SIAM J. Comput.
Vol. 39, No. 5, pp. 2090–2112

INFORMATION-THEORETICALLY SECURE PROTOCOLS AND SECURITY UNDER COMPOSITION[*]

EYAL KUSHILEVITZ[†], YEHUDA LINDELL[‡], AND TAL RABIN[§]

Abstract. We investigate the question of whether the security of protocols in the information-theoretic setting (where the adversary is computationally unbounded) implies the security of these protocols under concurrent composition. This question is motivated by the folklore that all known protocols that are secure in the information-theoretic setting are indeed secure under concurrent composition. We provide answers to this question for a number of different settings (i.e., considering perfect versus statistical security, and concurrent composition with adaptive versus fixed inputs). Our results enhance the understanding of what is necessary for obtaining security under composition, as well as providing tools (i.e., composition theorems) that can be used for proving the security of protocols under composition while considering only the standard stand-alone definitions of security.

Key words. theory of cryptography, secure multiparty computation, information-theoretic security, security under composition

AMS subject classifications. 94A60, 68Q17, 68Q25

DOI. 10.1137/090755886

1. Introduction.

1.1. Background.

In the setting of secure multiparty computation, a set of parties with private inputs wishes to jointly compute some function of their inputs. Loosely speaking, the security requirements of such a computation are that nothing is learned from the protocol other than the output (privacy), and that the output is distributed according to the prescribed functionality (correctness). More exactly, the result of an execution of a secure protocol must be like the result of an "ideal execution" with a trusted party who honestly computes the function for the parties (cf. [7] or [18, section 7.1]). These security requirements must hold in the face of a malicious adversary who controls some subset of the parties and can arbitrarily deviate from the protocol instructions. We focus on secure function evaluation here (although we do not know of any difference between this setting and that of reactive functionalities). Our results hold for both *static* and *adaptive* corruptions.

Secure multiparty computation has been studied in a number of different scenarios. One important distinction relates to the power of the adversary, yielding two main settings.

1. *The information-theoretic setting.* In this setting, the adversary is computationally unbounded. Thus, security here does not rely on any unproven complexity assumptions; rather, it is "information theoretic." There are two levels of security that have been considered here.

[*]Received by the editors April 13, 2009, accepted for publication (in revised form) December 20, 2009; published electronically March 5, 2010. An extended abstract of this work appeared in Proceedings of the 38th Annual ACM STOC, 2006.
http://www.siam.org/journals/sicomp/39-5/75588.html

[†]Department of Computer Science, Technion Israel Institute of Technology, Haifa 32000, Israel (eyalk@cs.technion.ac.il). This author's research was supported by grant 36/03 from the Israel Science Foundation.
[‡]Department of Computer Science, Bar-Ilan University, Ramat Gan, 52900, Israel (lindell@cs.biu.ac.il).
[§]IBM T.J. Watson Research, Yorktown Heights, NY 10598 (talr@us.ibm.com).

   (a) *Perfect security.* Very informally, here the result of a real execution of the protocol with a real adversary must be exactly the same as the result of an ideal execution with a trusted party and an ideal-world adversary/simulator.
   (b) *Statistical security.* Here, the result of the real protocol execution need "only" be statistically close to the result of an ideal execution.

   It has been shown that, assuming that more than two thirds of the parties are honest (or assuming a simple majority if the parties are additionally given a broadcast channel), it is possible to securely compute any functionality [5, 10, 28, 2]. We note that these results assume that the parties can communicate via perfectly secure communication channels.
2. *The computational setting.* In this setting, the adversary is assumed to run in probabilistic polynomial-time, and the security of protocols typically relies on the assumed hardness of some problem (such as factoring a large composite into its prime factors). Under appropriate cryptographic assumptions, it has been shown that any functionality can be securely computed, even if an overwhelming majority of the parties are corrupted [29, 19].[1]

Another important distinction relates to the setting in which the protocol is executed. We relate here two possible scenarios.

1. *The stand-alone model.* In this setting, the secure protocol is executed only once, and it is assumed that this is the only protocol being executed. This was the standard setting for analyzing protocols initially; e.g., the results of [29, 19, 5, 10, 28, 2], cited above, were all obtained in this setting.
2. *Security under composition.* In this setting, a protocol is executed many times, possibly alongside other (secure and insecure) protocols. Security in such cases is referred to as *security under composition.* There are many different types of composition, and we will mention two here.
   (a) *Concurrent general composition (a.k.a. universal composition).* In this setting, a protocol is run many times in an arbitrary network, possibly alongside other (secure and insecure) protocols. *Concurrent* composition implies that the scheduling of the delivery of all messages sent (even messages sent between honest parties) is fully controlled by the adversary. See, e.g., [12, 27, 8].
   (b) *Concurrent self-composition.* In this setting, a single protocol is run many times concurrently (with itself) in a network. See, e.g., [15, 13, 14]. Again, the scheduling of the delivery of all messages sent is under the control of the adversary.

   The question of security of protocols under concurrent composition has received a lot of attention recently (see, e.g., [14, 12, 27, 8, 23, 25, 24] and much more).

It was believed that all protocols (or at least, all known protocols) that are secure in the information-theoretic setting, even when proven secure only in the stand-alone model, are secure under concurrent composition. In this paper, we study this belief and more generally the interesting connection between information-theoretic security and security under composition. Our aim in initiating this study is twofold. First, as our results demonstrate, understanding this connection deepens our understanding of what is required for obtaining security under composition. Second, due to its complex

[1]When no honest majority is assumed, the definition of security is slightly relaxed so that fairness is not required.

nature, the task of proving the security of protocols under composition is a difficult one and we obtain a number of results that simplify this task.

1.2. Our results. As we have mentioned, the folklore prior to our work seems to have been that all protocols that are secure in the information-theoretic stand-alone setting are also secure under concurrent general composition. The reason for this folklore appears to be based on the fact that all known protocols in the information-theoretic setting are proven secure using a straight-line black-box simulator,[2] and the existence of such a simulator was believed to suffice for proving the security of a protocol under concurrent general composition. Indeed, it is common to hear the claim that universal composability (that implies security under concurrent general composition) is essentially the same as stand-alone security with the exception that the simulator is not allowed to rewind the adversary. We begin by showing that this folklore belief is false. That is, we obtain the following, informally stated, proposition.

