynomial time even if the verifier runs in expected polynomial time. The property is a relaxation of a perfect simulator: instead of requiring that every oracle query is distributed identically to messages sent in a real execution, we require that every oracle query is *indistinguishable* from messages sent in a real execution – where indistinguishability must hold even for slightly superpolynomial distinguishers. More precisely, let $\mathcal{S}$ be a black-box simulator with the following *strong indistinguishability property*:

> For all $\mathcal{A}$, every oracle query that $\mathcal{S}$ makes to its oracle $\mathcal{A}$ is "strongly indistinguishable" from some partial transcript (i.e., truncated transcript of the appropriate length) of a real protocol execution involving $\mathcal{A}$. By "strongly indistinguishable", we mean computationally indistinguishable even for machines running in some superpolynomial time $\alpha(k) = k^{\omega(1)}$.

Let now $\mathcal{A}$ be an expected polynomial-time adversary and let $\mathcal{S}$ be a simulator that fulfills the above property. We show that by truncating $\mathcal{S}^{\mathcal{A}}$ at $O(\alpha(k))$ steps, we obtain a "good" simulator that runs in expected polynomial time. The basic idea is that in $\alpha(k)$ steps the adversary $\mathcal{A}$ does not have time to detect that it is receiving simulated messages instead of real messages. Therefore, its running time in a simulation cannot be much longer than in a real execution (similar to the case of a perfect simulator as discussed above). We thus obtain a restricted form of closure under oracle composition that suffices for our application.

The above result holds even for the stronger class of adversaries running in expected polynomial time with respect to the protocol under consideration. We remark that, assuming $\alpha(k)$ hardness assumptions, all "natural" simulators of which we are aware can be converted in a straightforward way into simulators satisfying the above property (the conversion essentially works by replacing primitives having "standard" polynomial security with primitives that are secure even against adversaries running in time $\alpha(k)$).

Coming back to our initial example with A and B, one can see that the "simulator"/oracle machine A does not satisfy the above property if things are cast appropriately. In particular, say that the "protocol" calls for sending a random message q (of length k) to the "adversary" B. Now consider the deterministic adversary $\hat{B}$ that does the following: upon receiving message q, if $q = 0^k$ then run for 2^k steps and halt; otherwise, output 0^k. Adversary $\hat{B}$ runs in expected polynomial time w.r.t. the protocol (since the input message q it receives is randomly chosen). On the other hand, A does not satisfy the above indistinguishability property (even in the usual sense) relative to $\hat{B}$ since the second message sent by A to $\hat{B}$ is always 0^k (whereas such a message is sent only with probability 2^{-k} in a real execution with $\hat{B}$).

The restricted form of closure under oracle composition that we have stated above allows us to prove the following very informally-stated theorem:

(Informally Stated) Theorem 1.1 (closure theorem): *Let π be a protocol that securely computes some functionality f for strict polynomial-time adversaries, in the sense that for any such adversary there exists an appropriate black-box simulator that runs in expected polynomial time and furthermore satisfies the above strong indistinguishability property. Then, π also securely computes f for adversaries that run in expected polynomial time* (*with respect to the protocol*).

An important corollary of our result is that, under a suitable superpolynomial hardness assumption, there exist computational zero-knowledge *proofs* for all of $\mathcal{NP}$ that remain zero-knowledge (with respect to expected polynomial-time simulation) even if the adversarial verifier runs in expected polynomial time. With reference to Table 1, that is, we show the existence of all types of proofs/arguments for expected polynomial-time verifiers that were previously unknown. We note the following caveat: our simulator for the zero-knowledge proof system is guaranteed to run in expected polynomial time only when given a statement x that is in the language L; see Section 3.3 for more details.[7]

2. A composition theorem for expected polynomial-time simulation. The above result achieves security against adversaries running in expected polynomial time, but only for protocols proven secure using simulators satisfying a particular technical property. Our second result shows a composition theorem for protocols proven secure using *arbitrary* black-box simulation, but guarantees security against strict polynomial-time adversaries only. Specifically, under a superpolynomial hardness assumption, we prove an analogue of the modular (sequential) composition theorem of Canetti [5] for protocols that are proven secure for strict polynomial-time adversaries using expected polynomial-time simulation. Loosely speaking, the composition theorem of [5] states that if a secure protocol π contains *sequential* ideal calls to some functionalities, then it remains secure even when these ideal calls are replaced by *sequential* executions of sub-protocols that securely realize these functionalities. The original result of [5] was previously known to hold only for protocols proven secure via *strict* polynomial-time simulation (in fact, in Appendix B we show that the proof of [5] fails in general for protocols proven secure via expected polynomial-time simulation). In contrast, we prove that under superpolynomial hardness assumptions the theorem of [5] holds even if the component protocols are proven secure using *expected* polynomial-time simulation. We pay a price, however, for achieving our stronger result: our proof requires hardness assumptions, in contrast to the unconditional proof of [5]. We remark that we use superpolynomial hardness assumptions only when constructing and analyzing our simulator; we do not modify the underlying protocols and do not require that they be secure for superpolynomial adversaries. In summary, we have the following theorem:

(Informally Stated) Theorem 1.2 (composition for expected polynomial-time simulation): *Let π be a protocol utilizing "sequential ideal calls" to a functionality f. Say π securely computes a functionality g in the sense that there exists an appropriate expected polynomial-time simulator for every strict polynomial-time adversary. Say ρ is a protocol that securely computes functionality f in the same sense. Then, under the assumption that there exist families of functions that are pseudorandom to all adversaries running in (mildly) superpolynomial time, the composed protocol π^ρ securely computes the functionality g in the same sense.*

A caveat regarding Theorem 1.2 is that our proof holds only if the simulator for ρ runs in expected polynomial time *in any interaction.* (Note that a simulator in the setting of secure computation interacts with a trusted party. It is therefore an interactive machine, and so the different notions of expected polynomial time for interactive machines apply to it as well.)

[7] Standard definitions of zero knowledge require the simulator to generate a distribution that is indistinguishable from the view of the verifier *only* when it receives a statement $x \in L$. The question of what complexity a simulator should be when it is invoked on $x \notin L$ has not been considered. The straightforward approach is to require the simulator to maintain its complexity even when invoked on inputs $x \notin L$. As we describe in Section 3.3, this also has significant advantages regarding applications of zero knowledge. Unfortunately, our simulators are only guaranteed to run in expected polynomial-time for inputs $x \in L$. (We remark that when the simulator runs in *strict* polynomial time on inputs $x \in L$, then its execution can be safely truncated at some fixed polynomial number of steps and so we may simply assume that it runs in strict polynomial time even on inputs $x \notin L$.)

1.4 Techniques

In this section, we describe our techniques at a high level. Our aim is to provide an intuitive explanation as to how we deal with the problem of expected polynomial-time adversarial behavior. Some of this intuition has been described briefly above; we present it in more detail here.

Superpolynomial truncation. As we have mentioned, a problem we encounter with proving Theorem 1.1 is that an expected polynomial-time adversary may sometimes have very long executions. This causes a problem when the probability of obtaining such a long execution is higher during simulation than during a real execution (thereby causing the expected running time of the simulation to be superpolynomial). A first attempt at solving this problem is to simply truncate the execution of the adversary if it exceeds its expected running time by "too much." One can thus obtain an adversary running in *strict* polynomial time and then standard simulation techniques can be applied. The problem with this strategy is that it is unclear when to truncate. There are two natural possibilities:

1. *Truncate when the adversary exceeds $p(k)$ times its expected running time, where $p(\cdot)$ is some polynomial and k is the security parameter:* In this case, the truncated adversary clearly runs in strict polynomial time. However, the distribution generated by the truncated adversary may be noticeably far (i.e., distance $1/p(k)$) from the distribution generated by the original adversary. Therefore, the simulation (applied to the truncated adversary) will no longer be indistinguishable from a real execution of the original adversary.

2. *Truncate when the adversary has run a superpolynomial number of steps:* In this case, the resulting distribution of the truncated adversary will be indistinguishable from the original one. This is due to the fact that if the adversary runs in expected polynomial time, it can only run a superpolynomial number of steps with negligible probability. However, this strategy is problematic because, as described above, the adversary may still run for a "long" time during simulation much more often than it does in a real execution. Therefore, we may still obtain an overall expected superpolynomial running time when simulating for this adversary.

Our solution is to adopt the strategy of superpolynomial truncation, but to ensure that the probability that the adversary runs for a very long time during the simulation is close to the probability that it does so in a real execution. Loosely speaking, we achieve this as follows. Let $\alpha(k)$ be any superpolynomial function, and assume that circuits of size $\alpha(k)$ can distinguish a simulator-generated message from a real message with probability at most $1/\alpha(k)$ (this is a reformulation of the strong indistinguishability property introduced in the previous section). Now, in a real execution an adversary can exceed $\alpha(k)$ steps only with probability at most $\text{poly}(k)/\alpha(k)$; otherwise, it would not run in expected polynomial time. Because of the strong indistinguishability property we may now claim that the same must hold during *simulation*, or else a distinguisher of size $\alpha(k)$ would be able to distinguish simulator messages from real ones (with probability at least $1/\alpha(k)$) by monitoring the running time of the adversary. The reason we need a *superpolynomial* bound $\alpha(k)$ is to make sure that the truncation does not noticeably affect the output distribution. This in turn implies that the distinguisher must be allowed to run $\alpha(k)$ steps (since this is when differences in the adversary's running time might be manifested), and so $\alpha(k)$-hardness assumptions are needed.

Stated less technically than above, we show that if the simulator satisfies the strong indistinguishability property then a real adversary running in expected polynomial time does not have time to "detect" any difference between real and simulated executions. Therefore, its expected running time must be approximately the same in both cases. This can then be used to show that the simulated execution also runs in expected polynomial time.

Pseudo-independent oracle invocations. Our starting point for the proof of Theorem 1.2 is a different one. Recalling the discussion of oracle composition (where an oracle machine A makes calls to a machine B) from the previous section, we can see that one difficulty that arises is that A calls B multiple times, and in each call B uses *dependent* random coins. (Specifically, it uses *identical* random coins in the counterexample we gave.) Conversely, if A were to make a strict polynomial number of calls to an expected polynomial-time machine B, and in each invocation B used *fresh* random coins, then (by linearity of expectation) the total expected running time of A^B would be polynomial. (In fact, this holds even if A makes an *expected* polynomial number of calls to B [4, Problem 22.9].) Unfortunately, if A represents a simulator and B represents an expected polynomial-time adversary then A may not produce the appropriate output distribution if it invokes B using fresh random coins each time (in fact, all known black-box simulators would completely fail if forced to work this way).

We describe at a high level the strategy we use to resolve the above issue. The basic idea is to consider the modified algorithm B' that proceeds as follows: given random tape s and input q, algorithm B' computes $r = F_s(q)$ and then runs B using random tape r and input q. If F is a pseudorandom function, then the distributions on the output of B and B' are indistinguishable. So, intuitively, running the simulator A with access to B' should produce output that is "just as good" as running A with access to B. The key point is that even though the random tape of B' is fixed, the random tape used by B (when called by B') is (pseudo)independently and (pseudo)uniformly distributed each time B is invoked. (Without loss of generality, we may assume that A does not query its oracle on the same input twice.) Thus, even though the random coins of A's oracle B' remain *fixed*, the "effective" coins used by B are freshly generated each time it is invoked. (Our technique is somewhat reminiscent of similar ideas used in [7].)

Unfortunately, the above explanation is overly simplistic. A problem that may arise when constructing B' from B is that B' may no longer run in expected polynomial time (since B' runs B using pseudorandom coins rather than random ones). Indeed, the issue here is exactly analogous to the issue that arose with respect to non-perfect simulators (as discussed earlier). In order to prevent this from happening, we must use a function F that is pseudorandom even against adversaries running in some superpolynomial time $\alpha(k)$. We also use the superpolynomial truncation technique described previously in order to ensure that B' runs in expected polynomial time yet its behavior is not noticeably different.

1.5 Open Questions

In this paper, we present the first techniques for dealing with some difficulties that arise due to the use of expected polynomial-time simulation strategies. We view our results as *first steps* in solving these problems, and not as final solutions. The main weaknesses of our results are:

1. In both results, we rely on superpolynomial hardness assumptions rather than standard ones.

2. In order to prove the closure theorem (Theorem 1.1), we need to assume a black-box simulator with the "strong indistinguishability property."

3. In order to prove the sequential composition theorem (Theorem 1.2), we need to assume that the simulator for the sub-protocol ρ runs in expected polynomial time in any interaction. This result is therefore highly sensitive to the definition of expected polynomial time.

Thus, we still do not have truly satisfactory solutions to the problems that arise due to expected polynomial-time simulation.

2 Defining the Running Time of Probabilistic Machines

The security parameter is denoted by k; for conciseness, we equate the security parameter with the input length. (We therefore consider security for "sufficiently-long inputs.") We denote by $A(x,z,r)$ the output of machine A on input x, auxiliary input z, and random coins r. The running time of A is measured in terms of the length of its input x (where $|x|=k$), and the exact running time of the deterministic computation $A(x,z,r)$ is denoted by $\mathsf{time}_A(A(x,z,r))$. Machine A runs in strict polynomial time if there exists a polynomial $p(\cdot)$ such that for all x,z, and *all* r, it holds that $\mathsf{time}_A(A(x,z,r)) \leq p(|x|)$. A runs in expected polynomial time if there exists a polynomial $p(\cdot)$ such that for all x and z, it holds that $\mathbf{Exp}_r[\mathsf{time}_A(A(x,z,r))] \leq p(|x|)$.

A technical issue. A technical problem that arises when considering expected polynomial-time algorithms is that the expected running time is not *machine independent*. As an example, consider an algorithm which, in some model of computation, runs for 2^k steps with probability 2^{-k}, and runs for 1 step otherwise. In this model, the algorithm runs in expected polynomial time. However, if this algorithm is implemented in a second model of computation which, for sake of argument, has quadratic overhead with respect to the first model, we end up with an algorithm that runs for 2^{2k} steps with probability 2^{-k} and therefore no longer runs in expected polynomial time (this example is due to [10]). To avoid these difficulties, we implicitly fix a model of computation throughout our discussion.

Running time for interactive machines. If A is an interactive Turing machine (ITM), we let $A(x,z,r;\cdot)$ denote the "next message function" of A on inputs x,z, and random coins r. The ITM A runs in strict polynomial time if there exists a polynomial $p(\cdot)$ such that for all x,z,r, and any sequence of messages $\overline{m}$, it holds that $\mathsf{time}_A(A(x,z,r;\overline{m})) \leq p(|x|)$. That is, A replies to any message within $p(|x|)$ steps.

Defining expected polynomial-time ITMs is more complicated, and at least two such definitions have been considered. We first present the definition of Feige [8]. As mentioned in the Introduction, the idea behind this definition is that any adversarial strategy that is efficient when run against the specified target (i.e., the honest parties running the protocol) should be considered feasible. Thus, the running time of an adversary when interacting with an arbitrary ITM (that is not the honest party under attack) is irrelevant. Informally, an ITM A is said to run in expected polynomial time with respect to a particular protocol π if there exists a polynomial $p(\cdot)$ such that for all inputs, the expected running time of A when interacting with *honest parties running* π is at most $p(|x|)$. (The expectation here is taken over the random coins of both A and the honest parties.) More formally, let $\mathsf{time}_A(\langle A(x,z_A,r), B(y,z_B,s)\rangle)$ denote the exact running time of A with input x, auxiliary input z_A, and random coins r, when interacting with B holding input y, auxiliary input z_B, and random coins s. Then:

Definition 1 *An ITM A runs in* expected polynomial time with respect to an ITM B *if there exists a polynomial $p(\cdot)$ such that for all x,y with $|x|=|y|$ and all auxiliary inputs $z_A, z_B \in \{0,1\}^*$, the following holds:*

$$\mathbf{Exp}_{r,s}\left[\mathsf{time}_A(\langle A(x,z_A,r), B(y,z_B,s)\rangle)\right] \leq p(|x|).$$

Let $\pi = (P_1,P_2)$ be a two-party protocol. Then an adversary A corrupting P_1 (resp., P_2) runs in expected polynomial time with respect to π *if it runs in expected polynomial time with respect to P_2 (resp., P_1). If f is a (probabilistic) function, then A runs in* expected polynomial time with respect to f *if it runs in expected polynomial time with respect to any ITM computing f.*

The above definition relates to two-party protocols. The extension to the multiparty case is straightforward.

A definition of the above sort makes sense in a cryptographic context, but is arguably a strange way of defining a "complexity class" since, as we have mentioned earlier, the fact that an adversary A runs in expected polynomial time with respect to a protocol π implies nothing about its running time when it interacts with other machines. An alternative approach advocated by Goldreich [10] therefore states that an ITM runs in expected polynomial time if there exists a polynomial $p(\cdot)$ such that for all inputs, the expected running time of A when interacting with *any* (even all powerful) ITM B is at most $p(|x|)$. Here, the expectation is taken over the random coins of A only, because we can assume without loss of generality that B is deterministic. In such a case, we say that A runs in expected polynomial time in any interaction. More formally:

Definition 2 *An ITM A runs in* expected polynomial time in any interaction *if for every ITM B it holds that A runs in expected polynomial time with respect to B (as defined in Definition 1).*

It is immediate that if an ITM A runs in expected polynomial time in any interaction, then A also runs in expected polynomial time with respect to any protocol π. Furthermore, it is not difficult to show a protocol π for which the class of adversaries running in expected polynomial time with respect to π is *strictly larger* than the class of adversaries running in expected polynomial time in any interaction.

Running time for oracle machines. The running time of an oracle machine A (whether interactive or not) is defined along the lines of what has already been discussed. However, we will distinguish between the running time of A itself (counting A's calls to its oracle as a single step) and the running time of the composed machine A^B for some particular machine B. In more detail, let A be an oracle machine with oracle access to an ITM B. It will always be the case, and we will henceforth implicitly require, that A and B hold inputs of the same length (i.e., it is always the case that A and B use the same value for the security parameter k). In the execution of A with B, denoted by $A^{B(y,z_B,s;\cdot)}(x,z_A,r)$, machine A receives input x, auxiliary-input z_A, and random tape r, and provides queries of the form $\overline{m}$ to its oracle which are answered as $B(y,z_B,s;\overline{m})$. We distinguish between two notions of running time for the composed machine A^B:

1. $\mathsf{time}_A(A^{B(y,z_B,s;\cdot)}(x,z_A,r))$ denotes the exact running time of A on input x, auxiliary-input z_A, and random tape r when interacting with the oracle $B(y,z_B,s;\cdot)$, *counting calls to B as a single step* (i.e., we only "count" the steps taken by A).

2. $\mathsf{time}_{A+B}(A^{B(y,z_B,s;\cdot)}(x,z_A,r))$ denotes the exact running time of *both* A and B in the analogous execution. *Here, the steps taken by B to answer A's queries are also counted.*

Given the above, we can define expected polynomial-time oracle machines. (We provide definitions for the case that A^B is a stand-alone machine, but the definitions can be extended exactly as discussed earlier when A^B is an ITM.) We say that the oracle machine A runs in expected polynomial time if there exists a polynomial $p(\cdot)$ such that for every (even all powerful) machine B, all inputs x, and every auxiliary input z, $\mathbf{Exp}_r[\mathsf{time}_A(A^B(x,z,r))] \le p(|x|)$. On the other hand, the composed machine A^B runs in expected polynomial time if[8] there exists a polynomial $p(\cdot)$ such that for all inputs x and y with $|x| = |y|$, and all auxiliary inputs z_A and z_B, it holds that

$$\mathbf{Exp}_{r,s}[\mathsf{time}_{A+B}(A^{B(y,z_B,s;\cdot)}(x,z_A,r))] \le p(|x|).$$

[8]Since A^B is just a regular Turing machine, this definition is exactly the one given earlier (note that r,s are exactly the random coins of the composed machine A^B). We repeat the definition for convenience.

For any strict polynomial-time B, if A runs in expected polynomial time then so does the composed machine A^B. (This assumes that A and B use the same value of the security parameter k, as we indeed require.) We stress, however, that this *does not necessarily hold* when B runs in expected polynomial time (under either definition considered earlier).

3 Simulation for Expected Polynomial-Time Adversaries

In this section, we show how protocols proven secure against *strict* polynomial-time adversaries using a certain class of black-box simulation can in fact be proven secure against *expected* polynomial-time adversaries as well. That is, assuming that the original simulator for the protocol runs in expected polynomial time for any *strict* polynomial-time adversary, we obtain a simulator that runs in expected polynomial time even for adversaries running in *expected* polynomial time. The results of this section hold for the stronger class of adversaries running in expected polynomial time with respect to the protocol, but we obtain a simulator guaranteed to run in expected polynomial time only with respect to the ideal functionality[9] in question (even if the adversary runs in expected polynomial time in any interaction).

3.1 Preliminaries

As we have mentioned in the Introduction, the results of this section hold for a certain class of black-box simulators. We begin with a high-level review of secure computation, and then define the class of simulators we consider. For the sake of simplicity, we present the results here for the case of two-party protocols. The natural extension to the multiparty case also holds.

Secure two-party computation. We provide a very brief and informal overview of the definition of security for two-party computation; for more details, see [5, 11]. In the setting of two-party computation, two parties wish to jointly compute a (possibly probabilistic) functionality $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, upon respective inputs x and y, the parties wish to compute $f(x,y)$ so that party P_1 receives $f_1(x,y)$ and party P_2 receives $f_2(x,y)$. Furthermore, the parties wish to ensure that nothing more than the output is revealed and that the function is correctly computed, even if one of the parties behaves adversarially. These requirements (and others) are formalized by comparing a real protocol execution to an ideal execution involving a trusted party (an "ideal functionality"). In an ideal execution with f, the parties send their inputs x and y to a trusted party who computes $f(x,y)$ and sends $f_1(x,y)$ to P_1 and $f_2(x,y)$ to P_2. The adversary who controls one of the parties can choose to send any input it wishes to the trusted party, while the honest party always sends its specified input.[10] In a real execution of a protocol π, the parties P_1 and P_2 run π, where one of the parties may be corrupted and thus be under the complete control of the adversary $\mathcal{A}$. (We always assume the adversary *statically* corrupts one of the two parties.) Informally, a protocol π is secure if for every real-model adversary $\mathcal{A}$ interacting with an honest party running π, there exists an ideal-model adversary Sim interacting with the trusted party computing f, such that the output of $\mathcal{A}$ and the honest party in the real model is computationally indistinguishable from the output of Sim and the honest party in the ideal model.

[9] The simulator for a secure protocol is both an oracle machine as well as an interactive Turing machine that interacts with a trusted party computing an ideal functionality (see the following section). Thus, we must also explicitly state whether the simulator runs in expected polynomial time in any interaction, or only with respect to the ideal functionality under consideration.

[10] The adversary also has control over the delivery of the output from the trusted party to the honest party. Therefore, fairness and output delivery are not guaranteed.

Notation. Let $\pi = (P_1, P_2)$ be a two-party protocol and let f be a two-input functionality. We denote by $\text{REAL}_{\pi,\mathcal{A}(z)}(x, y)$ the output of a *real execution of* π, where party P_1 has input x, party P_2 has input y, and the adversary $\mathcal{A}$ has input z in addition to the input of the corrupted party (i.e., x if P_1 is corrupted and y if P_2 is corrupted). The output of an execution is defined to be the output of the honest party (which is simply the output dictated by the protocol) along with the output of the adversary (which, without loss of generality, is its view). Likewise, we denote by $\text{IDEAL}_{f,\mathsf{Sim}(z)}(x, y)$ the output of an *ideal execution with* f where the respective inputs are as noted.

When considering black-box simulation, the ideal-world adversary Sim will take the form $\mathcal{S}^{\mathcal{A}}$ where $\mathcal{S}$ is called the "black-box simulator." Here, Sim chooses a random tape r for $\mathcal{A}$ and then runs $\mathcal{S}^{\mathcal{A}(z,r)}$. The black-box simulator $\mathcal{S}$ is given the input of the corrupted party and oracle access to $\mathcal{A}(z, r)$, but is *not* given the auxiliary input z nor the randomness r used by $\mathcal{A}$. For visual convenience, we explicitly provide $\mathcal{S}$ with input 1^k (recall $|x| = |y| = k$, and running times of all parties are measured in terms of k). The black-box simulator $\mathcal{S}$ is both an oracle machine as well as an ITM (since it interacts with the trusted party computing the functionality f in the ideal model). In the previous section we have already defined what it means for $\mathcal{S}$ to run in expected polynomial time. To be explicit, however, we repeat some of that discussion here.

- We denote by $\mathsf{time}_{\mathcal{S}}(\text{IDEAL}_{f(r_f),\mathcal{S}^{\mathcal{A}(z)}(1^k,s)}(x, y))$ the running time of $\mathcal{S}$ (not counting the steps of $\mathcal{A}$) when $\mathcal{S}$ has random tape s and the trusted party computing f uses random tape r_f. For the sake of clarity, we will use shorthand and denote the expected running time of $\mathcal{S}$ in this case by $\mathbf{Exp}_{s,r_f}[\mathsf{time}_{\mathcal{S}}(\text{IDEAL})]$. (Following our conventions for oracle machines, we require that $\mathcal{S}$ runs in expected polynomial time even when $\mathcal{A}$ is computationally unbounded. Therefore, we do not make the random coins of $\mathcal{A}$ explicit when defining the running time in this case, and the expectation of $\mathcal{S}$'s running time is not taken over $\mathcal{A}$'s coins r.) The fact that we include f's coins r_f is due to the fact that we consider simulators $\mathcal{S}$ that may run in expected polynomial time with respect to f. (Note, however, that even when f is a deterministic functionality it may still be the case that $\mathcal{S}$ runs in expected polynomial time with respect to f, but does not run in expected polynomial time in any interaction. This can occur if, for example, it runs for a very long time when it receives a value that is not in the range of f.)

- We denote by $\mathsf{time}_{\mathcal{S}+\mathcal{A}}(\text{IDEAL}_{f(r_f),\mathcal{S}^{\mathcal{A}(z,r)}(1^k,s)}(x, y))$ the running time of the composed machine $\mathcal{S}^{\mathcal{A}}$ (i.e., including $\mathcal{A}$'s steps) when the inputs are as indicated. As above, for the sake of clarity, we write $\mathbf{Exp}_{r,s,r_f}[\mathsf{time}_{\mathcal{S}+\mathcal{A}}(\text{IDEAL})]$ as shorthand for the expectation of this running time. (Note that here the coins r of $\mathcal{A}$ are included.)

  Since $\mathcal{S}^{\mathcal{A}}$ is the ideal-world adversary whose existence provides the intuitive security guarantee we are after, we will require in our definition of security (below) that the composed machine $\mathcal{S}^{\mathcal{A}}$ run in expected polynomial time (i.e., it is not enough that $\mathcal{S}$ runs in expected polynomial time). Of course, when $\mathcal{A}$ runs in strict polynomial time the two requirements are equivalent.

We are now ready to present the definition of security.

Definition 3 *Let f and π be as above. Protocol π is said to* securely compute f for strict polynomial-time adversaries with a black-box simulator that runs in expected polynomial time with respect to f *if there exists an interactive oracle machine (black-box simulator) $\mathcal{S}$ that runs in expected polynomial time with respect to f, such that for every strict polynomial-time real-model adversary $\mathcal{A}$, every non-uniform polynomial-time distinguisher D, every polynomial $p(\cdot)$, all sufficiently-long inputs x*

and y such that $|x| = |y|$, and all $z \in \{0,1\}^*$,

$$\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}(z)}(1^k)}(x,y)) = 1] - \Pr[D(\text{REAL}_{\pi,\mathcal{A}(z)}(x,y)) = 1]\right| < \frac{1}{p(|x|)}.$$

Protocol π is said to securely compute f for expected polynomial-time adversaries with a black-box simulator that runs in expected polynomial time with respect to f *if the above holds even for $\mathcal{A}$ which run in expected polynomial time with respect to π and, furthermore, if the composed machine $\mathcal{S}^{\mathcal{A}}$ runs in expected polynomial time with respect to f even for such $\mathcal{A}$.*

Of course, one could modify the second part of the definition for the case of adversaries/simulators running in expected polynomial time in any interaction.

Strong black-box simulation. We now define a stronger notion of simulation which, informally, requires not only that the final output of $\text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}}}$ be indistinguishable from $\text{REAL}_{\pi,\mathcal{A}}$, but also that each *partial* (truncated) transcript generated during the simulation is indistinguishable from a partial transcript of the same length in a real execution of the protocol. Furthermore, we require indistinguishability to hold in a "strong" sense even against algorithms running in some slightly superpolynomial time.

All protocols from the literature (proven secure for strict polynomial-time adversaries in the sense of the above definition) of which we are aware seem naturally to satisfy the notion of strong black-box simulation under an appropriate superpolynomial hardness assumption. On the other hand, it is easy to construct "unnatural" counterexamples that fail to satisfy this notion.

Let π be a protocol that securely computes some functionality f for strict polynomial-time adversaries in the sense of the above definition. Let $\mathcal{S}$ be a black-box simulator for π (as required by the above definition), and let $\mathcal{A}$ be an adversary. If $\mathcal{A}$ sends its messages in the odd rounds (i.e., $\mathcal{A}$ sends the first message of the protocol), then each query query made by $\mathcal{S}$ to $\mathcal{A}$ has the form $\mathsf{query} = (m_1, \ldots, m_j)$, where m_i represents a $(2i)^{\text{th}}$-round message sent by the uncorrupted party to $\mathcal{A}$ (we do not include the responses of $\mathcal{A}$ in query since these are redundant given $\mathcal{A}$ and its inputs [including its random coins]). In this case we say j is the *number of messages* in query. We also allow the possibility that $\mathsf{query} = \epsilon$ (i.e., $j = 0$), in which case query represents $\mathcal{S}$'s query for the initial message sent by $\mathcal{A}$. The case when $\mathcal{A}$ sends its messages in even rounds (i.e., the uncorrupted party sends the first message of the protocol) is handled in an analogous manner.

Define the following distributions:

1. $\text{SIM}_{f,\mathcal{S}^{\mathcal{A}}}(x,y,z,r,i)$ is defined by the following experiment: choose random tapes s, r_f and run $\text{IDEAL}_{f(r_f),\mathcal{S}^{\mathcal{A}(z,r)}(1^k,s)}(x,y)$. Let query_i be the i^{th} oracle query made by $\mathcal{S}$ to $\mathcal{A}$; if no such query is made (i.e., if $\mathcal{S}$ makes fewer than i queries), set $\mathsf{query}_i = \perp$. Output query_i.

2. $\text{REAL}_{\pi,\mathcal{A}}(x,y,z,r,i)$ is defined by the following experiment: choose random s' and then run $\text{REAL}_{\pi,\mathcal{A}(z)}(x,y)$ with the honest party using random tape s' and $\mathcal{A}$ using random tape r. Let $\overline{T}$ be the vector of messages sent by the honest party to $\mathcal{A}$ in this execution, and let $\overline{T}_j$ denote the first j messages in $\overline{T}$.

   Next, run the experiment $\text{SIM}_{f,\mathcal{S}^{\mathcal{A}}}(x,y,z,r,i)$ above (choosing fresh coins s, r_f) and obtain query_i. If $\mathsf{query}_i = \perp$, then output $\perp$. Otherwise, let j denote the number of messages in query_i, and output $\overline{T}_j$.

The SIM experiment is run in the second case in order to decide the *length* of the partial transcript to output. That is, we wish to compare the distribution of query_i to a partial transcript (of a real execution) *of the same length.* This length is obtained from the invocation of SIM. Note that $\Pr[\mathsf{query}_i \neq \perp]$ is exactly the same in both the SIM and REAL experiments, since the event is due in each case to the outcome of the $\mathrm{SIM}_{f,\mathcal{S}^\mathcal{A}}(x,y,z,r,i)$ experiment. Furthermore, this probability is exactly the probability that the output of the experiment is not $\perp$ (because the output of each experiment is $\perp$ if and only if $\mathsf{query}_i = \perp$).

For any distinguisher D, define $\Delta_D(x,y,z,r,i)$ as follows: If $\Pr[\mathsf{query}_i \neq \perp] \neq 0$, then:

$$\Delta_D(x,y,z,r,i) \stackrel{\text{def}}{=} \Big| \Pr[D(\mathrm{SIM}_{f,\mathcal{S}^\mathcal{A}}(x,y,z,r,i)) = 1 \mid \mathsf{query}_i \neq \perp] - \Pr[D(\mathrm{REAL}_{\pi,\mathcal{A}}(x,y,z,r,i)) = 1 \mid \mathsf{query}_i \neq \perp] \Big| \tag{1}$$

and if $\Pr[\mathsf{query}_i \neq \perp] = 0$, then $\Delta_D(x,y,z,r,i) = 0$. (If $\Pr[\mathsf{query}_i \neq \perp] = 0$, then both SIM and REAL always output $\perp$. We define Δ_D as we do because when $\Pr[\mathsf{query}_i \neq \perp] = 0$, the conditional probability in Eq. (1) is undefined.)

We are now ready to present the definition of strong simulation. Informally, the definition requires that when $\mathsf{query}_i \neq \perp$, it holds that the i^{th} query in the SIM experiment is strongly indistinguishable from a partial transcript of the same length in a real execution.

Definition 4 (α-strong black-box simulation): *Let π be a two-party protocol that securely computes some functionality f in the sense of Definition 3, and let $\mathcal{S}$ be a black-box simulator for π (as required by that definition). We say that $\mathcal{S}$ is an* α-strong black-box simulator for π *(and say that π* securely computes f under α-strong black-box simulation*) if for every adversary running in time at most $\alpha(k)$, every non-uniform algorithm D running in time at most $\alpha(k)$, all sufficiently large x and y, all $z, r \in \{0,1\}^*$, and all $i \in \mathbb{N}$,*

$$\Delta_D(x,y,z,r,i) < \frac{1}{\alpha(k)}.$$

A consequence. As we have mentioned, $\Pr[\mathsf{query}_i \neq \perp]$ is the same in both the SIM and REAL experiments. Assuming the above definition holds, it therefore follows that:

$$\begin{aligned}\big|\Pr[D(\mathrm{SIM}_{f,\mathcal{S}^\mathcal{A}}(x,y,z,r,i)) = 1 \ \wedge\ \mathsf{query}_i \neq \perp] - \Pr[D(\mathrm{REAL}_{\pi,\mathcal{A}}(x,y,z,r,i)) = 1 \ \wedge\ \mathsf{query}_i \neq \perp]\big| \\ = \Pr[\mathsf{query}_i \neq \perp] \cdot \Delta_D(x,y,z,r,i) \\ < \frac{\Pr[\mathsf{query}_i \neq \perp]}{\alpha(k)}\end{aligned}$$

for large enough x, y. (The above holds even if $\Pr[\mathsf{query}_i \neq \perp] = 0$.) This consequence of Definition 4 will be used in our proof below.