THEOREM 1.1 (counterexample—informal). *There exist protocols that are statistically secure in the stand-alone information-theoretic model and are proven secure using straight-line black-box simulation, and yet are not secure under concurrent general composition.*

The idea behind the proof of Theorem 1.1 is quite simple and is due to the issue of "delayed inputs." Namely, in the setting of concurrent general composition, the parties run completely asynchronously and so some parties may begin a protocol execution before others begin. In fact, some parties may begin a protocol execution before others have even determined their input to that execution. This results in a possible scenario whereby the first message that one party sends in a protocol execution can be set as the input of another party to that execution. If the first message of the parties has high entropy, then in the stand-alone setting one party's first message will equal another party's input with only negligible probability. Thus, one can construct a protocol that will fail if this happens (and so will not be secure under concurrent general composition), but is otherwise statistically secure with straight-line simulation.

Although Theorem 1.1 demonstrates that stand-alone information-theoretically secure protocols are not necessarily secure under concurrent general composition, the fact that the source of the problem seems to be "delayed inputs" (as described above) implies that there may be a simple way to circumvent the problem. In addition, the proof of Theorem 1.1 relies inherently on the ability of the stand-alone protocol to fail (albeit with negligible probability). This raises the question as to whether or not delayed inputs are problematic in the setting of *perfect* security. We consider these questions and more.

Perfect security. We prove the following theorem regarding perfectly secure protocols.

THEOREM 1.2 (perfect security—informal). *Every protocol that is perfectly secure in the stand-alone model, and has a straight-line black-box simulator, is secure under concurrent general composition.*

The intuition behind the proof of Theorem 1.2 is that if a protocol is not secure under concurrent general composition, then there must be some setting of inputs and

[2]A black-box simulator is one that works while being given only *oracle access* to the adversary; see [17, section 4.5] for a detailed definition. Such a simulator is straight-line if it interacts with the adversary in the same way as real parties. That is, it first sends the adversary all of the messages it expects to receive from the honest parties in the first round. It then proceeds round by round, without ever going back.

random tapes for the honest parties in the concurrent setting for which the protocol "fails" (i.e., does not behave as it should). Although such a "fail event" may happen with only negligible probability in the stand-alone setting, this is enough to contradict perfect security that requires that protocols *never* fail.

Contrasting Theorems 1.2 and 1.1, we uncover a fundamental gap between the statistical and perfect notions of security; namely, perfectly secure protocols have a *real advantage* over statistically secure protocols, and so perfect security is not just an issue of aesthetics. To the best of our knowledge, this is the first example of such a separation.

Beyond its theoretical interest, Theorem 1.2 provides a useful tool for proving the security of protocols under concurrent composition. Specifically, it suffices to consider that the standard stand-alone definitions and security in the far more complex setting of concurrent composition can be derived as long as a perfect, black-box straight-line simulator is constructed. Thus, a corollary of Theorem 1.2 is that the protocol of [5] is secure under concurrent general composition.[3]

We remark that, under strictly more stringent conditions, a theorem similar to Theorem 1.2 was proven in [12], and that the theorem of [12] is also sufficient to derive the security of [5] under concurrent general composition. See section 1.3 for a more detailed comparison of their work with ours.

Fixed inputs. As we have seen, *statistical* security under straight-line black-box simulation is insufficient for obtaining security under concurrent general composition. Furthermore, our proof of this relies on the fact that one party's input can be derived from another party's behavior in the protocol. This raises the question as to whether straight-line black-box simulation suffices when inputs cannot be dynamically chosen, and in particular in the setting of *fixed inputs* where each honest party receives all of its inputs before any execution begins. That is, at the onset, each party receives a vector of inputs $\mathbf{x} = (x_1, x_2, \ldots)$ and uses x_i as its input to the ith protocol execution. (We note that despite the fact that this looks like a very relaxed notion of security, until now there was no known separation between security under general composition with fixed or with adaptively chosen inputs.) We show that security under straight-line black-box simulation *does suffice* for obtaining security under concurrent general composition with "fixed inputs." That is, we prove the following theorem.

THEOREM 1.3 (fixed inputs—informal). *Every protocol that is secure in the stand-alone model, and has a straight-line black-box simulator, is secure under concurrent general composition with fixed inputs.*

Theorem 1.3 holds for computational, statistical, and perfect security. Therefore, as a corollary to Theorem 1.3, we obtain that the protocols of [10, 28, 11]—which have straight-line black-box simulators—are all secure under concurrent general composition with fixed inputs. Also, by combining Theorem 1.3 with Theorem 1.1, we obtain the first separation between security under general composition with fixed inputs and with adaptively chosen inputs. That is, we have the following corollary.

COROLLARY 1.4. *There exist protocols that are secure under concurrent general composition with fixed inputs, and are not secure under concurrent general composition with adaptively chosen inputs.*

Since general composition implies self-composition, we have that every protocol

[3]The protocol of [5] was designed for the synchronous setting whereas, in the setting of concurrency, the network is essentially asynchronous [6, 16, 4]. Thus, what we really refer to here is an appropriately modified version of [5]; see section 2 for a detailed discussion on the issue of synchronicity.

that is secure in the stand-alone model, and has a straight-line black-box simulator, is secure under *concurrent self-composition with fixed inputs.* This is interesting for historical reasons as the classic problem of concurrent zero-knowledge [14] is exactly in the setting of concurrent self-composition with fixed inputs.

Theorem 1.3 states that any protocol that is proven secure with a straight-line black-box simulator is secure under concurrent general composition with fixed inputs (and therefore also secure under concurrent self-composition with fixed inputs). Since self-composition is a more restricted setting, this begs the question as to whether straight-line black-box simulation—without fixed inputs—suffices for achieving concurrent *self*-composition. However, by the equivalence shown in [24] (stating that concurrent self- and general composition are almost equivalent for adaptively chosen inputs), this is clearly not possible because, as stated in Theorem 1.1, black-box straight-line simulation does *not* suffice for concurrent general composition.