Extended black-box simulation. Finally, we introduce a generalization of black-box simulation in which the black-box simulator $\mathcal{S}$ is allowed to *truncate* its oracle $\mathcal{A}$ if $\mathcal{A}$ exceeds some (poly-time computable) number of steps α. Formally, we can define each oracle query to be a pair (α, q); if $\mathcal{A}$ responds to q within α steps then $\mathcal{S}$ is given the response, otherwise $\mathcal{S}$ is given $\perp$. (We will ignore this formalism from here on.) We call such a simulator extended black-box. Note that standard black-box simulators cannot perform such truncation since they are oblivious to how many steps their oracle uses in response to a query. However, requiring α to be polynomial-time computable ensures that any extended black-box simulator can be implemented by a non black-box simulator. We remark that when computing $\mathsf{time}_\mathcal{S}(\mathcal{S}^\mathcal{A})$, oracle calls are still considered a single step (even if $\mathcal{S}$ truncates $\mathcal{A}$ after some number of steps). The definition of $\mathsf{time}_{\mathcal{S}+\mathcal{A}}(\mathcal{S}^\mathcal{A})$ remains unchanged.

3.2 Simulation for Expected Polynomial-Time Adversaries

We are now ready to show how (and under what assumptions) it is possible to convert a simulation strategy that works for *strict* polynomial-time adversaries into a simulation strategy that works for *expected* polynomial-time adversaries.

Theorem 5 *Let $\alpha(k) = k^{\omega(1)}$ be a superpolynomial function that is polynomial-time computable, and let π be a protocol that securely computes some functionality f for* **strict** *polynomial-time adversaries with an α-strong (extended) black-box simulator that runs in expected polynomial time with respect to f. Then π securely computes f for adversaries that run in* **expected** *polynomial time with respect to the protocol. Furthermore, π has an α-strong extended black-box simulator that runs in expected polynomial time with respect to f.*

Proof: The idea behind the proof of this theorem is as follows. Let $\mathcal{S}$ be an α-strong (extended) black-box simulator for π which outputs a "good" simulation for strict polynomial-time adversaries, and let $\mathcal{A}$ be an expected polynomial-time adversary. We first truncate $\mathcal{A}$ at $O(\alpha(k))$ steps to obtain an adversary $\hat{\mathcal{A}}$ which performs "essentially" the same as $\mathcal{A}$. Now, since each query made by the α-strong simulator $\mathcal{S}$ to $\hat{\mathcal{A}}$ is indistinguishable from a partial real transcript *even for non-uniform algorithms running in time $\alpha(k)$*, it follows that $\hat{\mathcal{A}}$ cannot behave noticeably different when receiving an oracle query from $\mathcal{S}$ than when it receives a real partial transcript. In particular, $\hat{\mathcal{A}}$ cannot run "much" longer when it receives an oracle query than it would run when interacting in a real protocol execution, and we know that $\hat{\mathcal{A}}$ runs in expected polynomial time in the latter case. We use this to argue that the composed machine $\mathcal{S}^{\hat{\mathcal{A}}}$ runs in expected polynomial time, and that the resulting transcript is close to the one generated by $\mathcal{S}^{\mathcal{A}}$. We proceed with a formal description of the above steps.

Throughout the proof, we let $|x| = |y| = k$. We assume without loss of generality that $\alpha(k) = O(2^k)$. Let $\mathcal{S}$ be an α-strong black-box simulator for π (for strict polynomial-time adversaries) that is assumed to exist, and define $\hat{\mathcal{A}}$ as the algorithm that behaves exactly as $\mathcal{A}$ except that it outputs $\perp$ if it ever exceeds $\alpha(k)/2$ steps. By definition, $\mathcal{S}$ runs in expected polynomial time with respect to f, and $\mathcal{A}$ runs in expected polynomial time with respect to π.

We construct an extended black-box simulator $\hat{\mathcal{S}}$ that receives oracle access to $\mathcal{A}$ and emulates an execution of $\mathcal{S}^{\hat{\mathcal{A}}}$. That is, $\hat{\mathcal{S}}$ chooses a random tape $s \in \{0,1\}^*$ and invokes $\mathcal{S}$ with random tape s. Then, all oracle queries from $\mathcal{S}$ are forwarded by $\hat{\mathcal{S}}$ to its own oracle $\mathcal{A}$ and the oracle replies are returned to $\mathcal{S}$ unless $\mathcal{A}$ exceeds $\alpha(k)/2$ steps while answering the query, in which case the current execution of $\mathcal{A}$ is aborted and $\hat{\mathcal{S}}$ returns $\perp$. Furthermore, all communication between $\mathcal{S}$ and the trusted party computing f is forwarded unmodified by $\hat{\mathcal{S}}$. Note that $\hat{\mathcal{S}}$ is an *extended* black-box simulator because it truncates its oracle. (It makes no difference whether $\mathcal{S}$ is an extended black-box simulator or not. The only technicality is that if $\mathcal{S}$ is an extended black-box simulator, then if $\mathcal{S}$ requests to truncate a query at some point, so does $\hat{\mathcal{S}}$.) Notice that $\hat{\mathcal{S}}^{\mathcal{A}} = \mathcal{S}^{\hat{\mathcal{A}}}$.

Our goal is to show that $\hat{\mathcal{S}}$ satisfies the second part of Definition 3. We first show that $\hat{\mathcal{S}}^{\mathcal{A}}$ runs in expected polynomial time with respect to f even when $\mathcal{A}$ runs in expected polynomial time with respect to π (and thus also if $\mathcal{A}$ runs in expected polynomial time in any interaction).

Claim 6 *If $\mathcal{A}$ runs in expected polynomial time with respect to π and oracle machine $\mathcal{S}$ runs in expected polynomial time with respect to f, then the composed machine $\hat{\mathcal{S}}^{\mathcal{A}}$ defined above runs in expected polynomial time with respect to f.*

Proof: In the proof below, we assume without loss of generality that $\hat{\mathcal{S}}$ is given input x; i.e., $\mathcal{A}$ corrupts the first party P_1. We show that for any expected polynomial-time adversary $\mathcal{A}$ there exists a polynomial p such that for all sufficiently long x, y, and all z:

$$\mathbf{Exp}_{r,s,r_f}[\mathsf{time}_{\hat{\mathcal{S}}+\mathcal{A}}(\hat{\mathcal{S}}^{\mathcal{A}(z,r)}(x,s))] \le p(k).$$

To prove the claim, first recall that $\hat{\mathcal{S}}^{\mathcal{A}}$ is exactly the same as $\mathcal{S}^{\hat{\mathcal{A}}}$. Now, the running time of $\mathcal{S}^{\hat{\mathcal{A}}}$ consists of two components: the steps taken by $\mathcal{S}$ and the steps taken by $\hat{\mathcal{A}}$ in answering the oracle queries of $\mathcal{S}$. Using linearity of expectation, it suffices to show that the expectation of each of these components is polynomial. Since $\mathcal{S}$ is an expected polynomial-time oracle machine, its expected running time is polynomial when interacting with *any* oracle. It therefore remains only to bound the total number of steps taken by $\hat{\mathcal{A}}$. This is equal to $\mathbf{Exp}_{r,s,r_f}[\sum_{i=1}^{\tau} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)]$, where τ is a random variable denoting the number of oracle queries made by $\mathcal{S}$ to $\hat{\mathcal{A}}$ and $\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)$ is a random variable denoting the running time of $\hat{\mathcal{A}}(z,r)$ in answering the i^{th} query from $\mathcal{S}$. (These random variables may depend on r, s, r_f and the inputs x, y.) We first write

$$\begin{aligned}
\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] &= \sum_{j=1}^{\infty} \Pr_{r,s,r_f}[\tau = j] \cdot \mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{j} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \,\middle|\, \tau = j\right] \\
&= \sum_{j=1}^{\infty} \Pr_{r,s,r_f}[\tau = j] \sum_{i=1}^{j} \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau = j\right] \\
&= \sum_{j=1}^{\infty} \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \mid \tau \ge j\right] \cdot \Pr_{r,s,r_f}[\tau \ge j], \qquad (2)
\end{aligned}$$

where the second equality uses the linearity of expectation, and the third follows by rearranging the probabilities (the full derivation appears in Appendix C). Continuing, and using the fact that $\hat{\mathcal{A}}$ runs for at most $\alpha(k)/2$ steps, we have:

$$\begin{aligned}
\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] &= \sum_{j=1}^{\infty} \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \mid \tau \ge j\right] \cdot \Pr_{r,s,r_f}[\tau \ge j] \\
&= \sum_{j=1}^{\infty} \sum_{t=1}^{\infty} t \cdot \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) = t \mid \tau \ge j\right] \cdot \Pr_{r,s,r_f}[\tau \ge j] \\
&= \sum_{j=1}^{\infty} \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \ge t \mid \tau \ge j\right] \cdot \Pr_{r,s,r_f}[\tau \ge j] \\
&= \sum_{j=1}^{\infty} \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \ge t \wedge \tau \ge j\right]. \qquad (3)
\end{aligned}$$

For any fixed r, the distribution on the message sequence input to $\hat{\mathcal{A}}$ when defining $\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j)$ (namely, the j^{th} query from $\mathcal{S}$) is exactly that given by $\textsc{sim}_{f,\mathcal{S}^{\hat{\mathcal{A}}}}(x,y,z,r,j)$. Let $\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}(j)$ be a random variable denoting the running time of $\hat{\mathcal{A}}(z,r)$ when run on input distributed according to $\textsc{real}_{\pi,\hat{\mathcal{A}}}(x,y,z,r,j)$. (Recall that when $\mathsf{query}_j \neq \perp$ this is a message that $\hat{\mathcal{A}}$ receives in a real execution.) We claim that for large enough x and y, for any z, r, j, and any $t \le \alpha(k)/2$,

$$\left|\Pr_{s',s,r_f}[\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}(j) \ge t \wedge \tau \ge j] - \Pr_{s,r_f}[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \ge t \wedge \tau \ge j]\right| < \frac{\Pr_{s,r_f}[\tau \ge j]}{\alpha(k)}. \qquad (4)$$

(Recall that in $\textsc{real}_{\pi,\hat{\mathcal{A}}}(x,y,z,r,j)$, the random tape s' is that belonging to the honest party running protocol π, while s, r_f are used to run $\textsc{sim}$ and thereby determine whether to output $\perp$.) Noticing that the event "$\tau \geq j$" is exactly the event "$\mathsf{query}_j \neq \perp$," the bound in Eq. (4) holds because otherwise we obtain a non-uniform distinguisher that distinguishes between $\textsc{real}$ and $\textsc{sim}$, in contradiction to the fact that $\mathcal{S}$ is an α-strong black-box simulator (by the consequence of Definition 4 as discussed immediately following that definition). The distinguisher works by counting how long $\hat{\mathcal{A}}$ runs for, using this to distinguish the $\textsc{sim}$ and $\textsc{real}$ distributions. In more detail, given an auxiliary input $z' = (z, r, t)$ with $t \leq \alpha(k)/2$, and the result O of either experiment $\textsc{sim}_{f,\mathcal{S}^{\hat{\mathcal{A}}}}(x,y,z,r,j)$ or $\textsc{real}_{\pi,\hat{\mathcal{A}}}(x,y,z,r,j)$, proceed as follows: if O is $\perp$, output 1. Otherwise, O is a sequence of i messages $\overline{T}_i$. In that case, run $\hat{\mathcal{A}}(z,r)$ on message sequence $\overline{T}_i$, and output 1 if and only if $\hat{\mathcal{A}}$ runs for at least t steps. For large enough k, the total running time of this distinguishing algorithm (including the overhead for maintaining a counter and running $\hat{\mathcal{A}}$) is at most $\alpha(k)$. Therefore, by the discussion following Definition 4, it follows that Eq. (4) holds. We remark that the non-uniformity of Definition 4 is essential here. We also note that this argument is the main conceptual point in the proof of this claim.

Continuing, from Eq. (4) it follows that:

$$\begin{aligned}
&\sum_{t=1}^{\alpha(k)/2} \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \geq t \wedge \tau \geq j\right] \\
&\qquad < \sum_{t=1}^{\alpha(k)/2} \left(\Pr_{r,s',s,r_f}\left[\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}(j) \geq t \wedge \tau \geq j\right] + \frac{\Pr_{r,s,r_f}[\tau \geq j]}{\alpha(k)} \right) \\
&\qquad = \frac{\Pr_{r,s,r_f}[\tau \geq j]}{2} + \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s',s,r_f}\left[\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}(j) \geq t \wedge \tau \geq j\right]. \qquad (5)
\end{aligned}$$

Using the simple observations that:

1. $\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}(j) \leq \mathsf{realtime}_{\hat{\mathcal{A}}(z,r)}$ (where the latter expression refers to the *total* running time of $\hat{\mathcal{A}}(z,r)$ in a real execution), and

2. $\mathsf{realtime}_{\hat{\mathcal{A}}(z,r)} \leq \mathsf{realtime}_{\mathcal{A}(z,r)}$ (because $\hat{\mathcal{A}}$ is truncated whereas $\mathcal{A}$ is not),

and combining Equations (3) and (5), we obtain the following:

$$\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] < \sum_{j=1}^{\infty} \left(\frac{\Pr_{r,s,r_f}[\tau \geq j]}{2} + \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s',s,r_f}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t \wedge \tau \geq j\right] \right).$$

We bound each of the two terms above by a polynomial. First, recall that $\mathcal{S}$ is an expected polynomial-time oracle machine, and thus runs in expected polynomial time for any r. Therefore, the expected value of τ is polynomial (for any r), and we have

$$\sum_{j=1}^{\infty} \frac{\Pr_{r,s,r_f}[\tau \geq j]}{2} = \frac{\mathbf{Exp}_{r,s,r_f}[\tau]}{2} = \frac{q(k)}{2},$$

for some polynomial $q(\cdot)$. Next,

$$\sum_{j=1}^{\infty} \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s',s,r_f}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t \wedge \tau \geq j\right]$$

$$
\begin{aligned}
&= \sum_{j=1}^{\infty}\sum_{t=1}^{\alpha(k)/2} \Pr_{r,s'}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \cdot \Pr_{r,s',s,r_f}\left[\tau \geq j \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \\
&= \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s'}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \cdot \sum_{j=1}^{\infty} \Pr_{r,s',s,r_f}\left[\tau \geq j \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right]. \qquad (6)
\end{aligned}
$$

Fix t, and consider the expression

$$
\sum_{j=1}^{\infty} \Pr_{r,s',s,r_f}\left[\tau \geq j \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] = \mathbf{Exp}_{r,s',s,r_f}\left[\tau \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right].
$$

Recall again that $\mathcal{S}$ is an expected polynomial-time oracle machine and thus runs in expected polynomial time for any r. In particular, it runs in expected polynomial time even when r and s' (the random coins of $\mathcal{A}$ and the honest party) are such that $\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t$. This implies that

$$
\begin{aligned}
\sum_{j=1}^{\infty} \Pr_{r,s',s,r_f}\left[\tau \geq j \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] &= \mathbf{Exp}_{r,s',s,r_f}\left[\tau \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \\
&\leq q(k). \qquad (7)
\end{aligned}
$$

Combining Equations (6) and (7) we obtain:

$$
\begin{aligned}
&\sum_{j=1}^{\infty}\sum_{t=1}^{\alpha(k)/2} \Pr_{r,s',s,r_f}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t \wedge \tau \geq j\right] \\
&= \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s'}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \cdot \sum_{j=1}^{\infty} \Pr_{r,s',s,r_f}\left[\tau \geq j \mid \mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \\
&\leq \sum_{t=1}^{\alpha(k)/2} \Pr_{r,s'}\left[\mathsf{realtime}_{\mathcal{A}(z,r)} \geq t\right] \cdot q(k) \\
&\leq q(k) \cdot \mathbf{Exp}_{r,s'}\left[\mathsf{realtime}_{\mathcal{A}(z,r)}\right],
\end{aligned}
$$

which is polynomial because $\mathcal{A}$ runs in expected polynomial time with respect to π. This completes the proof of Claim 6. □

We have shown that the composed machine $\hat{\mathcal{S}}^{\mathcal{A}}$ runs in expected polynomial time with respect to f. It remains to show that it is an α-strong (extended black-box) simulator for expected polynomial-time adversaries. We first show that it provides a "good" simulation; namely:

Claim 7 *For any $\mathcal{A}$ running in expected polynomial time with respect to π, every non-uniform polynomial-time distinguisher D, every polynomial $p(\cdot)$, all sufficiently-long inputs x and y such that $|x| = |y|$, and all $z \in \{0,1\}^*$,*

$$
\left|\Pr[D(\text{IDEAL}_{f,\hat{\mathcal{S}}^{\mathcal{A}(z)}(1^k)}(x,y)) = 1] - \Pr[D(\text{REAL}_{\pi,\mathcal{A}(z)}(x,y)) = 1]\right| < \frac{1}{p(|x|)}.
$$