Input availability/start synchronization. Finally, we ask whether there is a simple property of protocols that can be required, in addition to a protocol having a straight-line black-box simulator, so that it will be secure under concurrent general composition. We have shown that requiring fixed inputs, i.e., that all inputs to *all* invocations of the executed protocols be set prior to the start of *any* protocol, yields security under concurrent general composition. However, this is a very restrictive requirement that applies to the invocations and not to the protocol, and it severely limits the applicability of the protocols. We therefore ask whether it is possible to achieve concurrent general composition by imposing a less restrictive (and more realistic) property on the protocol. Our aim is to define a minimal property that circumvents the problem of "delayed inputs," with the hope that this suffices for achieving security under composition.

We show that it is sufficient to require that the inputs of all parties participating in each *specific* invocation of a protocol be fixed before this execution begins. We call this property input availability. We stress that the inputs need not be fixed before *all* protocol executions begin (as in the assumption of fixed inputs); rather, we require that the inputs of all parties to *each specific* execution be fixed before that execution begins. Formally, this holds if all honest parties have read (and fixed) their input to an execution before any of the honest parties sends the first message of the protocol. In order to see why this seemingly small modification makes a difference, recall Theorem 1.3, which states that the existence of a straight-line black-box simulator is sufficient to demonstrate security under composition as long as the parties' inputs are fixed before any execution begins. Now, if the parties' inputs are not fixed at the onset, but are fixed before a specific execution begins, then it is still possible to view everything that happened before the specific execution as *auxiliary input* that determines the parties' inputs to that execution. This enables us to reduce the security of the specific execution to the stand-alone setting, while the auxiliary input is used to emulate all the other concurrent executions. Stated differently, when input availability holds, the setting is essentially the same as that of fixed inputs (cf. Theorem 1.3), and so security under concurrent general composition holds.

We stress that when input availability does *not* hold, it is not possible to emulate all concurrent executions outside of a specific execution using an auxiliary input. This is because, by definition, auxiliary input is fixed before the protocol execution begins and thus must be *independent* of the random tapes of the parties. However, if a party's input is determined only after other parties have already begun the execution, then there is no guarantee of independence. Indeed, this is exactly how we prove the counterexample of Theorem 1.1.

Importantly, it is possible to ensure that the above property holds using a simple protocol preamble that we call start synchronization. Specifically, when a party has received its input to an execution and is ready to begin executing, it broadcasts a message to all other parties that it is ready to begin. The parties then only commence the actual execution *after* they have heard from all other parties that they are ready. This has the effect of ensuring that the actual protocol execution begins only after all parties' inputs have already been fixed (because no party begins until it hears from all other parties that they are ready to start, meaning that they already have input). This modification captures the essence of what is needed for the composition, which is that the inputs for each single execution of a protocol be known prior to the start of the execution. Thus, we have the following theorem.

THEOREM 1.5 (black-box straight-line and start synchronization—informal). *Every protocol that is secure in the stand-alone model and has start synchronization and a straight-line black-box simulator is secure under concurrent general composition.*

We remark that Theorem 1.5 holds for computational, statistical, and perfect security. Since any protocol can easily be modified so that it has start synchronization, we achieve the following corollary (informally stated).

COROLLARY 1.6. *Assuming a broadcast channel and an honest majority, every (standard, nonreactive) functionality can be computed by a statistically secure protocol under concurrent general composition.*

Note that Corollary 1.6 is stated in terms of *functionalities* while our other results are stated as results about *protocols*; the corollary is obtained by applying Theorem 1.5 to the protocols of [28, 2], once modified by adding start synchronization. We remark that this is the first proof of the existence of such protocols. The main novelty of this corollary is that it is the first proof that there exist protocols that achieve fairness in the setting of concurrent general composition, assuming an honest majority (previously, this was known only for the case that more than two thirds of the parties are honest). The fact that the protocol of [28] is universally composable and thus secure under concurrent general composition was claimed with a high-level proof sketch in an early version of [8].

Summary. Table 1 below summarizes our results; the security level "all" means that the result holds for computational, statistical, *and* perfect security. We remark that the proof of Theorem 1.3 is actually a simple special case of the proof of Theorem 1.5. We therefore first prove Theorem 1.5 (in section 6) and only then Theorem 1.3 (in section 7).

TABLE 1
A summary of our results.

Theorem	**Simulator**	**Security level**	**Composition**
Theorem 1.1	straight-line, black-box	statistical or computational	*not necessarily* secure under concurrent general composition
Theorem 1.2	straight-line, black-box	perfect	secure under concurrent general composition
Theorem 1.3	straight-line, black-box	all	secure under concurrent general composition with fixed inputs
Theorem 1.5	straight-line, black-box	all + start synchronization	secure under concurrent general composition

A definitional discussion. A fundamental difference between the definitions of security in the stand-alone setting and security under composition via *universal com-*

posability [8] is the question of whether the distinguisher is *passive* and views the result of the execution after it terminates (as in the stand-alone setting) or whether it is *interactive* and views the intermediate stages of the computation as well (as in universal composability). (Note that although we formally use the definition of concurrent general composition, and not universal composability, these are known to be equivalent [25].) In this light, it is possible to technically interpret our results as a study of the relation between security under passive and interactive distinguishers, and not as a study of the relation between stand-alone security and security under composition. Nevertheless, the equivalence of universal composability (a stand-alone definition with an interactive distinguisher) and concurrent general composition (a definition that considers composition and a passive distinguisher) demonstrates that these perspectives are essentially the same.

1.3. Related and subsequent work. As we have mentioned, the question of composition of information-theoretic protocols was previously studied in [12]. Those authors showed that any protocol that is secure under their definition (which is a slight modification of the definition of [26]) is secure under concurrent general composition. There are three main requirements in the definition considered in [12]: (a) the protocol must be perfectly secure; (b) a straight-line black-box simulator must be used; and (c) there must be a fixed "committal round" at which point all of the parties' inputs are fully defined by the protocol traffic, but no parties have yet received output. In contrast, our proof of Theorem 1.2 requires (a) and (b), but not (c). (We note that the proof of [12] relies explicitly on this property, as stated in [12, section 4.3].) Apart from Theorem 1.2, there is no overlap between our work and the work of [12]. Indeed, they leave the question of statistical security open, and we provide both positive and negative answers (depending on the setting).