Proof: Assume the claim does not hold. Then there exists a non-uniform polynomial-time distinguisher D, an infinite sequence $\{(x_i, y_i, z_i)\}_{i\in\mathbb{N}}$ (with $|x_k| = |y_k| = k$), and a constant $c > 0$ such that for infinitely-many values of k:

$$
\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}^{\hat{\mathcal{A}}(z_k)}(1^k)}(x_k,y_k)) = 1] - \Pr[D(\text{REAL}_{\pi,\mathcal{A}(z_k)}(x_k,y_k)) = 1]\right| \geq \frac{1}{k^c}. \qquad (8)
$$

(We drop the subscripts on x, y, z from now on.) Recall that $\hat{\mathcal{S}}^{\mathcal{A}}$ is identical to $\mathcal{S}^{\hat{\mathcal{A}}}$, and thus Eq. (8) is the negation of the claim (even though the actual claim relates to $\mathcal{S}^{\hat{\mathcal{A}}}$ and not to $\hat{\mathcal{S}}^{\mathcal{A}}$). Let $m(k)$ be the maximum of the expected running times of $\mathcal{A}$ (when interacting with π) and $\mathcal{S}^{\hat{\mathcal{A}}}$; since both run in expected polynomial time, $m(k)$ is polynomial as well. Define $\tilde{\mathcal{A}}$ to be identical to $\mathcal{A}$ except that it halts immediately (with output $\perp$) if it ever exceeds $4m(k)k^c$ steps; note that $\tilde{\mathcal{A}}$ runs in *strict* polynomial time. We have that the statistical difference between $\text{REAL}_{\pi,\mathcal{A}(z)}(x, y)$ and $\text{REAL}_{\pi,\tilde{\mathcal{A}}(z)}(x, y)$ is at most $k^{-c}/4$ for large enough k, and similarly for $\text{IDEAL}_{f,\mathcal{S}^{\hat{\mathcal{A}}(z)}(1^k)}(x, y)$ and $\text{IDEAL}_{f,\mathcal{S}^{\tilde{\mathcal{A}}(z)}(1^k)}(x, y)$. We conclude that

$$\left|\Pr[D(\text{IDEAL}_{f,\mathcal{S}^{\tilde{\mathcal{A}}(z)}(1^k)}(x, y)) = 1] - \Pr[\text{REAL}_{\pi,\tilde{\mathcal{A}}(z)}(x, y)) = 1]\right| \geq \frac{1}{k^c/2}.$$

Since $\tilde{\mathcal{A}}$ runs in strict polynomial time, however, the above contradicts the assumed security of π against strict polynomial-time adversaries. □

It remains to show that $\hat{\mathcal{S}}$ is in fact an α-strong simulator. This follows quite easily from the facts that for any $\mathcal{A}$ we have $\hat{\mathcal{S}}^{\mathcal{A}} = \mathcal{S}^{\hat{\mathcal{A}}}$ (where $\hat{\mathcal{A}}$ is the truncation of $\mathcal{A}$ at $\alpha(k)/2$ steps), and the assumption that $\mathcal{S}$ is an α-strong simulator. This completes the proof of the theorem. ■

3.3 Zero-Knowledge Proofs: A Corollary

Consider the zero-knowledge functionality for a language $L \in \mathcal{NP}$. This function is defined by $f(x, x) = (\lambda, \chi_L(x))$, where $\chi_L(x) = 1$ if and only if $x \in L$ (here λ denotes the empty string). A zero-knowledge protocol π securely realizes f for strict polynomial-time adversaries. Now, for the sake of concreteness, consider the zero-knowledge protocol of Goldreich, Micali, and Wigderson [16]. Assuming the existence of commitment schemes that are hiding for non-uniform algorithms running in time $\alpha(k)$, it is easy to verify that the black-box simulator provided by [16] is α-strong. Applying Theorem 5, we obtain that the protocol of [16] is also secure for adversaries that run in expected polynomial time with respect to the protocol. We thereby obtain the *first* computational zero-knowledge *proof system* that remains zero-knowledge for expected polynomial-time adversaries (with respect to either of the definitions in Section 2).[11] Thus, as a corollary of Theorem 5, we partially resolve the open questions from [8, 20] discussed in the Introduction. The result is only "partial" because we need superpolynomial hardness assumptions. In addition, there is an important caveat regarding this result, which we describe now.

Zero-knowledge simulation and inputs $x \notin L$. In order to describe the caveat regarding the above corollary, we need to discuss a subtle issue regarding zero-knowledge simulation. The issue that we refer to relates to the behavior of the simulator when run on an input $x \notin L$. On the one hand, the definition of zero-knowledge requires *nothing* of the output distribution of the simulator in this case (since a real prover will never execute the protocol when $x \notin L$); indistinguishability from a real execution is only required if $x \in L$. On the other hand, since the simulator is assumed to be an expected (or strict) polynomial-time machine, its running time should be preserved even when run on an input $x \notin L$. This *implicit* requirement regarding the running time is not just for the sake of aesthetics. In many proofs of security, the zero-knowledge simulator is actually run on an input $x \notin L$. For example, consider a "commit-and-prove" protocol where a party commits

[11] These are also the first computational zero-knowledge *arguments* for adversaries that run in expected polynomial time *with respect to the protocol*. Previously, these were known only for adversaries that run in expected polynomial time *in any interaction*; see Table 1.

to a value and then proves some property of that value in zero-knowledge (this is exactly what happens in the proof of [17]). The proof of security (stating that the committed value is not learned) typically works by first replacing the zero-knowledge proof with a simulated one. Next, the commitment is replaced with a commitment to garbage. Since the zero-knowledge proof is already simulated, and so does not relate to the actual committed value, the indistinguishability of this last step follows from the hiding property of the commitment. It is therefore possible to derive that the real protocol is indistinguishable from one where garbage is first committed to, and then a simulated zero-knowledge proof is provided.

The important point to notice here is that when the commitment is "real," the simulator is run on an input $x \in L$. However, when the commitment is "garbage," the simulator is run on an input $x \notin L$. Now, if the simulator does not run in expected polynomial time in the event that it is given input $x \notin L$, the above proof of security fails. Specifically, the hiding property of commitments no longer guarantees anything because the distinguisher (who runs the zero-knowledge simulator) may exceed a polynomial number of steps. We conclude that the scenario of running a simulator on an input $x \notin L$ arises in many contexts, and so it is important that a simulator remains (expected or strict) polynomial time in such a case.

Our simulator is not guaranteed to run in expected polynomial time in case it receives an input $x \notin L$. The reason for this, informally, is that α-strongness may no longer hold in this case; in particular, a distinguisher D may be able to distinguish "real" from "simulated" transcripts just by checking if the statement is in the language. (This is easiest to see for the case when L is an "easy" language; say $L \in \mathcal{P}$.) On the positive side, if the language L is such that inputs $x \in L$ cannot be distinguished from inputs $x \notin L$ by non-uniform machines running in time $\alpha(k)$ with probability better than $1/\alpha(k)$, then the α-strong simulation property once again holds and so the simulator *is* guaranteed to run in expected polynomial time. We conclude that the "commit and prove" subprotocol described above can be used, as long as the commitment scheme is hiding (to within probability $1/\alpha(k)$) even for non-uniform machines running in time $\alpha(k)$.

3.4 Protocol Composition and Other Scenarios

Our result above has been stated for the stand-alone setting of secure computation. However, it actually holds for *any* setting, as long as the black-box simulator is α-strong for that setting. In particular, the result holds also for the setting of protocol composition where many protocol executions are run (and thus the simulator interacts with the trusted party many times).

4 Modular Sequential Composition

Our goal in this section is to prove a modular sequential composition theorem for secure multi-party computation that is analogous to the result of Canetti [5], but which holds even for protocols that have been proven secure using a simulation strategy that runs in *expected* polynomial time (for strict polynomial-time adversaries). The sequential composition theorem of [5] can be informally described as follows. Let π be a multi-party protocol computing a function g, designed in an idealized model in which the parties have access to a trusted party who evaluates functions $f_1, \ldots, f_m$; furthermore, assume that at most one ideal function call is made during any round of π. This model is called the $(f_1, \ldots, f_m)$-hybrid model, denoted $\textsc{hybrid}^{f_1,\ldots,f_m}$, because parties send real messages as part of the protocol π and also interact with a trusted party computing functions $f_1, \ldots, f_m$. Let $\rho_1, \ldots, \rho_m$ be multi-party protocols such that ρ_i computes f_i, and let $\pi^{\rho_1,\ldots,\rho_m}$ denote the "composed protocol" in which each ideal call to f_i is replaced by an invocation of ρ_i (we stress that each executed protocol ρ_i is run to completion before continuing the execution of π). The modular sequential composition theorem then states that if π securely computes g in the $(f_1, \ldots, f_m)$-hybrid model, and if each ρ_i securely computes f_i, then the composed real protocol $\pi^{\rho_1,\ldots,\rho_m}$ securely computes g. The work of [5] only considers the case where π, as well each of the component protocols ρ_i, is proven secure via *strict* polynomial-time simulation (for strict polynomial-time adversaries). In fact, the proof of [5] *fails* for the case when one (or more) of the ρ_i subprotocols is proven secure via expected polynomial-time simulation; a specific counterexample is shown in Appendix B.

In this section, we show that a suitable modification of the approach of [5] can be used to prove an analogous modular composition theorem even when π and each of the component protocols ρ_i is proven secure via expected polynomial-time simulation for strict polynomial-time adversaries. With this change, the composition theorem we prove is analogous to the one shown in [5] for the case of protocols proven secure using *strict* polynomial-time simulation. Our proof holds only when π is proven secure using a *black-box* simulator, and each ρ_i is proven secure using a simulator that runs in expected polynomial time *in any interaction*. We also require the existence of pseudorandom functions that are secure even for adversaries running in time $\alpha(k)$ for some $\alpha(k) = k^{\omega(1)}$. In contrast, the result of [5] holds regardless of the type of simulation used to prove π and the ρ_i secure, and is unconditional.

4.1 Preliminaries

Since we deal here explicitly with n-party protocols, and because of the need to introduce additional notation, some of the discussion here overlaps with that of Section 3.1. Due to the high-level similarity of our proof to the proof of [5], wherever possible we make our notation consistent with that of [5].

A *distribution ensemble* $X = \{X(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ is an infinite sequence of probability distributions, where a distribution $X(k,a)$ is associated with each value of k and a. Two distribution ensembles X, Y are *computationally indistinguishable*, denoted $X \stackrel{c}{\equiv} Y$, if there exists a negligible function μ such that for every non-uniform polynomial-time algorithm D, all a, and all auxiliary information z we have

$$\left|\Pr[D(1^k, a, z, X(k,a)) = 1] - \Pr[D(1^k, a, z, Y(k,a)) = 1]\right| \leq \mu(k).$$

We present a definition of security for n-party protocols computing a (probabilistic) function f in the presence of a non-adaptive adversary. We will work exclusively in the computational (rather than information-theoretic) setting, and therefore do not assume private channels but instead allow the adversary to monitor all communication in the network.

The real-world model. We assume a set of parties $P_1, \ldots, P_n$, where party P_i begins with input $(1^k, x_i)$ and random tape r_i. An adversary $\mathcal{A}$ begins with an input containing the security parameter 1^k, the identities of the corrupted parties I, the inputs of the corrupted parties, and their random tapes. In addition, $\mathcal{A}$ has an *auxiliary input* z and a random tape $r_{\mathcal{A}}$ (since we are allowing non-uniform adversaries, we could assume that $\mathcal{A}$ is deterministic; however, we find it conceptually easier to let $\mathcal{A}$ be probabilistic).

Computation proceeds in rounds, in the standard way. At the end of the computation, all parties locally generate their outputs. Honest parties output what is specified by the protocol, while corrupted parties output a special symbol $\perp$. Denote the output of party P_i by v_i, and let $\vec{v} = (v_1, \ldots, v_n)$. In addition, the adversary outputs an arbitrary function of its view of the computation, where the view consists of the corrupted parties' inputs, the adversary's auxiliary input, and all messages sent and received throughout the computation. It is stressed that the outputs $\vec{v}$ are a function of the protocol π, the adversary $\mathcal{A}$, the set of corrupted parties I, and the values $(k, \vec{x}, z, \vec{r})$, where $\vec{x}$ is the vector of all parties' inputs and $\vec{r}$ consists of all the parties' random tapes, including the random tape $r_{\mathcal{A}}$ of the adversary. Thus, formally, $\vec{v}$ is denoted as a function $\vec{v}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r})$. Likewise, we denote by $\textsc{out}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r})$ the output of $\mathcal{A}$, with auxiliary input z and controlling the parties in I, when running an execution of π with parties having input $\vec{x}$ and random tapes $\vec{r}$, and with security parameter k. Finally, define:

$$\textsc{real}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}) \stackrel{\text{def}}{=} \textsc{out}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}) \circ \vec{v}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}).$$

Let $\textsc{real}_{\pi,\mathcal{A},I}(k, \vec{x}, z)$ denote the probability distribution of $\textsc{real}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r})$ when $\vec{r}$ is randomly chosen, and let $\textsc{real}_{\pi,\mathcal{A},I}$ denote the distribution ensemble $\{\textsc{real}_{\pi,\mathcal{A},I}(k, \vec{x}, z)\}_{k\in\mathbb{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

The ideal process. An adversary $\mathcal{S}$ in the ideal world again begins with input that includes a security parameter, identities and inputs of the corrupted parties, auxiliary input, and a random tape $r_{\mathcal{S}}$. The adversary interacts with a trusted party (computing an ideal functionality) in the standard way and, in particular, obtains an output value from the trusted party for each of the corrupted parties. Honest parties output the value given to them by the trusted party, and corrupted parties output $\perp$. Denote the output of party P_i by v_i and let $\vec{v} = (v_1, \ldots, v_n)$. As above, $\vec{v}$ is a function of f, $\mathcal{S}$, I and the values $(k, \vec{x}, z, \vec{r})$. However, here $\vec{r}$ consists only of the random tape of the adversary and of the trusted party. The adversary outputs an arbitrary function of its view, and we denote this by $\textsc{out}_{f,\mathcal{S},I}(k, \vec{x}, z, \vec{r})$. Define

$$\textsc{ideal}_{f,\mathcal{S},I}(k, \vec{x}, z, \vec{r}) \stackrel{\text{def}}{=} \textsc{out}_{f,\mathcal{S},I}(k, \vec{x}, z, \vec{r}) \circ \vec{v}_{f,\mathcal{S},I}(k, \vec{x}, z, \vec{r}).$$

Let $\textsc{ideal}_{f,\mathcal{S},I}(k, \vec{x}, z)$ denote the probability distribution of $\textsc{ideal}_{f,\mathcal{S},I}(k, \vec{x}, z, \vec{r})$ when $\vec{r}$ is randomly chosen, and let $\textsc{ideal}_{f,\mathcal{S},I}$ denote the distribution ensemble $\{\textsc{ideal}_{f,\mathcal{S},I}(k, \vec{x}, z)\}_{k\in\mathbb{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

We now define security of a protocol. We stress that the following definition does not require black-box simulation.

Definition 8 *Let f be an n-input functionality and let π be an n-party protocol. We say that π* t-securely computes f for strict polynomial-time adversaries and with a simulator that runs in expected polynomial time with respect to f *if for every* strict *polynomial time real-world adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ that runs in expected polynomial time with respect to f, such that for every $I \subseteq [n]$ with $|I| \leq t$, it holds that*

$$\textsc{ideal}_{f,\mathcal{S},I} \stackrel{\text{c}}{\equiv} \textsc{real}_{\pi,\mathcal{A},I}.$$

We refer to such an $\mathcal{S}$ as a simulator *for $\mathcal{A}$.*

The definition can be modified in the natural way to allow ideal-world simulators that run in expected polynomial time in any interaction. We stress that we allow the ideal-world adversary/simulator $\mathcal{S}$ to run in expected polynomial time (with respect to f), unlike [5] where a strict polynomial-time adversary/simulator is required.

Black-box simulation. A more restricted notion of security requires the existence of a black-box simulator. The definition below is the same as Definition 3, rephrased using the present notation for convenience.