An interesting question that arises from our work relates to the necessity (or lack thereof) of straight-line simulation in our results. Specifically, both Theorems 1.2 and 1.3 require that the original black-box simulator be "straight-line". This raises the question as to whether the requirement that the simulator be straight-line is inherent or whether similar theorems could be proven also if the simulator is not straight-line. In the conference version of this paper [22] we claimed that the theorems could be extended to any black-box simulator by showing a method to convert a rewinding polynomial-time simulator into a straight-line exponential-time simulator (even though the resulting simulator is not efficient, this can still be of some interest in the information-theoretic setting). However, our proof was erroneous and it was shown in [1] that there exist perfectly secure protocols (for the stand-alone model) for which *any* black-box simulator must use rewinding and which are *not* secure under concurrent composition even with an inefficient simulator.

We remark that the proof of a similar result to Theorem 1.1 was obtained independently and concurrently in [21]. In [21] it is also proven that any protocol that is perfectly secure under bounded concurrent general composition is also secure under (unbounded) concurrent general composition. We prove a much stronger result here; namely, that it suffices to prove perfect security in the *stand-alone* model.

2. On concurrent composition and the synchronous model.

In this paper, we consider protocols that were proven secure in the stand-alone model, and we analyze their security in the setting of concurrent composition. There are two "standard" modes of message scheduling in the stand-alone setting: *synchronous scheduling* and *asynchronous scheduling*. In the first, all parties send the messages of a given round at the same time, and these messages arrive at their destination instantaneously.[4] In the asynchronous setting [6, 16, 4], the adversary is given full control over the scheduling of the sending and receiving of messages, with the requirement that *all messages be eventually delivered*. Now, in the standard model of concurrent computation [8], the adversary is given full control over the delivery of messages, as in the asynchronous model, but need not ever deliver messages sent by the honest parties. That is, unlike in the asynchronous setting where all messages must eventually be delivered, in the standard concurrent setting the adversary may completely block some of the honest parties and *never* deliver their messages. As such, it is not possible to guarantee output delivery or fairness (at least when concurrency is modeled in this way). Thus, the best that we can hope for in the concurrent setting is *security with abort* (meaning that the adversary may receive output while the honest parties do not), as in the stand-alone setting with no honest majority.[5]

As we have mentioned, in this paper we analyze the security under concurrent composition of protocols that were designed for the stand-alone model. In order to maintain compatibility between these models, we consider the security under concurrent composition of protocols that are *secure with abort in the stand-alone model when the adversary has full control over message delivery*. We stress that this includes the capability of the adversary to completely block messages between honest parties, unlike the stand-alone asynchronous model where all messages must eventually be delivered. Since this is not the standard model for stand-alone computation (neither for the synchronous nor asynchronous model), we need to discuss how to execute these protocols when the adversary has full control including blocking. For the synchronous model, we show below how to convert any secure protocol designed as a stand-alone protocol (with or without abort) into a protocol that is secure *with abort* in the model where the adversary has full control over message delivery (including blocking). The case of the asynchronous setting is more straightforward. The adversary's actions in the concurrent setting (where message blocking is allowed) is actually a prefix of an adversary's actions in the asynchronous setting where all messages must eventually be delivered (this can be seen by viewing a blocked message simply as a message that was not yet delivered). This immediately implies that the security of any protocol designed in the asynchronous setting is not violated by the scheduling of an adversary who can also block messages. We stress, however, that if the adversary does not complete the delivery of messages as required in the asynchronous setting, then output delivery and fairness may not be achieved. This implies that the level of security obtained is still only security with abort. We conclude that asynchronous stand-alone protocols can be executed unmodified in the presence of an adversary who is also allowed to block messages, with the result that security with abort is achieved. We now proceed to describe the transformation for the synchronous protocols.

[4]It is typically assumed that the adversary may receive the messages from the honest parties in any given round before it sends its own. This is called "rushing."

[5]In the asynchronous stand-alone model of secure computation, output delivery and fairness are guaranteed to some extent. Loosely speaking, the definition of [4] allows the adversary to exclude up to t honest parties' inputs from the computation (where t denotes the maximum number of corrupted parties), but guarantees fairness and output delivery for all others. This can be achieved because the adversary must eventually deliver all messages; see [4] for details. (We remark that the exclusion of t parties is possible because the honest parties cannot know whether those t parties are corrupted and will never send their messages or whether they are honest and the adversary has not yet delivered their messages. Thus, they cannot wait for all messages because they may be from corrupted parties and so may never arrive.) In contrast, when no parties' messages need to be eventually delivered, it is not possible to do better in general than security with abort.

Modifying synchronous protocols for the concurrent setting. Protocols that were designed for the stand-alone synchronous model are proven secure based on the assumption that the protocol advances from round to round based on some global clock (or round synchronizer). In particular, it is assumed that all parties send and receive their round i messages before proceeding to round $i+1$. When the adversary has the ability to schedule the messages as it wishes, then some parties may receive all round i messages and therefore proceed to send their round $i+1$ messages before others have concluded the previous round. In such a case, the security properties of the protocol may be violated.

In order to solve this problem, we modify the stand-alone synchronous protocol so that no party will proceed to round $i+1$ before all parties conclude round i, irrespective of when and if messages are delivered. We achieve this by adding a "synchronization step" between each round of the protocol. Specifically, after an honest party has sent its round i messages, it is instructed to wait for all of its incoming round i messages. Then, upon receiving all round i messages, the party broadcasts a notification to all other parties that it has completed round i. It then continues to wait to receive notifications that *all* parties have completed round i. Once all notifications are received, the parties proceed to round $i+1$. (We can also assume that honest parties concatenate the round index to every message they send so that the adversary cannot replay messages from previous rounds.) Such modifications can be trivially made to every protocol that is secure *with or without abort* in the synchronous setting, and the result is a protocol that remains *secure with abort* when the adversary has full control over message delivery. (We stress that when applying the transformation to a protocol that is secure without abort, meaning that fairness and output delivery are guaranteed, the result is still a protocol that is only secure with abort. Thus, in all cases, fairness and output delivery are no longer guaranteed since the adversary can mount a simple denial-of-service attack by not forwarding a parties' messages.)

3. Definitions and notation.

In this paper we study the connection between a number of different notions of security for multiparty computation. Specifically, we refer to information-theoretic security (perfect and statistical) and to computational security. In addition, we consider stand-alone computation (where only a single protocol execution takes place) versus concurrent general composition. Finally, we consider the restriction to the case that all inputs are fixed ahead of time.

In this section, we present the definitions and terminology that we use in this paper. We provide only brief outlines of the definitions and refer the reader to [18, Chapter 7] for definitions of secure multiparty computation in the stand-alone setting, and [25] for the definition of security under concurrent general composition.

Preliminaries. A function μ is negligible if for every polynomial p there exists an integer N such that for every $k > N$ it holds that $\mu(k) < 1/p(k)$. Two distribution ensembles $\{X(k,a)\}_{k\in\mathsf{N},a\in\{0,1\}^*}$ and $\{Y(k,a)\}_{k\in\mathsf{N},a\in\{0,1\}^*}$ are computationally indistinguishable, denoted $\{X(k,a)\} \stackrel{\rm c}{\equiv} \{Y(k,a)\}$, if for every nonuniform polynomial-time distinguisher D there exists a negligible function μ such that for all $a \in \{0,1\}^*$ and $k \in \mathsf{N}$,

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

Two distribution ensembles as above are statistically close, denoted $\{X(k,a)\} \stackrel{\rm s}{\equiv} \{Y(k,a)\}$, if for every nonuniform distinguisher (not necessarily polynomial-time) there exists a negligible function μ such that for all a and k,

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

We write $\{X(k,a)\} \equiv \{Y(k,a)\}$ if the distributions are identical.

Secure multiparty computation. In the setting of secure multiparty computation, n parties, denoted $P_1, \ldots, P_n$, wish to jointly compute some function f of their inputs in a secure fashion. The standard way of defining security in this scenario is to require that a real protocol "behave" just like an ideal execution involving a trusted third party who computes the function for the parties [20, 3, 26, 7]. More specifically, an ideal execution is defined in which the parties simply hand their inputs to a trusted third party, who then computes the function and returns the designated outputs to each party. When considering security with abort, the trusted party first hands the output to the adversary, who may then decide whether or not the honest parties also receive output [18, Chapter 7]. A protocol is said to be secure with abort if for every adversary attacking a real execution of the protocol, there exists an ideal-world adversary (running in an ideal execution) such that the output of the honest parties and the ideal-world adversary in the ideal execution is "close" to the output of the honest parties and the real adversary in the real protocol execution. As we have mentioned, the real model that we consider in this paper is one where the adversary has full control over the delivery of messages between parties, including the ability to never deliver messages.

There are a number of different settings for this definition. In the information-theoretic setting, the real-world adversary and ideal-world adversary are computationally unbounded. Here, there are two notions of security: perfect security, where the outputs of the ideal model are required to be *identically distributed* to the outputs of the real model, and statistical security, where the outputs of the ideal model are required to be *statistically close* to the outputs of the real model. We say that a protocol computes a function f with perfect security (resp., statistical security) if it meets the requirements in the information-theoretic setting (with the appropriate notion of "closeness"; identical distribution for perfect security and statistical closeness for statistical security). In the computational setting, the real-world and ideal-world adversaries run in probabilistic polynomial-time and the outputs of the ideal model are required to be *computationally indistinguishable* from the outputs of the real model. In this setting, we say that a protocol computes a function f with computational security.

Security is defined as follows. Let k be the security parameter. We denote a real execution of a protocol ρ with adversary $\mathcal{A}$ by $\text{REAL}_{\rho,\mathcal{A}(z)}(k,\mathbf{x})$, where z is the auxiliary input of $\mathcal{A}$ and $\mathbf{x}$ is the vector of the parties' inputs. Likewise, we denote by $\text{IDEAL}^f_{\mathcal{S}(z)}(k,\mathbf{x})$ an execution in the ideal model where the trusted party computes the function f, $\mathcal{S}$ is the ideal-world adversary (or simulator), and k, z, and $\mathbf{x}$ are as above. For a full specification of the ideal and real executions, see [7] and [18, Chapter 7]. We are now ready to present the definitions.

DEFINITION 3.1 (secure computation).

1. *Information-theoretic security. A protocol ρ* computes f with statistical security and abort *if for every real-model adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ running in time that is polynomial in the running time of $\mathcal{A}$ such that*

$$\left\{\text{IDEAL}^f_{\mathcal{S}(z)}(k,\mathbf{x})\right\}_{k\in\mathsf{N};z,\mathbf{x}\in\{0,1\}^*} \overset{\text{s}}{\equiv} \left\{\text{REAL}_{\rho,\mathcal{A}(z)}(k,\mathbf{x})\right\}_{k\in\mathsf{N};z,\mathbf{x}\in\{0,1\}^*}.$$

Protocol ρ computes f with perfect security and abort *if* $\{\text{IDEAL}^f_{\mathcal{S}(z)}(k,\mathbf{x})\} \equiv \{\text{REAL}_{\rho,\mathcal{A}(z)}(k,\mathbf{x})\}$.

2. *Computational security. A protocol ρ* computes f with computational security and abort *if for every nonuniform probabilistic polynomial-time real-model adversary $\mathcal{A}$ there exists a nonuniform probabilistic polynomial-time ideal-model adversary $\mathcal{S}$ such that*

$$\left\{\text{IDEAL}^f_{\mathcal{S}(z)}(k,\mathbf{x})\right\}_{k\in\mathsf{N};z,\mathbf{x}\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\rho,\mathcal{A}(z)}(k,\mathbf{x})\right\}_{k\in\mathsf{N};z,\mathbf{x}\in\{0,1\}^*}.$$

All of our results in this paper hold for both static and adaptive adversaries. An adaptive adversary can "adaptively" choose which honest parties to corrupt throughout the execution, based on its view so far. In the case of a static adversary, the set of corrupted parties is fixed at the onset.

Black-box straight-line simulation. In our presentation, we will refer to black-box straight-line simulation. Informally speaking, a black-box simulator is a universal ideal-world adversary (i.e., it is a single simulator that works for all real adversaries) that interacts with the real adversary in a black-box way only (i.e., it uses only oracle access to the real adversary). Furthermore, a black-box simulator is straight-line if the real adversary maintains state between oracle calls, in the same way as in a real protocol execution. Stated differently, a black-box straight-line simulator interacts with the real adversary in essentially the same way as real parties in a real protocol interaction (the only difference is that the simulator may simultaneously play the role of many parties). When $\mathcal{S}$ is a black-box simulator, we denote the output of an ideal-model execution by $\text{IDEAL}^f_{\mathcal{S}^{\mathcal{A}(z)}}(k,\mathbf{x})$; note that $\mathcal{S}$ is given oracle access only to $\mathcal{A}$ and in particular is not given the auxiliary input z.[6] This is the standard way of defining black-box simulation; see [17, section 4.5].