Definition 9 *Let f and π be as above. Protocol π is said to t-*securely compute f for strict polynomial-time adversaries with a black-box simulator that runs in expected polynomial time with respect to f *if there exists an oracle machine (black-box simulator) $\mathcal{S}$ that runs in expected polynomial time with respect to f, such that for every strict polynomial-time real-world adversary $\mathcal{A}$ and every $I \subseteq [n]$ with $|I| \leq t$, it holds that*

$$\text{IDEAL}_{f,\mathcal{S}^{\mathcal{A}},I} \stackrel{c}{\equiv} \text{REAL}_{\pi,\mathcal{A},I}.$$

The hybrid model and modular composition. We start by specifying the $(f_1, \ldots, f_m)$-hybrid model in which a protocol evaluating g is run with the assistance of a trusted party who evaluates functions $f_1, \ldots, f_m$. This trusted party will be invoked at special rounds determined by the protocol, and we require (as in [5]) that at most one function call is made at any round. (As in [5], we assume that the number m of functions to be evaluated, the rounds in which these functions are called, and the functions themselves are fixed for any particular value of the security parameter.[12]) Upon termination of the protocol each honest party outputs the value prescribed by the protocol while each corrupted party outputs $\perp$; denote the output of party P_i by v_i and let $\vec{v} = (v_1, \ldots, v_n)$. The hybrid-model adversary, controlling parties I, outputs an arbitrary function of its view, denoted $\text{OUT}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r})$ (note that $\vec{r}$ now includes the random tapes of the honest parties, the adversary, *and* the trusted party). Define:

$$\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}) \stackrel{\text{def}}{=} \text{OUT}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}) \circ \vec{v}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r}).$$

Let $\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z)$ denote the distribution of $\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z, \vec{r})$ when $\vec{r}$ is chosen at random, and let $\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}$ denote the ensemble $\{\text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}(k, \vec{x}, z)\}_{k\in\mathbb{N},\langle\vec{x},z\rangle\in\{0,1\}^*}$.

We define security in the hybrid model in a way similar to before:

Definition 10 *Let $f_1, \ldots, f_m$ and g be n-party functions, and let π be a protocol in the $(f_1, \ldots, f_m)$-hybrid model. Then π is said to t-*securely compute g in the $(f_1, \ldots, f_m)$-hybrid model for strict polynomial-time adversaries with a simulator that runs in expected polynomial time with respect to g *if for any strict polynomial-time $(f_1, \ldots, f_m)$-hybrid-model adversary $\mathcal{A}$ there exists an ideal-process adversary/simulator $\mathcal{S}$ that runs in expected polynomial time with respect to g, and such that for every subset $I \subseteq [n]$ with $|I| \leq t$ we have*

$$\text{IDEAL}_{g,\mathcal{S},I} \stackrel{c}{\equiv} \text{HYBRID}^{f_1,\ldots,f_m}_{\pi,\mathcal{A},I}.$$

The notion of π t-securely evaluating g with a *black-box* simulator is defined in the same way as in Definition 9.

In a real-world execution, calls to a trusted party evaluating the f_i are replaced by an execution of a protocol ρ_i. This is done in the natural way, as described in [5]. We let $\pi^{\rho_1,\ldots,\rho_m}$ denote the real-world protocol that results from replacing each f_i with ρ_i.

Families of α-pseudorandom functions (α-PRFs). Our proof of the modular sequential composition theorem for the case of expected polynomial-time simulation relies on the existence of function ensembles that are pseudorandom for adversaries running in time $\alpha(k)$, where $\alpha(k) = k^{\omega(1)}$ is some superpolynomial function. We formally define this notion now.

[12]As pointed out in [5], any protocol not fulfilling these assumptions can be easily converted into a protocol that does fulfill the assumptions. However, we stress that our composition theorem, like the theorem of [5], only refers to protocols that indeed fulfill the assumptions.

Definition 11 *Let $\mathcal{F}^\ell = \{F_s : \{0,1\}^{\ell(|s|)} \to \{0,1\}\}_{s \in \{0,1\}^*}$ be a set of functions such that F_s is computable in time polynomial in $|s|$, and where each F_s has domain $\{0,1\}^{\ell(|s|)}$ and range $\{0,1\}$. We denote by $\mathcal{H}^{\ell,k}$ the space of all functions with domain $\{0,1\}^{\ell(k)}$ and range $\{0,1\}$. We say that $\mathcal{F}^\ell$ is a* family of α-secure pseudorandom functions with input length ℓ *if for any distinguisher D running in time at most $\alpha(k)$, any auxiliary input z, and all sufficiently-large k, we have:*

$$\left| \Pr_{s \leftarrow \{0,1\}^k}\left[D^{F_s(\cdot)}(1^k, z) = 1\right] - \Pr_{f \leftarrow \mathcal{H}^{\ell,k}}\left[D^{f(\cdot)}(1^k, z)\right] \right| < \frac{1}{\alpha(k)}.$$

If there exists a one-way function f such that for some superpolynomial function $\alpha'(k) = k^{\omega(1)}$, no adversary running in time $\alpha'(k)$ can invert f with probability greater than $1/\alpha'(k)$, then there exist α-secure pseudorandom functions for every polynomial ℓ, for some $\alpha(k) = k^{\omega(1)}$. This is obtained by using a superpolynomially hard one-way function in order to construct superpolynomially hard pseudorandom generators [19], which are then in turn used to construct superpolynomially hard pseudorandom functions [14].

4.2 Proving the Modular Composition Theorem

We now state and prove our main theorem of this section.

Theorem 12 *Assume that for every polynomial ℓ, there exists a superpolynomial function $\alpha(k) = k^{\omega(1)}$ and a family of α-secure pseudorandom functions with input length ℓ. Let $f_1, \ldots, f_m$ and g be n-party functions, let π be an n-party protocol that t-securely computes g in the $(f_1, \ldots, f_m)$-hybrid model (for strict polynomial-time adversaries and with a black-box simulator that runs in expected polynomial time with respect to g) and in which no more than one ideal evaluation call is made at each round, and let $\rho_1, \ldots, \rho_m$ be n-party protocols such that each ρ_i t-securely computes f_i (for strict polynomial-time adversaries and with a simulator[13] that runs in expected polynomial time in any interaction). Then protocol $\pi^{\rho_1,\ldots,\rho_m}$ t-securely computes g (for strict polynomial-time adversaries and with a simulator that runs in expected polynomial time with respect to g).*

We require the stronger property that the simulator for each ρ_i runs in expected polynomial time *in any interaction*, while the resulting simulator for $\pi^{\rho_1,\ldots,\rho_m}$ "only" runs in expected polynomial time with respect to g. However, it is sufficient that the simulator for π runs in expected polynomial time with respect to g. We also do not know how to extend the theorem to show that the stronger assumption that the simulator for π runs in expected polynomial time in any interaction yields the stronger consequence that the resulting simulator for $\pi^{\rho_1,\ldots,\rho_m}$ runs in expected polynomial time in any interaction. Resolving the issue in either of these directions would be an interesting result.

Proof: We begin by describing the high-level structure of the proof and the motivation for our strategy. We follow the structure and notation of the proofs of [5, Theorems 5, 15] and [5, Corollaries 7, 17] as closely as possible. We focus on the case $m = 1$; the general case follows using the techniques described here. We begin with an informal, high-level overview of our proof, stressing where it diverges from [5]: Let $f = f_1$ be an n-party functionality, π a protocol in the f-hybrid model, ρ a protocol that t-securely computes f, and π^ρ the composed protocol. Given a strict polynomial-time adversary $\mathcal{A}$ in the real world (who interacts with parties running π^ρ), our goal is to construct an ideal-world adversary $\mathcal{S}$ (interacting with a trusted party who evaluates g) that runs in expected polynomial time with respect to g and such that for every $I \subseteq [n]$ with $|I| \leq t$, it holds that $\text{IDEAL}_{g,\mathcal{S},I} \stackrel{c}{\equiv} \text{REAL}_{\pi^\rho,\mathcal{A},I}$. (In the remainder of the proof, we omit I as a subscript in an attempt to reduce visual clutter.) We proceed in the following steps:

[13] We stress that the simulator for each of the ρ_i need not be a black-box simulator.

- As in [5], we first construct from $\mathcal{A}$ the (natural) real-world adversary $\mathcal{A}_\rho$ who interacts with parties running ρ as a stand-alone protocol (see Figure 1). Adversary $\mathcal{A}_\rho$ runs in strict polynomial time, and so the security of ρ implies the existence of a simulator $\mathcal{S}_\rho$, who interacts with a trusted party evaluating f, such that $\text{IDEAL}_{f,\mathcal{S}_\rho} \stackrel{c}{\equiv} \text{REAL}_{\rho,\mathcal{A}_\rho}$. Simulator $\mathcal{S}_\rho$ runs in expected polynomial time in any interaction.

- As in [5], using $\mathcal{A}$ and $\mathcal{S}_\rho$ we construct an adversary $\mathcal{A}_\pi$ interacting with parties running π in the f*-hybrid* model and satisfying $\text{HYBRID}^f_{\pi,\mathcal{A}_\pi} \stackrel{c}{\equiv} \text{REAL}_{\pi^\rho,\mathcal{A}}$. Since $\mathcal{A}_\pi$ runs $\mathcal{S}_\rho$ as a sub-routine, and the latter runs in *expected* polynomial time, we *cannot* at this point claim the existence of an expected polynomial-time ideal-world adversary $\mathcal{S}$ such that $\text{IDEAL}_{g,\mathcal{S}} \stackrel{c}{\equiv} \text{HYBRID}^f_{\pi,\mathcal{A}_\pi}$ (such a claim, if true, would complete the proof as in [5]).

- Instead, we first construct from $\mathcal{A}_\pi$ a modified adversary $\mathcal{A}'_\pi$ (still interacting with parties running π in the f-hybrid model) that runs in expected polynomial time with respect to π in the f-hybrid model, and for which $\text{HYBRID}^f_{\pi,\mathcal{A}'_\pi} \stackrel{c}{\equiv} \text{HYBRID}^f_{\pi,\mathcal{A}_\pi}$ under the assumption that α-secure pseudorandom functions exist. (See Claims 13 and 14.) This is the crux of our proof, and further details are given below.

- Let $\mathcal{S}_\pi$ denote a black-box simulator for π (as required by Definition 9). We construct an ideal-world adversary $\mathcal{S}$ that runs a slightly modified version of $\mathcal{S}_\pi$ with oracle access to $\mathcal{A}'_\pi$. We then prove that (1) $\mathcal{S}$ runs in expected polynomial time with respect to g (even when taking the running time of $\mathcal{A}'_\pi$ into account); and (2) $\text{IDEAL}_{g,\mathcal{S}} \stackrel{c}{\equiv} \text{HYBRID}^f_{\pi,\mathcal{A}'_\pi}$. (See Claims 15 and 16.)

See Table 2 for a high-level summary of the above.

Adversary	**Attack setting**	**Running time**	**Comments**
$\mathcal{A}$	π^ρ (real world)	strict poly-time	original adversary
$\mathcal{A}_\rho$	ρ (real world)	strict poly-time	constructed by ignoring π-messages of $\mathcal{A}$
$\mathcal{S}_\rho$	f (ideal world)	expected poly-time in any interaction	constructed from $\mathcal{A}_\rho$ using security of ρ
$\mathcal{A}_\pi$	π (f-hybrid model)	expected poly-time in any interaction	constructed from $\mathcal{A}$ and $\mathcal{S}_\rho$
$\mathcal{A}'_\pi$	π (f-hybrid model)	expected poly-time w.r.t. π (in f-hybrid model)	modification of $\mathcal{A}_\pi$ (see text)
$\mathcal{S}$	g (ideal world)	expected poly-time w.r.t. g	constructed using $\mathcal{A}'_\pi$ and black-box simulator $\mathcal{S}_\pi$ for π

Table 2: Informal summary of adversaries/simulators used in the proof.

Motivation for the proof. Before continuing with the details, we provide an informal overview of the key idea behind the proof of the theorem. Let $\mathcal{S}_\pi$ be the black-box simulator for π, and recall that $\mathcal{A}_\pi$ invokes $\mathcal{S}_\rho$ as a sub-routine. Naïvely following the approach of [5], one runs into the problem that the expected running time of $\mathcal{S}_\pi^{\mathcal{A}_\pi}$, counting steps of both machines, may not be polynomial (in any sense). In particular, if $\mathcal{S}_\pi$ "rewinds" $\mathcal{A}_\pi$, then it also (effectively) "rewinds"

$\mathcal{S}_\rho$. In this case, an expected polynomial-time machine $\mathcal{S}_\rho$ is invoked multiple times *with the same random tape*. This introduces a dependency between the executions, and we can no longer claim that the total running time of $\mathcal{A}_\pi$ (including the running time of $\mathcal{S}_\rho$) is polynomial. An example of why this is problematic was described in the Introduction, regarding the composition of two expected polynomial-time machines A and B. We also provide a counterexample for the specific case of modular sequential composition in Appendix B.

A first solution that comes to mind is to have $\mathcal{S}_\pi$ choose an independent random tape for $\mathcal{A}_\pi$ every time it invokes $\mathcal{A}_\pi$ (with the consequence that $\mathcal{S}_\rho$ will be called using a fresh random tape each time). The problem with this approach is that Definition 9 places no limitations on how $\mathcal{S}_\pi$ sets the random tape of $\mathcal{A}_\pi$. In particular, the resulting simulation output by $\mathcal{S}_\pi$ may no longer be "good" if $\mathcal{S}_\pi$ is *forced* to invoke $\mathcal{A}_\pi$ with a new random tape in each oracle call. This concern is not only definitional. Rather, typical "rewinding" simulation strategies rely heavily on the fact that the random tape of the adversary is fixed throughout.

Instead, our solution is to modify $\mathcal{A}_\pi$ (resulting in $\mathcal{A}'_\pi$ as described above) so that it invokes $\mathcal{S}_\rho$ with a random tape that is determined by applying a *pseudorandom* function to the auxiliary input provided to $\mathcal{S}_\rho$ (this is reminiscent of a similar technique used in [7]). This has the effect that even when $\mathcal{A}'_\pi$ is run with a *fixed* random tape (that includes the key for the pseudorandom function), $\mathcal{S}_\rho$ is effectively invoked with a fresh random tape each time. That is, even when $\mathcal{A}'_\pi$ is "rewound" during simulation the random tape used by $\mathcal{S}_\rho$ upon each "rewinding" will be computationally independent from all past random tapes. (This holds unless $\mathcal{A}'_\pi$ is invoked with a series of incoming messages that was already sent in the past. However, in this case, since $\mathcal{A}'_\pi$ uses a fixed random tape, the exact same response will be given as previously. We could also assume without loss of generality that $\mathcal{S}_\pi$ never repeats a query to $\mathcal{A}'_\pi$.) The technical portion of the proof then comes down to showing that if an α-secure pseudorandom function is used to do this, the resulting simulation is "good." The intuition as to why this is the case, and the techniques used in the proof (including superpolynomial truncation of the adversary's execution), are similar to those used in the proof of Theorem 5.

Adversary $\mathcal{A}_\rho$, interacting with parties $P_1, \ldots, P_n$ running protocol ρ, begins with the following inputs: security parameter 1^k, identities I of corrupted parties along with their inputs $\vec{x}_I$, auxiliary input z, and random tape $r_\mathcal{A}$. Do:

1. Let ℓ_ρ be the round in which π^ρ calls protocol ρ. Interpret z as an internal state of $\mathcal{A}$, controlling the parties in I, at round $\ell_\rho - 1$.
2. Run $\mathcal{A}$ from internal state z. Messages sent by uncorrupted parties in the real world (running ρ) are forwarded to $\mathcal{A}$. Messages sent by $\mathcal{A}$ on behalf of corrupted parties are forwarded to the appropriate real-world recipients.
3. Once execution of ρ is done, $\mathcal{A}_\rho$ outputs the current internal state of $\mathcal{A}$.

Figure 1: The description of adversary $\mathcal{A}_\rho$.

Proof details. We now proceed with the proof in detail. The first steps of our proof as described in the above outline — namely, the construction of $\mathcal{A}_\rho$, $\mathcal{S}_\rho$, and $\mathcal{A}_\pi$ — are exactly as in [5], but are nevertheless described in Figures 1 and 2 for convenience. Recall that we begin with a real-world adversary $\mathcal{A}$ interacting with parties running protocol π^ρ. As described earlier and shown in Figure 1, we first construct a real-world adversary $\mathcal{A}_\rho$ attacking protocol ρ. Note that $\mathcal{A}_\rho$ runs in *strict* polynomial time. Security of ρ thus implies the existence of an ideal-process adversary $\mathcal{S}_\rho$ that runs in expected polynomial time in any interaction and such that $\text{IDEAL}_{f,\mathcal{S}_\rho} \stackrel{c}{\equiv} \text{REAL}_{\rho,\mathcal{A}_\rho}$. We remind the reader again that we do *not* assume or require that ρ *black-box* securely computes f.

Next, we construct the adversary $\mathcal{A}_\pi$ attacking parties running protocol π in the f-hybrid model. Loosely speaking, $\mathcal{A}_\pi$ runs $\mathcal{A}$ until the protocol ρ is supposed to begin. At this point, $\mathcal{A}$ expects to run ρ, whereas $\mathcal{A}_\pi$ should use an ideal call to f. Therefore, $\mathcal{A}_\pi$ invokes $\mathcal{S}_\rho$ using the current internal state z^ρ of $\mathcal{A}$ as its auxiliary input, and forwarding the messages between $\mathcal{S}_\rho$ and the trusted party computing f. The output of $\mathcal{S}_\rho$ is an internal state of $\mathcal{A}$ at the end of the execution of ρ. Adversary $\mathcal{A}_\pi$ continues by invoking $\mathcal{A}$ from this state and running $\mathcal{A}$ until the conclusion of π. This is described in Figure 2. In describing $\mathcal{A}_\pi$ we explicitly have it parse its random tape into two portions: a portion r used to run $\mathcal{A}$ and a portion r^* used to run $\mathcal{S}_\rho$ (we will use this fact later). Exactly as in [5] (and so we do not repeat the proof here), it holds that $\text{HYBRID}^f_{\pi,\mathcal{A}_\pi} \stackrel{c}{\equiv} \text{REAL}_{\pi^\rho,\mathcal{A}}$.

Adversary $\mathcal{A}_\pi$, interacting with parties $P_1, \ldots, P_n$ running protocol π in the f-hybrid model, begins with the following inputs: security parameter 1^k, identities I of corrupted parties along with their inputs $\vec{x}_I$, auxiliary input z, and random tape $r_\mathcal{A}$ parsed as r, r^*. Do:

1. Invoke $\mathcal{A}$ on $1^k, I, \vec{x}_I, z$ using random tape r, and follow the instructions of $\mathcal{A}$ up to round $l_\rho - 1$.
2. At the onset of round l_ρ, $\mathcal{A}$ expects to interact with parties running ρ (as a subroutine), whereas parties $P_1, \ldots, P_n$ actually call a trusted party to evaluate f. To continue the run of $\mathcal{A}$, invoke simulator $\mathcal{S}_\rho$ as follows:
   (a) $\mathcal{S}_\rho$ is given $1^k, I$, and arbitrary values (say, all 0's) for the inputs of the parties in I. The auxiliary input z^ρ for $\mathcal{S}_\rho$ is set to the current internal state of $\mathcal{A}$. The random tape for $\mathcal{S}_\rho$ is r^*.
   (b) When $\mathcal{S}_\rho$ wishes to send the trusted party the inputs of the corrupted parties, send these values to the trusted party, and hand the values returned by the trusted party back to $\mathcal{S}_\rho$.
3. The output of $\mathcal{S}_\rho$ is an internal state of $\mathcal{A}$ at the end of the execution of ρ. Run $\mathcal{A}$ using this internal state and resume following $\mathcal{A}$'s instructions until the completion of π. Then output whatever $\mathcal{A}$ outputs and halt.

Figure 2: The description of adversary $\mathcal{A}_\pi$.

$\mathcal{A}_\pi$ runs in *expected* polynomial time in any interaction (this is due to the fact that the only part of $\mathcal{A}_\pi$ that does not run in strict polynomial time is the invocation of $\mathcal{S}_\rho$, and the latter is done only once). However, π is only guaranteed to be secure for *strict* polynomial-time adversaries. Therefore, we cannot immediately claim the existence of an appropriate ideal-world simulator corresponding to the hybrid-model adversary $\mathcal{A}_\pi$. Dealing with this issue forms the crux of our proof.

Let $\alpha(k)$ be as in the theorem statement, and assume without loss of generality that $\alpha(k) = O(2^k)$. We modify $\mathcal{A}_\pi$ to construct an adversary $\mathcal{A}'_\pi$ as described in Figure 3. Let $\mathcal{F}^\ell$ be an α-secure PRF taking inputs of an appropriate length (namely, k plus the length of z^ρ, as described in Figure 3).[14] The random tape of $\mathcal{A}'_\pi$ is now parsed as r, s, where r is used as before (namely,

> Adversary $\mathcal{A}'_\pi$, interacting with parties $P_1, \ldots, P_n$ running protocol π in the f-hybrid model, begins with the following inputs: security parameter 1^k, identities I of corrupted parties along with their inputs $\vec{x}_I$, auxiliary input z, and random tape $r_\mathcal{A}$ parsed as r, s with $|s| = k$. Do:
>
> 1. $\mathcal{A}'_\pi$ keeps track of the total number of steps run below, counting each invocation of $F_s(\cdot)$ as a single step. If the total number of steps ever exceeds $\alpha(k)/2$, halt with output $\perp$.
>
> 2. Invoke $\mathcal{A}$ on $1^k, I, \vec{x}_I, z$ using random tape r, and follow the instructions of $\mathcal{A}$ up to round $l_\rho - 1$.
>
> 3. At the onset of round l_ρ, invoke simulator $\mathcal{S}_\rho$ as follows:
>
>   (a) $\mathcal{S}_\rho$ is given $1^k, I$, and arbitrary values (say, all 0's) for the inputs of the parties in I. The auxiliary input z^ρ for $\mathcal{S}_\rho$ is set to the current internal state of $\mathcal{A}$. The random tape for $\mathcal{S}_\rho$ is determined as described below.
>
>   (b) The random tape r^* for $\mathcal{S}_\rho$ is generated bit-by-bit, as needed, in the following way: the i^{th} random bit needed by $\mathcal{S}_\rho$ is set to $F_s(z^\rho \| \langle i \rangle)$, where $\langle i \rangle$ is the k-bit binary representation of i. (Note that $\mathcal{A}'_\pi$ aborts anyway if $\mathcal{S}_\rho$ ever requires more than $\alpha(k)/2$ random bits. Since $k > \log(\alpha(k)/2)$ for k large enough [recall $\alpha(k) = O(2^k)$], a k-bit counter is sufficient to run $\mathcal{S}_\rho$ to completion.)
>
>   (c) When $\mathcal{S}_\rho$ wishes to send the trusted party the inputs of the corrupted parties, send these values to the trusted party, and hand the values returned by the trusted party back to $\mathcal{S}_\rho$.
>
> 4. The output of $\mathcal{S}_\rho$ is an internal state of $\mathcal{A}$ at the end of the execution of ρ. Run $\mathcal{A}$ using this internal state and resume following $\mathcal{A}$'s instructions until the completion of π. Then output whatever $\mathcal{A}$ outputs and halt.

Figure 3: The description of adversary $\mathcal{A}'_\pi$.

to run $\mathcal{A}$) while s is used as a key to an α-secure pseudorandom function. Then $\mathcal{A}'_\pi$ generates the random tape r^* for $\mathcal{S}_\rho$ as a pseudorandom function of z^ρ (see Figure 3 for details). In addition, $\mathcal{A}'_\pi$ halts with output $\perp$ if it ever exceeds $\alpha(k)/2$ steps overall (not including steps used in computing F_s). Otherwise, $\mathcal{A}'_\pi$ works in exactly the same way as $\mathcal{A}_\pi$. We stress the differences between $\mathcal{A}_\pi$ and $\mathcal{A}'_\pi$:

1. $\mathcal{A}'_\pi$ chooses $\mathcal{S}_\rho$'s random tape by invoking a pseudorandom function on the internal state of $\mathcal{A}$, whereas $\mathcal{A}_\pi$ chooses it uniformly.

2. $\mathcal{A}'_\pi$ truncates its execution after $\alpha(k)/2$ steps, whereas $\mathcal{A}_\pi$ does not.

We now prove that $\mathcal{A}'_\pi$ runs in expected polynomial time with respect to π in the f-hybrid model, and that $\textsc{hybrid}^f_{\pi,\mathcal{A}'_\pi} \stackrel{\text{c}}{\equiv} \textsc{hybrid}^f_{\pi,\mathcal{A}_\pi}$.

[14] In the theorem statement we have assumed that α-secure PRFs exist for any polynomial domain length $\ell(k)$. Since z^ρ is the internal state of a strict polynomial-time machine, ℓ is a fixed polynomial (depending only on $\mathcal{A}$).

Claim 13 *If $\mathcal{F}^\ell$ is an α-secure pseudorandom function for some $\alpha(k) = k^{\omega(1)}$, the original adversary $\mathcal{A}$ runs in strict polynomial time, and $\mathcal{S}_\rho$ runs in expected polynomial time in any interaction, then $\mathcal{A}'_\pi$ runs in expected polynomial time with respect to π in the f hybrid model.*

Proof: All we need for the proof of this claim is that $\mathcal{S}_\rho$ runs in expected polynomial time with respect to f. Nevertheless, since $\mathcal{S}_\rho$ runs in expected polynomial time in any interaction anyway, we will rely on this stronger assumption to simplify the proof. (We will need $\mathcal{S}_\rho$ to run in expected polynomial time in any interaction in order to prove Claim 15.)

We will actually prove a slightly stronger result: namely, that for any setting of the random coins of the other parties (i.e., the honest parties running π as well as the trusted party computing f), the expected running time of $\mathcal{A}'_\pi$ — over the random coins used by $\mathcal{A}'_\pi$ — when interacting with these parties is polynomial. Consider an adversary $\hat{\mathcal{A}}_\pi$ which is identical to $\mathcal{A}_\pi$ except that it halts with output $\perp$ if it ever exceeds $\alpha(k)/2$ steps. (In particular, $\hat{\mathcal{A}}_\pi$ chooses a truly random tape r^* for $\mathcal{S}_\rho$ instead of a pseudorandom one.) For any fixed set of global values global (which contains the security parameter 1^k, inputs and random coins for the honest parties and the trusted party computing f, inputs to $\hat{\mathcal{A}}_\pi$, and the initial portion r of the random tape of $\hat{\mathcal{A}}_\pi$), let $\mathsf{time}_X(\mathsf{global})$ be a random variable (over choice of coins used to run $\mathcal{S}_\rho$) denoting the running time of the algorithm $X \in \{\mathcal{A}'_\pi, \hat{\mathcal{A}}_\pi\}$ when interacting with parties running π in the f-hybrid model, counting calls to $F_s(\cdot)$ as a single step in the case of $\mathcal{A}'_\pi$. We first claim that

$$\mathbf{Exp}_{r^*}[\mathsf{time}_{\hat{\mathcal{A}}_\pi}(\mathsf{global})] \leq q(k) \tag{9}$$

for some polynomial $q(\cdot)$. This is simply due to the facts that $\mathcal{S}_\rho$ runs in expected polynomial time in any interaction, and that $\mathcal{A}$ runs in strict polynomial time. (Therefore, aside from the call to $\mathcal{S}_\rho$, the adversary $\mathcal{A}_\pi$ runs in strict polynomial time. Furthermore, the overhead due to the counter maintained by $\hat{\mathcal{A}}_\pi$ introduces only a multiplicative polynomial factor, and this is the only difference between $\mathcal{A}_\pi$ and $\hat{\mathcal{A}}_\pi$.) We proceed by showing that replacing the uniform coins r^* (used by $\hat{\mathcal{A}}_\pi$ when running $\mathcal{S}_\rho$) with α-strong pseudorandom coins (used by $\mathcal{A}'_\pi$ when running $\mathcal{S}_\rho$) does not make a "significant" difference to the running time of $\hat{\mathcal{A}}_\pi$.

If we can bound $\mathbf{Exp}_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global})]$ by a polynomial it would follow that $\mathcal{A}'_\pi$ runs in expected polynomial time with respect to π in the f-hybrid model, since the additional overhead due to computing F_s introduces at most a multiplicative polynomial factor. The crux of the proof is that for every value of global, all $t \leq \alpha(k)/2$, and all large enough values of k:

$$\left|\Pr_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global}) \geq t] - \Pr_{r^*}[\mathsf{time}_{\hat{\mathcal{A}}_\pi}(\mathsf{global}) \geq t]\right| < \frac{1}{\alpha(k)}. \tag{10}$$

Intuitively, if this were not the case, then the running time of $\mathcal{A}'_\pi$ could be used to distinguish $\mathcal{F}^\ell$ from a random function. Formally, given (global, t, k) such that Eq. (10) does not hold we can construct a distinguisher D that takes these values as auxiliary input and runs $\mathcal{A}'_\pi$ (using global to simulate the actions of the honest parties and the trusted party computing f) but using its oracle to generate the random tape for $\mathcal{S}_\rho$. (Namely, D generates the i^{th} random bit for $\mathcal{S}_\rho$, as needed, by querying $z^\rho \| \langle i \rangle$ to its oracle.) If the running time of $\mathcal{A}'_\pi$ exceeds t steps, then D outputs 1; otherwise, it outputs 0. The running time of D is strictly bounded by $\alpha(k)$ (for large enough k). We therefore have:

$$\Pr_{s \leftarrow \{0,1\}^k}\left[D^{F_s(\cdot)}(1^k, \mathsf{global}, t) = 1\right] = \Pr_{s \leftarrow \{0,1\}^k}\left[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global}) \geq t\right]$$

and

$$\Pr_{f \leftarrow \mathcal{H}^{\ell,k}}\left[D^{f(\cdot)}(1^k, \mathsf{global}, t) = 1\right] = \Pr_{r^* \leftarrow \{0,1\}^*}\left[\mathsf{time}_{\hat{\mathcal{A}}_\pi}(\mathsf{global}) \geq t\right]$$

(where, recall, $\mathcal{H}^{\ell,k}$ represents the space of all boolean functions with domain $\ell(k)$). Since $\mathcal{F}^\ell$ is an α-secure PRF, Equation (10) follows. We conclude that the expected running time of $\mathcal{A}'_\pi$ for large enough k and all global inputs global is bounded by:

$$
\begin{aligned}
\mathbf{Exp}_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global})] &= \sum_{t=1}^{\infty} t \cdot \Pr_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global}) = t] \\
&= \sum_{t=1}^{\alpha(k)/2} t \cdot \Pr_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global}) = t] \\
&= \sum_{t=1}^{\alpha(k)/2} \Pr_s[\mathsf{time}_{\mathcal{A}'_\pi}(\mathsf{global}) \geq t] \\
&\leq \sum_{t=1}^{\alpha(k)/2} \left(\Pr_{r^*}[\mathsf{time}_{\hat{\mathcal{A}}_\pi}(\mathsf{global}) \geq t] + \frac{1}{\alpha(k)} \right) \\
&= \frac{1}{2} + \mathbf{Exp}_{r^*}[\mathsf{time}_{\hat{\mathcal{A}}_\pi}(\mathsf{global})],
\end{aligned}
$$

where the second equality is due to the fact that $\mathcal{A}'_\pi$ truncates its execution if it ever exceeds $\alpha(k)/2$ steps. Using Eq. (9), we conclude that $\mathcal{A}'_\pi$ runs in expected polynomial time with respect to π in the f-hybrid model. $\square$

The reader may wonder why we are unable to show that $\mathcal{A}'_\pi$ runs in expected polynomial time in any interaction. In fact, $\hat{\mathcal{A}}_\pi$ does run in expected polynomial time in any interaction; however, Eq. (10) may no longer hold when $\mathcal{A}'_\pi$ interacts with an arbitrary ITM M. The problem is that, in proving Equation (10), in constructing D, we need to construct a distinguisher D that can simulate the actions of ITM M in time $\alpha(k)$; however, it is not clear how to do this for arbitrary M since, in particular, M might be all-powerful.

In the next claim, we show that the behavior of $\mathcal{A}'_\pi$ is "close" to that of $\mathcal{A}_\pi$.

Claim 14 *If $\mathcal{F}^\ell$ is an α-secure pseudorandom function for some $\alpha(k) = k^{\omega(1)}$, then*

$$\textsc{hybrid}^f_{\pi,\mathcal{A}'_\pi} \stackrel{c}{\equiv} \textsc{hybrid}^f_{\pi,\mathcal{A}_\pi}.$$

Proof: Let $\hat{\mathcal{A}}_\pi$ be as in the previous claim. Since $\mathcal{A}_\pi$ runs in expected polynomial time (in any interaction), the probability that $\mathcal{A}_\pi$ exceeds $\alpha(k)/2$ steps in any execution is negligible (using Markov's inequality). Hence $\textsc{hybrid}^f_{\pi,\hat{\mathcal{A}}_\pi}$ and $\textsc{hybrid}^f_{\pi,\mathcal{A}_\pi}$ are *statistically* close. Now, $\hat{\mathcal{A}}_\pi$ is identical to $\mathcal{A}'_\pi$ except that $\hat{\mathcal{A}}_\pi$ uses a truly random tape r^* for $\mathcal{S}_\rho$ whereas $\mathcal{A}'_\pi$ uses a *pseudorandom* tape for $\mathcal{S}_\rho$. Since $\hat{\mathcal{A}}_\pi$ and $\mathcal{A}'_\pi$ both run in at most $\alpha(k)/2$ steps (for the case of $\mathcal{A}'_\pi$, not counting the time required to compute F_s), the assumption that $\mathcal{F}^\ell$ is an α-secure PRF implies that $\textsc{hybrid}^f_{\pi,\mathcal{A}'_\pi}$ is computationally indistinguishable from $\textsc{hybrid}^f_{\pi,\hat{\mathcal{A}}_\pi}$. In fact, we claim something stronger: for any value of global (recall, this includes the inputs and randomness used by all parties except for the random tape used to run $\mathcal{S}_\rho$), any poly-time distinguisher D, all auxiliary input z, and all large enough values of k:

$$
\begin{aligned}
\Big| &\Pr[D(1^k, \mathsf{global}, z, \textsc{hybrid}^f_{\pi,\mathcal{A}'_\pi}(\mathsf{global})) = 1] \\
&- \Pr[D(1^k, \mathsf{global}, z, \textsc{hybrid}^f_{\pi,\hat{\mathcal{A}}_\pi}(\mathsf{global})) = 1] \Big| < \frac{1}{\alpha(k)}
\end{aligned}
$$

(this is stronger both because $\alpha(k)$ is superpolynomial and also because we are fixing all random coins except those used by $\mathcal{S}_\rho$). If the above does not hold, then we can construct a distinguisher D' for $\mathcal{F}^\ell$ in the natural way: given (k, global, z) for which the above does not hold, D' takes these values as input, runs $\mathcal{A}'_\pi$ (simulating the actions of all other parties using global), but uses its oracle to generate the random tape for $\mathcal{S}_\rho$. It runs D on the output of $\mathcal{A}'_\pi$, and outputs whatever D outputs. It is not hard to see that D' runs for at most $\alpha(k)$ steps (for large enough k) and distinguishes $\mathcal{F}^\ell$ from a random boolean function with probability better than $1/\alpha(k)$, a contradiction. $\square$

Constructing the simulator $\mathcal{S}$. Until now, we have constructed an f-hybrid adversary $\mathcal{A}'_\pi$ that runs in expected polynomial time with respect to π in the f-hybrid model, and has the property that the output distribution of an execution of π in the f-hybrid model with $\mathcal{A}'_\pi$ is computationally indistinguishable from a real execution of the composed protocol π^ρ with adversary $\mathcal{A}$. It remains to construct the ideal-model simulator $\mathcal{S}$ that interacts with a trusted party computing g.