Concurrent general composition. In the setting of concurrent general composition, a secure protocol ρ that computes a function f is run concurrently to an arbitrary other protocol π (where π represents many arbitrary protocols running concurrently in the network). In order to define security, a real execution of π with secure protocol ρ is compared to an idealized setting where protocol π is run, but the parties also have access to a trusted party that computes f. Furthermore, instead of running ρ, the parties send their inputs to the trusted party (like in the regular ideal model). This setting is called the hybrid model, because a trusted party is used for computing f, but real messages are also sent for computing π. The security requirement here is that for every real adversary $\mathcal{A}$ and every arbitrary protocol π, there exists an ideal-world adversary $\mathcal{S}$ such that for every set of inputs to π, the output of the ideal-world adversary and the honest parties in an ideal/hybrid execution of π with a trusted party computing f is "close" to the output of the real adversary and the honest parties in a real execution of π with ρ. We stress that the inputs to ρ (or equivalently to f) are determined by π. This models the setting that ρ is run in an arbitrary network (modeled by π) and the inputs to ρ are influenced by other protocol executions that take place in this network. We denote by $\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(\mathbf{x})$ a real execution of ρ with π, with adversary $\mathcal{A}$, inputs $\mathbf{x}$, and auxiliary input z for $\mathcal{A}$. Furthermore, we denote by $\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(\mathbf{x})$ a hybrid execution of protocol π with ideal calls to f, and inputs $\mathbf{x}$

[6]Observe that if $\mathcal{S}$ is given the auxiliary input, then it is no longer really a black-box simulator. In particular, one can define a "universal" adversary $\mathcal{A}$ that receives for auxiliary input the description of an adversary and then runs that adversary. If $\mathcal{S}$ is given the auxiliary input, then it has the strategy of $\mathcal{A}$.

and z as above. Detailed definitions of security under concurrent general composition can be found in [25]. (We remark that although the above refers to a single execution of a protocol ρ, security in this case is actually equivalent to security in the case where a polynomial number of protocols $\rho_1, \rho_2, \ldots$ are concurrently with each other and with π.)

Concurrent general composition with fixed inputs. The only difference between this case and the above definition is that the input to ρ is fixed before the execution of π begins. Formally, this is modeled by handing each party a pair of inputs (x_i, y_i), where x_i is party P_i's input to π and y_i is its input to ρ. (Likewise, when considering the more general setting that π is run concurrently with protocols $\rho_1, \rho_2, \ldots,$ the inputs to all executions of $\rho_1, \ldots$ are fixed before π begins. In this case, each party receives a vector of inputs $(x_i, y_i^1, y_i^2, \ldots)$ such that x_i is party P_i's input to π and y_i^j is its input to ρ_j.) Security is required to hold for all possible vectors of inputs of all possible lengths (that is, we quantify over all adversaries, all polynomials $p(\cdot)$, and all possible vectors of inputs of length $p(n)$). As above, we will use the simpler formulation where a single execution of a protocol ρ is run together with an arbitrary protocol π.

Universal composability [8]. Universal composability is a definition of security that considers a stand-alone execution of a protocol in a special setting involving an environment machine $\mathcal{Z}$, in addition to the honest parties and adversary. As with the standard definition, ideal and real models are considered where a trusted party carries out the computation in the ideal model and the real protocol is run in the real model. The environment adaptively chooses the inputs for the honest parties, interacts with the adversary throughout the computation, and receives the honest parties' outputs. Security is formulated by requiring the existence of an ideal-model simulator $\mathcal{S}$ so that no environment $\mathcal{Z}$ can distinguish between the case that it runs with the real adversary $\mathcal{A}$ in the real model and the case that it runs with the ideal-model simulator $\mathcal{S}$ in the ideal model. The importance of this definition is a composition theorem that states that any protocol that is universally composable is secure under concurrent general composition [8]. It has also been shown that full equivalence holds; that is, any protocol that is secure under concurrent general composition (as formulated above) is also universally composable [25]. Thus, all of our results here for concurrent general composition hold equivalently for universal composability.

4. A counterexample to the folklore.

It seems to be well-accepted folklore that any protocol that is secure with a *straight-line black-box simulator* is secure under concurrent composition. This folklore probably stems from the fact that all known protocols for the information-theoretic setting have black-box straight-line simulators and are assumed (though most of them do not have proofs) to be secure under concurrent composition. In this section we show that this folklore is false. Specifically, we present a protocol that is statistically secure *and* has a straight-line black-box simulator, yet is not secure under concurrent general composition. (Recall that in the setting of concurrent general composition, a secure protocol runs concurrently to arbitrary other protocols.) We note that the simulator that we present for this protocol also runs in polynomial-time. Thus, the counterexample holds also for the computational setting.

The main idea behind our counterexample is as follows. In the stand-alone model, the parties' inputs are all fixed before the execution begins. In contrast, in the setting of concurrent general composition, a party's input to the secure protocol may depend on messages that it receives in another protocol. Furthermore, since the scheduling is concurrent (and controlled by the adversary), it is possible that a party's input to a secure protocol ρ is defined *after* other parties have already begun running ρ. Such an event cannot happen in the stand-alone model, where all the parties' inputs are fixed before the execution begins.

Universal composability—discussion. We note that our example highlights an important issue regarding the definition of universal composability [8]. Namely, this definition has two stringencies over other definitions. First, it requires that the simulator be straight-line and black-box. Second, it requires that the inputs may be chosen adaptively (by an external environment). It seems to have been generally believed (although not stated in [8]) that the first stringency is the main one needed for achieving composition. Our example shows that both are actually necessary.

The counterexample. Our counterexample consists of a certain protocol that computes the three-party constant function f defined by $f(x_1, x_2, x_3) = (0, 0, 0)$, where $|x_1| = |x_2| = |x_3| = k$. (That is, all parties receive 0, irrespective of the values of their inputs; obviously, this function can also be computed by other protocols that do guarantee any type of security.) Specifically, we prove the following proposition.