Since π *black-box* securely computes g, there exists an oracle machine $\mathcal{S}_\pi$ satisfying the conditions of Definition 9 (with appropriate modifications for consideration of the f-hybrid model). Our simulator $\mathcal{S}$ works by simply invoking $\mathcal{S}_\pi$ with oracle $\mathcal{A}'_\pi$, with the limitation that $\mathcal{S}$ halts with output $\perp$ if it (i.e., $\mathcal{S}$) ever exceeds $\alpha(k)/2$ steps (including the running time of $\mathcal{A}'_\pi$ but, as always, not including time spent computing F_s). In order to prove the theorem, we need to show that

1. $\mathcal{S}$ runs in expected polynomial time with respect to g (even when taking the running time of $\mathcal{A}'_\pi$ into account)
2. $\text{HYBRID}^f_{\pi,\mathcal{A}'_\pi} \stackrel{c}{\equiv} \text{IDEAL}_{g,\mathcal{S}}$.

We stress that neither of these claims are immediate since $\mathcal{A}'_\pi$ is an *expected* polynomial-time adversary, and the simulator $\mathcal{S}_\pi$ is only guaranteed to "work" when it is given a *strict* polynomial-time oracle. However, as we have discussed in the motivation to the proof, the fact that $\mathcal{A}'_\pi$ essentially uses a new (pseudo)random tape for every invocation of $\mathcal{S}_\rho$ ensures that the expected overall running time is polynomial.

Claim 15 *Assuming that $\mathcal{F}^\ell$ is an α-secure pseudorandom function for some $\alpha(k) = k^{\omega(1)}$, assuming that $\mathcal{S}_\pi$ is an oracle machine that runs in expected polynomial time with respect to g, and assuming that $\mathcal{S}_\rho$ runs in expected polynomial time in any interaction, the simulator $\mathcal{S}$ runs in expected polynomial time with respect to g.*

Proof: First imagine a simulator $\widetilde{\mathcal{S}}$ that differs from $\mathcal{S}$ in the following way: whenever $\mathcal{S}_\rho$ is called from within $\mathcal{A}'_\pi$, simulator $\widetilde{\mathcal{S}}$ monitors the value of z^ρ at that point. Let z_i^ρ denote the value of z^ρ the i^{th} time $\mathcal{S}_\rho$ is called. Now, in contrast to $\mathcal{S}$, adversary $\widetilde{\mathcal{S}}$ generates the random tape r_i^* bit-by-bit, as needed, in the following way: if $z_i^\rho = z_j^\rho$ for some $j < i$, then set $r_i^* = r_j^*$. Otherwise, choose r_i^* uniformly at random. We first show that $\widetilde{\mathcal{S}}$ runs in expected polynomial time with respect to g, and then show (as in the proof of Claim 13) that the expected running times of $\widetilde{\mathcal{S}}$ and $\mathcal{S}$ cannot differ "too much." Intuitively, $\widetilde{\mathcal{S}}$ runs in expected polynomial time because it invokes $\mathcal{S}_\rho$ with fresh random coins each time. Thus, there is no dependence between the different invocations.

Formally, the running time of $\widetilde{\mathcal{S}}$ is the sum of three components: $\mathsf{time}_{\mathcal{S}_\pi}$, the running time of black-box simulator $\mathcal{S}_\pi$ (counting its oracle calls to $\mathcal{A}'_\pi$ as a single step); $\mathsf{time}_{\mathcal{A}'_\pi}$, the running time of $\mathcal{A}'_\pi$ (when answering oracle calls of $\mathcal{S}_\pi$) but excluding time spent running $\mathcal{S}_\rho$; and $\mathsf{time}_{\mathcal{S}_\rho}$, the total running time of $\mathcal{S}_\rho$ when called by $\mathcal{A}'_\pi$ (over all invocations of $\mathcal{S}_\rho$; recall that $\mathcal{A}'_\pi$ invokes $\mathcal{S}_\rho$ once in each invocation). By linearity of expectations, it suffices to bound the expectation of each of these components individually. The expected value of $\mathsf{time}_{\mathcal{S}_\pi}$ is polynomial since oracle machine

$\mathcal{S}_\pi$ runs in expected polynomial time with respect to g (and, as defined in Section 2, this holds regardless of the oracle with which $\mathcal{S}_\pi$ interacts). Furthermore, since $\mathcal{A}'_\pi$ runs in *strict* polynomial time when excluding the steps of $\mathcal{S}_\rho$, and since $\mathcal{S}_\pi$ makes an expected polynomial number of calls to $\mathcal{A}'_\pi$, the expected value of $\mathsf{time}_{\mathcal{A}'_\pi}$ (excluding $\mathcal{S}_\rho$'s steps) is polynomial as well.

It remains to analyze $\mathsf{time}_{\mathcal{S}_\rho}$. Since $\mathcal{S}_\pi$ makes at most $\mathsf{time}_{\mathcal{S}_\pi}$ oracle calls, we have

$$\mathsf{time}_{\mathcal{S}_\rho} \leq \sum_{i=1}^{\mathsf{time}_{\mathcal{S}_\pi}} \mathsf{time}_{\mathcal{S}_\rho}(i),$$

where $\mathsf{time}_{\mathcal{S}_\rho}(i)$ represents the running time of $\mathcal{S}_\rho$ in its i^{th} execution. We thus have:

$$\begin{aligned}
\mathbf{Exp}\left[\mathsf{time}_{\mathcal{S}_\rho}\right] &\leq \mathbf{Exp}\left[\sum_{i=1}^{\mathsf{time}_{\mathcal{S}_\pi}} \mathsf{time}_{\mathcal{S}_\rho}(i)\right] \\
&= \sum_{\ell=1}^{\infty}\left(\Pr\left[\mathsf{time}_{\mathcal{S}_\pi}=\ell\right]\cdot \mathbf{Exp}\left[\sum_{i=1}^{\ell}\mathsf{time}_{\mathcal{S}_\rho}(i) \mid \mathsf{time}_{\mathcal{S}_\pi}=\ell\right]\right) \\
&= \sum_{\ell=1}^{\infty}\left(\Pr\left[\mathsf{time}_{\mathcal{S}_\pi}=\ell\right]\cdot \sum_{i=1}^{\ell}\mathbf{Exp}\left[\mathsf{time}_{\mathcal{S}_\rho}(i) \mid \mathsf{time}_{\mathcal{S}_\pi}=\ell\right]\right)
\end{aligned}$$

Now, recall that $\mathcal{S}_\rho$ runs in expected polynomial time *in any interaction.* This means that for every input x and every auxiliary input z, the expected running time of $\mathcal{S}_\rho$ (taken over its own random tape) is polynomial. Now, since $\widetilde{\mathcal{S}}$ invokes $\mathcal{S}_\rho$ with a fresh random tape each time, the only dependence between the execution of $\mathcal{S}_\rho$ and the random variable $\mathsf{time}_{\mathcal{S}_\pi}$ is due to the auxiliary-input z that $\mathcal{S}_\rho$ receives. We conclude that $\mathbf{Exp}\left[\mathsf{time}_{\mathcal{S}_\rho}(i) \mid \mathsf{time}_{\mathcal{S}_\pi}=\ell\right]$ is bound by a fixed polynomial $p_{\mathcal{S}_\rho}(\cdot)$. We stress that $\mathcal{S}_\rho$ is invoked by $\widetilde{\mathcal{S}}$, who also "plays" the ideal functionality for $\mathcal{S}_\rho$. Since we have no control over the strategy of $\widetilde{\mathcal{S}}$ (which is derived from $\mathcal{S}_\pi$), it is possible that $\widetilde{\mathcal{S}}$ does not reply as the trusted party computing f would reply to $\mathcal{S}_\rho$. It is for this reason that we need that $\mathcal{S}_\rho$ be expected polynomial time *in any interaction* (rather than just being expected polynomial time with respect to f).

Continuing, we obtain:

$$\begin{aligned}
\mathbf{Exp}\left[\mathsf{time}_{\mathcal{S}_\rho}\right] &\leq \sum_{\ell=1}^{\infty}\left(\Pr\left[\mathsf{time}_{\mathcal{S}_\pi}=\ell\right]\cdot \ell \cdot p_{\mathcal{S}_\rho}(k)\right) \\
&= p_{\mathcal{S}_\rho}(k)\cdot \mathbf{Exp}\left[\mathsf{time}_{\mathcal{S}_\pi}\right].
\end{aligned}$$

Since the expected running time of $\mathcal{S}_\pi$ is polynomial, this completes the proof that $\widetilde{\mathcal{S}}$ runs in expected polynomial time with respect to g.

Exactly as in the proof of Claim 13, we now use the fact that $\mathcal{F}^\ell$ is an α-secure PRF to show that $\mathcal{S}$ also runs in expected polynomial time with respect to g. Let $\mathsf{time}_{\mathcal{S}}$ denote the running time of $\mathcal{S}$, counting calls to $F_s(\cdot)$ as a single step (as usual, if the expectation of $\mathsf{time}_{\mathcal{S}}$ is polynomial, then the expectation of the true running time of $\mathcal{S}$ — which includes the time required to compute F_s — is polynomial as well). We first claim that for all values global (this includes the inputs of all honest parties, the inputs and auxiliary input given to the simulator, and the random coins used in computing g), all $t \leq \alpha(k)/2$, and all large enough values of k:

$$\left|\Pr[\mathsf{time}_{\mathcal{S}}(\mathsf{global}) \geq t] - \Pr[\mathsf{time}_{\widetilde{\mathcal{S}}}(\mathsf{global}) \geq t]\right| < \frac{1}{\alpha(k)}. \tag{11}$$

Otherwise, we construct a distinguisher D for $\mathcal{F}^\ell$. Specifically, let (k, global, t) be such that the above does not hold. Then D, on input 1^k and auxiliary input global, will run the strategy of $\mathcal{S}/\widetilde{\mathcal{S}}$ while using its oracle to generate the random tape for $\mathcal{S}_\rho$, as needed. Finally, if the running time exceeds t steps, D outputs 1; otherwise, it outputs 0. Note that D runs for at most $\alpha(k)$ steps. Furthermore, when D's oracle is a function from $\mathcal{F}^\ell$, then it exactly simulates $\mathcal{S}$; on the other hand, when D's oracle is a random boolean function, then it exactly simulates $\widetilde{\mathcal{S}}$. Using the fact that $\mathcal{F}^\ell$ is an α-secure PRF, Eq. (11) follows. We conclude that the expected running time of $\mathcal{S}$ on any set of inputs global is bounded by

$$\begin{aligned}\sum_{t=1}^{\alpha(k)/2} \Pr[\mathsf{time}_{\mathcal{S}} \geq t] &\leq \sum_{t=1}^{\alpha(k)/2} \Pr[\mathsf{time}_{\widetilde{\mathcal{S}}} \geq t] + \frac{1}{\alpha(k)} \\ &= \mathbf{Exp}\left[\mathsf{time}_{\widetilde{\mathcal{S}}}\right] + \frac{1}{2}.\end{aligned}$$

Since, as we have already shown, the expected running time of $\mathsf{time}_{\widetilde{\mathcal{S}}}$ is polynomial with respect to g, it follows that $\mathcal{S}$ runs in expected polynomial time with respect to g as well. □

To complete the proof of the theorem, we prove the following claim:

Claim 16 $\text{IDEAL}_{g,\mathcal{S}} \stackrel{c}{\equiv} \text{HYBRID}^f_{\pi,\mathcal{A}'_\pi}$.

Proof: The proof is similar to the proof of Claim 7. Assume the claim does not hold. Then there exists a non-uniform polynomial-time distinguisher D, an input (global, z), and a constant $c > 0$ such that for infinitely-many values of k:

$$\begin{aligned}\Big|\Pr\left[D(1^k, \mathsf{global}, z, \text{HYBRID}^f_{\pi,\mathcal{A}'_\pi}(\mathsf{global})) = 1\right] \\ - \Pr\left[D(1^k, \mathsf{global}, z, \text{IDEAL}_{g,\mathcal{S}}(\mathsf{global})) = 1\right]\Big| > k^{-c}.\end{aligned}$$

Let $q(k)$ be the maximum, for security parameter k, of the expected running times of $\mathcal{A}'_\pi$ (with respect to π in the f-hybrid model) and $\mathcal{S}$ (with respect to g); since both of these expected running times are polynomial, $q(k)$ is polynomial as well. Define $\tilde{\mathcal{A}}'_\pi$ to be identical to $\mathcal{A}'_\pi$ except that it halts immediately (with output $\perp$) if it ever exceeds $4q(k)k^c$ steps. Define $\tilde{\mathcal{S}}$ to be identical to $\mathcal{S}$ except that whenever $\mathcal{S}$ (via $\mathcal{S}_\pi$) makes an oracle call to $\mathcal{A}'_\pi$, if the running time of $\mathcal{A}'_\pi$ in answering that query exceeds $4q(k)k^c$ steps we simply answer the query with $\perp$. (Thus, effectively, we are using $\tilde{\mathcal{A}}'_\pi$ as the oracle rather than $\mathcal{A}'_\pi$.) Since the statistical difference between $\text{HYBRID}^f_{\pi,\mathcal{A}'_\pi}$ and $\text{HYBRID}^f_{\pi,\tilde{\mathcal{A}}'_\pi}$ is at most $k^{-c}/4$ (and similarly for $\text{IDEAL}_{g,\mathcal{S}}$ and $\text{IDEAL}_{g,\tilde{\mathcal{S}}}$), we have that for infinitely-many values of k,

$$\begin{aligned}\Big|\Pr\left[D(1^k, \mathsf{global}, z, \text{HYBRID}^f_{\pi,\tilde{\mathcal{A}}'_\pi}(\mathsf{global})) = 1\right] \\ - \Pr\left[D(1^k, \mathsf{global}, z, \text{IDEAL}_{g,\tilde{\mathcal{S}}}(\mathsf{global})) = 1\right]\Big| > k^{-c}/2. \end{aligned} \tag{12}$$

Now, since $\tilde{\mathcal{A}}'_\pi$ runs in *strict* polynomial time, we are guaranteed that

$$\text{IDEAL}_{g,\mathcal{S}_\pi^{\tilde{\mathcal{A}}'_\pi}} \stackrel{c}{\equiv} \text{HYBRID}^f_{\pi,\tilde{\mathcal{A}}'_\pi} \tag{13}$$

and furthermore that $\mathcal{S}_\pi^{\tilde{\mathcal{A}}'_\pi}$ runs in expected polynomial time with respect to g. Note that $\tilde{\mathcal{S}}$ is identical to $\mathcal{S}_\pi^{\tilde{\mathcal{A}}'_\pi}$ except that $\tilde{\mathcal{S}}$ halts if its running time ever exceeds $\alpha(k)/2$ steps. Since $\mathcal{S}_\pi^{\tilde{\mathcal{A}}'_\pi}$ runs in expected polynomial time, this implies that $\text{IDEAL}_{g,\mathcal{S}^{\tilde{\mathcal{A}}'_\pi}_\pi}$ and $\text{IDEAL}_{g,\tilde{\mathcal{S}}}$ are statistically close. But this and Eq. (13) imply

$$\text{IDEAL}_{g,\tilde{\mathcal{S}}} \stackrel{c}{\equiv} \text{HYBRID}^{f}_{\pi,\tilde{\mathcal{A}}'_\pi},$$

contradicting Eq. (12). $\square$

This completes the proof of Theorem 12. $\blacksquare$

References

[1] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[2] B. Barak and O. Goldreich. Universal Arguments and their Applications. *17th IEEE Conference on Computational Complexity*, pages 194–203, 2002.

[3] B. Barak and Y. Lindell. Strict Polynomial-Time in Simulation and Extraction. *SIAM Journal on Computing,* 33(4):783–818, 2004.

[4] P. Billingsley. *Probability and Measure, 2nd edition.* Wiley, New York, 1986.

[5] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145, 2001.

[7] R. Canetti, O. Goldreich, S. Goldwasser and S. Micali. Resettable Zero-Knowledge. In *32nd STOC*, pages 235–244, 2000.

[8] U. Feige. *Alternative Models for Zero Knowledge Interactive Proofs.* Ph.D. Thesis, Weizmann Institute, 1990.

[9] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 526–544, 1989.

[10] O. Goldreich. *Foundations of Cryptography, Volume 1: Basic Tools.* Cambridge University Press, 2001.