THEOREM 4.1. *Assuming an honest majority of participants, there exist a function f and a protocol that computes f with statistical/computational security in the stand-alone model and with black-box straight-line simulation, that is not secure under concurrent general composition. This holds for both static and adaptive corruptions.*

Proof. We begin by presenting a three-party protocol that computes the function $f(x_1, x_2, x_3) = (0, 0, 0)$ with statistical security in the stand-alone model and with black-box straight-line simulation, in the case of an honest majority.

PROTOCOL 4.2.

1. *Party P_2 chooses a random string r_2 of length $k/2$ and sends it to P_1.*
2. *Party P_1 chooses a random string r_1 of length $k/2$ and sends it to P_2.*
3. *Parties P_1 and P_2 define $r = (r_1, r_2)$, where (a, b) denotes the concatenation of a and b. If one of the parties does not receive the r_i value from the other, then it just waits. If one of the parties receives an invalid value (i.e., one not of length $k/2$), then it sets r to a uniformly distributed string of length k.*
4. *Parties P_1 and P_2 both send r to P_3.*
5. *If P_3 received the same string from both P_1 and P_2, and the string equals its input x_3, then it outputs* 1. *Otherwise, it outputs* 0.
6. *P_1 and P_2 always output* 0.

We begin by showing that Protocol 4.2 is secure in the stand-alone model. Intuitively, the protocol is secure because P_3 outputs 1 only in the case that r equals its input. However, since at least one of P_1 or P_2 is honest, and thus chooses r_i at random ($i \in \{1, 2\}$), the probability that r equals P_3's input is at most $2^{-k/2}$.

CLAIM 4.3. *Protocol* 4.2 *computes f with statistical/computational security in the stand-alone model with black-box straight-line simulation and an honest majority.*

Proof. In order to prove Claim 4.3, we construct a simulator $\mathcal{S}$ as follows.

1. If no parties are corrupted, then $\mathcal{S}$ internally runs the code of all parties for the adversary $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.
2. If P_3 is corrupted, then $\mathcal{S}$ internally runs the code of the honest parties and hands the adversary $\mathcal{A}$ controlling P_3 the messages that it expects to receive from the honest parties. At the end, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.
3. If P_1 is corrupted, then $\mathcal{S}$ interacts with the adversary $\mathcal{A}$ controlling P_1 just as the honest P_2 would (sending it a random r_2 and receiving back r_1). Upon receiving r_1 from $\mathcal{A}$, simulator $\mathcal{S}$ simulates sending $r = (r_1, r_2)$ to P_3. At the

end, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

4. If P_2 is corrupted, then $\mathcal{S}$ behaves in a symmetrical way to the case that P_1 is corrupted.

Since we assume an honest majority, these are all the possible corruption scenarios. We claim that the above simulation results in outputs that are statistically close to a real execution. Notice that the only possible difference between a real execution with an adversary $\mathcal{A}$ and an ideal execution with $\mathcal{S}$ is in the case that the messages that P_3 receives from P_1 and P_2 equal its input x_3 (because in this case P_3 outputs 1, whereas in an ideal execution it always outputs 0). However, we claim that this can happen only with negligible probability. In the case that one of P_1 or P_2 receives an invalid string r_i, this is clear (because the aborted party sends a truly random string of length k to P_3). In all other cases, this follows from the fact that at least one of P_1 or P_2 are honest (recall that we assume an honest majority here). Therefore, the entropy of the string received by P_3 is at least $k/2$. Since x_3 is fixed before the execution began, the probability that an honest P_3 outputs 1 is at most $2^{-k/2}$. We conclude that Protocol 4.2 computes f with statistical (and hence computational) security,[7] completing the proof of Claim 4.3. ☐

We now prove that Protocol 4.2 is *not* secure under concurrent general composition.

CLAIM 4.4. *Protocol* 4.2 *does not compute f with statistical or computational security in the setting of concurrent general composition.*

Proof. The idea behind the proof of this claim is that the input of P_3 is not necessarily fixed before P_1 and P_2 start running their protocol. Since we are working in a concurrent setting, this is inevitable. We begin by describing a protocol π that contains an ideal call to f. (The protocol π is the "arbitrary protocol" that runs concurrent to the secure Protocol 4.2.)

Arbitrary protocol π.

1. Party P_1 sends a random string $s \in_R \{0,1\}^n$ to P_3.
2. Parties P_1 and P_2 send the trusted party computing f the input 0^n.
3. Party P_3 sends the trusted party computing f the input s (as received from P_1) and outputs whatever output it receives back from the trusted party.
4. Parties P_1 and P_2 record their outputs whenever they receive them from the trusted party and they output these recorded values.

Consider now a real execution of π with Protocol 4.2 replacing the trusted computation of f (Steps 2–4), and an adversary $\mathcal{A}$ who controls a corrupted P_1. In such an execution, parties P_1 and P_2 begin running Protocol 4.2. (Note that P_3 does not start running Protocol 4.2 until it receives its input s from P_1 in π.) P_1 receives a string r_2 from the honest P_2 and chooses its own random r_1. The adversarial strategy of $\mathcal{A}$ controlling P_1 here is to send $s = r = (r_1, r_2)$ to P_3 *as part of the protocol* π (i.e., Step 1 in protocol π), and to send r_1 to P_2 *as part of Protocol* 4.2 (i.e., Step 2 in Protocol 4.2). $\mathcal{A}$ then proceeds with Protocol 4.2 following the instructions of P_1 honestly. (In particular, it sends $r = (r_1, r_2)$ to P_3 and outputs 0.)

Notice that in a real execution of π with Protocol 4.2 and adversary $\mathcal{A}$, in Protocol 4.2 party P_3 always receives messages from P_1 and P_2 that equal its input x_3. This

[7]One minor issue here relates to the fact that by our description, it is possible that P_3 does not receive output. This occurs if it waits forever for P_1 and P_2 to send their messages. The simplest way of dealing with this is just to state that our definition of security does not require the parties to generate output. This makes sense when we move to the full concurrent setting, where the adversary controls all scheduling. If we want to assume some synchrony or timing, this is also fine. Then, P_3 will just wait the appropriate amount of time and will output 0 if it didn't receive both messages.

is because P_3 sets its input x_3 to equal s, and P_1 is able to make $s = r$. Therefore, P_3 always outputs 1 from such an execution of Protocol 4.2 with π and $\mathcal{A}$. By the definition of π, this means that P_3 also always outputs 1 from π in such an execution with $\mathcal{A}$. (Recall that in a real execution of π with Protocol 4.2, the output of f is replaced with the output of Protocol 4.2.)