[11] O. Goldreich. *Foundations of Cryptography, Volume 2: Basic Applications.* Cambridge University Press, 2004.

[12] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for *NP*. *Journal of Cryptology*, 9(3):167–190, 1996.

[13] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing* 25(1):169–192, 1996.

[14] O. Goldreich, S. Goldwasser and S. Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, 1986.

[15] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing,* 18(1):186–208, 1989.

[16] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but Their Validity or All Languages in *NP* Have Zero-Knowledge Proof Systems. *Journal of the ACM* 38(1):691–729, 1991.

[17] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987. For details see [11].

[18] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology* 7(1):1–32, 1994.

[19] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[20] Y. Lindell. Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation. *Journal of Cryptology*, 16(3):143–184, 2003.

A The Simulator of [12] and Expected Polynomial-Time Verifiers

In this appendix, we repeat the result of [20] showing that the simulator provided for the zero-knowledge proof system of Goldreich and Kahan [12] does *not* necessarily remain expected polynomial time when simulating for an expected polynomial-time verifier. We stress that we do not claim that it is impossible to construct a *different* simulator that will have this property. However, it seems from our analysis below that it would be difficult to construct such a simulator.

For this section, we assume familiarity with the proof system of [12]. Recall that in this proof system, the verifier begins by committing to its random query string (using a perfectly hiding commitment scheme). The parties then continue by running the zero-knowledge proof for 3-coloring of [16] in parallel, using the verifier's queries from the first step. That is, the prover sends (perfectly binding) commitments to randomly permuted colorings of the graph. Then, the verifier decommits, revealing its query string. Finally, the prover answers according to the revealed queries. The exact soundness of the system depends on the number of parallel executions and is negligible. We denote the soundness of the proof system by $\mu(k)$ (i.e., the probability that V accepts and the graph is not 3-colorable is less than $\mu(k)$). We stress that the exact value of $\mu(k)$ can be calculated and this does not depend on any computational assumptions.

Before proceeding, we note that the prover's commitments (to the colorings) are only computationally hiding. Therefore, given enough time, it is possible to break them and extract the committed values (which in this case equals the coloring itself). In particular, in time 2^k (where k is the security parameter), it is possible to break these commitments.

Loosely speaking, we will construct a verifier that with probability 2^{-k} runs for 2^k steps and breaks the prover's commitments. Then, the verifier checks if these commitments are "real" or "convincing garbage", where convincing garbage is a commitment that would convince the verifier, yet does not constitute a legal 3-coloring. Then, if it finds that it received convincing garbage, it enters a very long loop (and otherwise continues like the honest verifier). The key point is that although the simulator can generate convincing garbage, the probability that any (even all-powerful) machine can do the same is negligible. Therefore, when interacting in a real protocol execution, the verifier enters the loop with very small probability. On the other hand, the simulator *always* generates convincing garbage. By correctly choosing the number of steps run by the verifier in the loop, we can ensure that its overall expected-time during simulation is superpolynomial. Note that this verifier strategy is expected polynomial time in any interaction (and thus also with respect to the protocol). We now describe the expected polynomial-time verifier V^* in detail:

The Verifier V^*:

1. Send the prover a perfectly-hiding commitment to a random query string q, exactly according to the protocol specification.

2. Upon receiving the prover's commitments (to many 3-colorings) do the following:

   - With probability 2^{-k}, break the prover's commitments and obtain the values. (This takes time at most 2^k.)

     If the commitments are such that *none* of them constitute a valid 3-coloring, yet they all answer the query string q perfectly,[15] then run for $2^k/\mu(k)$ steps.

3. Continue in the same way as the honest verifier.

We first claim that V^* runs in expected polynomial time in any interaction. This can be seen as follows. V^* attempts to break the commitments with probability 2^{-k}. Therefore, the 2^k time it takes to do this contributes only a single step to its expected running time. Furthermore, the probability that any machine sends a commitment of the form that causes V^* to run for $2^k/\mu(k)$ steps is at most $\mu(k)$ (by the soundness of the proof system). Therefore, V^* runs for $2^k/\mu(k)$ steps only with probability $2^{-k} \cdot \mu(k)$ and this also contributes only a single step to its expected running time. That is, the expected running time of V^* is at most:

$$\frac{1}{2^k} \cdot \left(2^k + \mu(k) \cdot \frac{2^k}{\mu(k)} + p(k)\right) + \left(1 - \frac{1}{2^k}\right) \cdot p(k) = \text{poly}(k)$$

where $p(k)$ equals the running time of the honest verifier.

Next, we claim that the expected running time of the simulator of [12] is superpolynomial when simulating for this V^*. This is because the simulator of [12] *always* sends a commitment that causes V^* to run in time $2^k/\mu(k)$ (in the event that the verifier breaks open the commitments). Therefore, the expected running time of the simulator of V^* is greater than:

$$\frac{1}{2^k} \cdot \left(2^k + 1 \cdot \frac{2^k}{\mu(k)} + p(k)\right) + \left(1 - \frac{1}{2^k}\right) \cdot p(k) > \frac{1}{\mu(k)}$$

Since $\mu(k)$ is a negligible function, we have that the expected running time of the simulator is superpolynomial. Therefore, the simulator presented by [12] for demonstrating the zero-knowledge property of their proof system does not necessarily run in expected polynomial time, if the verifier runs in expected polynomial time.

[15] A commitment answers the query string perfectly if for every edge in the query string, it turns out that the committed colors of the vertices specified by the edge are different. Therefore, such a commitment would convince the honest verifier in the proof.

B Counterexample for the Case of Modular Composition

In this section, we show that the *proof* of Canetti [5] does not necessarily hold when expected polynomial-time simulation strategies are used for real adversaries that run in strict polynomial time. We stress that our example does not show that the modular composition theorem fails to hold in this setting (indeed, we prove such a composition theorem in Section 4 under reasonable assumptions), but merely indicates that a different analysis than that appearing in [5] is necessary.

Our counterexample is comprised of an idiotic functionality, an idiotic protocol, and an idiotic simulation strategy. Nevertheless, this suffices for justifying a different analysis than that used in [5]. Somewhat more natural examples can be presented, although the more natural examples known to us are still rather artificial. Our example below consists of an *outer* protocol π and an *inner* protocol ρ, where the outer protocol calls the inner one. We begin by describing the inner protocol.

The inner functionality f, protocol ρ, and simulator $\mathcal{S}_\rho$. We consider a two-party ideal functionality f that receives no input and generates no output. Consider the following protocol ρ that securely computes f:

1. Party P_2 chooses a random string $r_2 \in \{0,1\}^k$ and sends it to party P_1.
2. Party P_1 receives r_2 from P_2. It then sends a random string $r_1 \in \{0,1\}^k$ to P_2.

This concludes the protocol. Since f receives no input and generates no output, every protocol securely computes f and therefore so does ρ. Nevertheless, we will use the following black-box ideal-model simulator $\mathcal{S}_\rho$ (we consider only the case where P_2 is corrupted):

1. Let R_a, R_b denote the first and second k bits of $\mathcal{S}_\rho$'s random tape, respectively.
2. Upon receiving a string r_2 from the corrupted P_2, simulator $\mathcal{S}_\rho$ checks if $r_2 = R_a$. If yes, $\mathcal{S}_\rho$ runs for 2^k steps, hands P_2 the string $r_1 = R_b$, and outputs whatever P_2 outputs. Otherwise, $\mathcal{S}_\rho$ hands P_2 the string $r_1 = R_a$ and outputs whatever P_2 outputs.

Oracle machine $\mathcal{S}_\rho$ runs in expected polynomial time because the probability that $r_2 = R_a$ is at most 2^{-k}. Furthermore, $\mathcal{S}_\rho$ provides a statistically-close simulation for ρ.

The outer functionality g, protocol π, and simulator $\mathcal{S}_\pi$. The functionality g is the same as f and does not receive any input or generate any output. However, the outer protocol π is different, and *calls* the inner functionality f. The protocol description of π is as follows:

1. Party P_1 chooses a random string $s_1 \in \{0,1\}^k$ and sends it to party P_2.
2. Parties P_1 and P_2 both call the ideal functionality f.
3. Party P_2 chooses a random string $s_2 \in \{0,1\}^k$ and sends it to P_1.

As before, it is clear that π securely computes g (because every protocol does). Nevertheless, we will use the specific black-box simulator $\mathcal{S}_\pi$ that acts in the following "strange" way. (Again, we deal only with the case where P_2 is corrupted). $\mathcal{S}_\pi$ executes π three times:

1. The first time:
   (a) $\mathcal{S}_\pi$ sends a random string $s_1 \in \{0,1\}^k$ to the corrupted P_2.

(b) $\mathcal{S}_\pi$ "calls" the ideal functionality f. (Note that this call is made by $\mathcal{S}_\pi$ within a simulated execution of the protocol π, and is *not* being made to the external trusted party computing g.)

(c) $\mathcal{S}_\pi$ receives s_2 from P_2.

2. $\mathcal{S}_\pi$ rewinds the adversary and runs π a second time:

   (a) $\mathcal{S}_\pi$ sends the string $s'_1 = s_2$ to P_2 (where s_2 is the string that it received from P_2 previously).

   (b) $\mathcal{S}_\pi$ calls the ideal functionality f.

   (c) $\mathcal{S}_\pi$ receives s'_2 from P_2.

3. $\mathcal{S}_\pi$ rewinds the adversary and runs π a third time. This time $\mathcal{S}_\pi$ behaves exactly as it did the first time, using a fresh random $s''_1 \in \{0,1\}^k$. Finally, $\mathcal{S}_\pi$ outputs whatever P_2 outputs.

$\mathcal{S}_\pi$ is a good simulator for π because the view of P_2 in the third run of π with $\mathcal{S}_\pi$ is exactly the same as its view in a real execution with P_1. Furthermore, $\mathcal{S}_\pi$ runs in strict polynomial time.

Composing π with ρ. We now show that the simulator for the composed protocol π^ρ obtained by the proof of [5] does not run in expected polynomial time. This holds even for a real adversary running in strict polynomial time. Intuitively, the reason for this is that the outer protocol can be used to "leak" information to the inner protocol.

We begin by describing a real adversary $\mathcal{A}$ who controls the corrupted P_2 and runs the composed protocol π^ρ with the honest P_1. Adversary $\mathcal{A}$ receives the first message s_1 of π from P_1. Then, in the first message of ρ (that is run next), $\mathcal{A}$ sends $r_2 = s_1$ (where s_1 is the message it received from P_1). $\mathcal{A}$ then receives r_1 from P_1, concluding the inner protocol ρ. Finally, $\mathcal{A}$ concludes the execution of π by sending P_1 the second message $s_2 = r_1$ (where r_1 is the message it received from P_1 in ρ). Note that $\mathcal{A}$ essentially relays P_1's message from the outer protocol to the inner protocol, and P_1's message from the inner protocol to the outer one. $\mathcal{A}$ runs in strict polynomial time, as required.

Let us now step through the simulation strategy of Canetti [5]. (See the discussion at the very beginning of the proof of Theorem 12.) First construct the strict polynomial-time adversary $\mathcal{A}_\rho$ which, essentially, takes as auxiliary input some state s_1; outputs s_1 as its first ρ-message r_2; receives the second ρ-message r_1; and concludes by outputting final state r_1. Next, consider the ideal-world adversary/simulator $\mathcal{S}_\rho^{\mathcal{A}_\rho}$ that interacts with a trusted party computing f. This simulator runs in expected polynomial time.

1. (Running $\mathcal{A}$:) Receive s_1 from P_1. The state of $\mathcal{A}$ is set to s_1.

2. Make an ideal call to f and run $\mathcal{S}_\rho^{\mathcal{A}_\rho(s_1)}(R_a, R_b)$. In more detail: (1) receive $r_2 = s_1$ from $\mathcal{A}_\rho$; (2) if $s_1 = R_a$, run for 2^k steps and give $\mathcal{A}_\rho$ the message $r_1 = R_b$; otherwise, give $\mathcal{A}_\rho$ the message $r_1 = R_a$. Then, (3) $\mathcal{A}_\rho$ outputs state r_1.

3. (Running $\mathcal{A}$ using state r_1:) Send $s_2 = r_1$ to P_1.

Figure 4: Execution of $\mathcal{A}_\pi$ using random tape R_a, R_b.

Then construct the expected polynomial-time adversary $\mathcal{A}_\pi$ interacting with parties running π in the f-hybrid model. See Figure 4. Finally, the simulator for π^ρ is obtained by running $\mathcal{S}_\pi^{\mathcal{A}_\pi}$ using random coins s_1, s_1'' for the black-box simulator $\mathcal{S}_\pi$ and random coins R_a, R_b for the adversary $\mathcal{A}_\pi$. (We stress that these coins are *fixed* throughout the entire execution.)

Note, however, what happens in an execution of $\mathcal{S}_\pi^{\mathcal{A}_\pi(R_a,R_b)}(s_1, s_1'')$:

1. The first iteration of $\mathcal{S}_\pi$:
   (a) $\mathcal{S}_\pi$ sends s_1 to $\mathcal{A}_\pi$ (cf. step 1 of $\mathcal{A}_\pi$).
   (b) $\mathcal{S}_\pi$ calls the ideal functionality f (cf. step 2 of $\mathcal{A}_\pi$). As a consequence, $\mathcal{A}_\rho$ outputs state $r_1 = R_a$ except with probability 2^{-k}.
   (c) $\mathcal{S}_\pi$ receives $s_2 = r_1 = R_a$ from $\mathcal{A}_\pi$ (cf. step 3 of $\mathcal{A}_\pi$).
2. The second iteration of $\mathcal{S}_\pi$:
   (a) $\mathcal{S}_\pi$ sends the string $s_1' = s_2 = r_1 = R_a$ to $\mathcal{A}_\pi$.
   (b) $\mathcal{S}_\pi$ calls the ideal functionality f. As a consequence, $\mathcal{A}_\rho$ "responds" with $s_1' = R_a$ and then $\mathcal{A}_\pi$ runs for 2^k steps (cf. step 2 of $\mathcal{A}_\pi$).
   (c) The simulation continues...

The remainder of the simulation is not important because we have already demonstrated that the simulation always runs for 2^k steps.

Although the above example is truly idiotic, it demonstrates a real issue in simulation proofs that use rewinding. Specifically, the internal state of the adversary ($\mathcal{A}_\pi$ in the above example) is learned during rewinding. If this internal state is used in a detectable way at a later stage of the simulation, it can cause the overall simulation to run for a superpolynomial number of steps.

C The Full Derivation of Equation (2)

In the proof of Claim 6, we state that

$$\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau}\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] = \sum_{j=1}^{\infty}\mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(j) \mid \tau \geq j\right] \cdot \Pr_{r,s,r_f}[\tau \geq j].$$

A full derivation of this equality follows:

$$\begin{aligned}
\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau}\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] &= \sum_{j=1}^{\infty}\Pr_{r,s,r_f}[\tau = j]\cdot\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{j}\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\,\middle|\,\tau = j\right]\\
&= \sum_{j=1}^{\infty}\Pr_{r,s,r_f}[\tau = j]\sum_{i=1}^{j}\mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau = j\right]\\
&= \sum_{i=1}^{\infty}\sum_{j=i}^{\infty}\Pr_{r,s,r_f}[\tau = j]\cdot\mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau = j\right],
\end{aligned}$$

where the second equality is by the linearity of expectations, and the third is obtained by simply re-arranging terms. Fix an arbitrary i. Using the definition of expectation and then making a series of straightforward re-arrangements we obtain:

$$\begin{aligned}
&\sum_{j=i}^{\infty} \Pr_{r,s,r_f}[\tau = j] \cdot \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau = j\right] \\
&= \sum_{j=i}^{\infty} \Pr_{r,s,r_f}[\tau = j] \cdot \sum_{t=0}^{\infty} t \cdot \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) = t \mid \tau = j\right] \\
&= \sum_{t=0}^{\infty} t \cdot \sum_{j=i}^{\infty} \Pr_{r,s,r_f}[\tau = j] \cdot \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) = t \mid \tau = j\right] \\
&= \sum_{t=0}^{\infty} t \cdot \sum_{j=i}^{\infty} \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) = t \wedge \tau = j\right] \\
&= \sum_{t=0}^{\infty} t \cdot \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) = t \wedge \tau \geq i\right] \\
&= \sum_{t=0}^{\infty} t \cdot \Pr_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) = t \mid \tau \geq i\right] \cdot \Pr_{r,s,r_f}[\tau \geq i] \\
&= \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau \geq i\right] \cdot \Pr_{r,s,r_f}[\tau \geq i].
\end{aligned}$$

Since the above holds for arbitrary i, we conclude that

$$\begin{aligned}
\mathbf{Exp}_{r,s,r_f}\left[\sum_{i=1}^{\tau} \mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i)\right] &= \sum_{i=1}^{\infty}\sum_{j=i}^{\infty} \Pr_{r,s,r_f}[\tau = j] \cdot \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau = j\right] \\
&= \sum_{i=1}^{\infty} \mathbf{Exp}_{r,s,r_f}\left[\mathsf{simtime}_{\hat{\mathcal{A}}(z,r)}(i) \mid \tau \geq i\right] \cdot \Pr_{r,s,r_f}[\tau \geq i]
\end{aligned}$$

which is the same as Eq. (2) (except that the above has "i" instead of "j").