It remains to note that for all simulators working in a hybrid/ideal model where π is run together with an ideal call to a trusted party computing f, the output of the honest parties P_2 and P_3 from π is always 0, because this is the output *always* sent by the trusted party. Therefore, the outputs of the real and ideal (hybrid) executions are easily distinguishable. This completes the proof of Claim 4.4. $\square$

Theorem 4.1 follows by combining Claims 4.3 and 4.4. $\square$

5. Perfect security and concurrent general composition. Theorem 4.1 states that black-box straight-line simulation and *statistical security* do not suffice for achieving security under concurrent general composition. In this section, we show that if the protocol has a black-box straight-line simulator and is *perfectly secure*, then it *is* secure under concurrent general composition.

THEOREM 5.1. *Let f be a function, and let ρ be a protocol that computes f with perfect security and abort in the stand-alone model, with a black-box straight-line simulator. Then, ρ computes f with perfect security under concurrent general composition. Furthermore, if the black-box simulator for ρ in the stand-alone model runs in polynomial-time, then so does the simulator for ρ in the setting of concurrent general composition. This holds for both static and adaptive adversaries.*

Proof. The idea behind the proof of this theorem is as follows. Assume, by contradiction, that ρ is not secure under concurrent general composition. Loosely speaking, this means that there exists a protocol π and an adversary $\mathcal{A}$ such that a real execution of π and ρ with $\mathcal{A}$ cannot be simulated in the hybrid world where π is run together with a trusted party who computes f.[8] In particular, it must be that the output distribution of the adversary and honest parties in ρ, when running it together with π, is not the same as their output distribution when they just send their inputs to a trusted party computing f. The main idea here is to use this fact to attack a stand-alone execution of ρ. Specifically, we construct a stand-alone adversary $\mathcal{A}_\rho$ who attacks ρ by internally simulating the entire π execution for $\mathcal{A}$, while running the protocol ρ externally with the honest parties. If $\mathcal{A}_\rho$'s simulation of π is "good," then it follows that the *stand-alone* execution of ρ will be the same as when it is run together with π. Thus, the output of ρ in this stand-alone execution will not be the same as the output of an ideal execution with a trusted party computing f. The problem with this approach is that in order for $\mathcal{A}_\rho$'s simulation of π to be "good," it must somehow guess the honest party's inputs and random coins in a way that will make the combination of the internal π simulation and the external ρ execution look the same as a full external execution of π with ρ. This is especially problematic because in the setting of concurrent composition, the honest parties' inputs to ρ may be *determined* by π (whereas in the stand-alone model, the inputs of the honest parties are fixed ahead of time). Nevertheless, in the case of *perfect security,* this is not a problem because we only need $\mathcal{A}_\rho$'s attack on ρ to "succeed" with nonzero probability. In particular, even a statistical distance of $2^{-n^{O(1)}}$ between the real and

[8] We remark that we prove the theorem directly for security under concurrent general composition, without going through the definition of universal composability [8]. Those who feel more comfortable using universal composability can just think of the protocol π as the environment $\mathcal{Z}$ and everything remains the same.

ideal executions is not allowed. Now, with very small probability, $\mathcal{A}_\rho$'s guesses are "correct," and in this case, the output distribution and execution of ρ with $\mathcal{A}_\rho$ is the same as after an execution of π and ρ with $\mathcal{A}$. By the contradicting assumption, this output distribution is not simulatable in the ideal model, and so we have that the real and ideal distributions are not identical, in contradiction to the perfect security of ρ in the stand-alone model. We note that the above contains the main idea behind the proof. However, the structure of the actual proof is a little different. We now proceed to the formal proof. For the sake of simplicity, we provide the proof for the case of static adversaries. In the case of adaptive corruptions, the only difference is that the $\mathcal{A}_\rho$ needs to corrupt a party whenever $\mathcal{A}$ does and then provide $\mathcal{A}$ with the entire state of that party. This is not a problem because the state of all parties in π is known completely by $\mathcal{A}$ (who runs the entire π execution internally) and the state of a newly corrupted party in ρ is learned by $\mathcal{A}_\rho$ when it carries out the corruption. (The simulation strategy in the case of adaptive corruptions is derived in the same way.)

In this proof, without loss of generality, we consider only the case that the arbitrary protocol π has a *single* ideal call to f. We note that in the case that a single universal simulator is provided for all protocols π (as will be the case here), this is equivalent to the case that π has any polynomial number of ideal calls to f (see the full version of [25] for a proof). Recall that we denote by $\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(\mathbf{x})$ the distribution of the outputs of the adversary $\mathcal{A}$ with auxiliary input z and the honest parties in a real execution of π with ρ when given input vector $\mathbf{x}$. Likewise, we denote by $\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(\mathbf{x})$ the distribution of the outputs of the adversary $\mathcal{S}$ and the honest parties in a hybrid execution of π with a trusted party computing f when given input vector $\mathbf{x}$. (We don't use a security parameter k here in the notation because it is not needed when considering perfect security.)

Let f be a function, let ρ be a protocol that computes f with perfect security, and let $\mathcal{S}_\rho$ be the black-box straight-line ideal-model simulator for ρ that is assumed to exist. Then, we construct a simulator $\mathcal{S}$ for the setting of concurrent general composition as follows. Let $\mathcal{A}$ be a real-model adversary, and let π be an arbitrary protocol that contains a single ideal call to f. Then, $\mathcal{S}$ invokes the real adversary $\mathcal{A}$ and forwards all π-messages untouched between $\mathcal{A}$ and the honest parties (recall that $\mathcal{S}$ runs a real execution of π with external parties). In contrast, when $\mathcal{A}$ begins the ρ execution, $\mathcal{S}$ invokes a copy of $\mathcal{S}_\rho$ and forwards all ρ-messages between $\mathcal{A}$ and $\mathcal{S}_\rho$ (in contrast to the π-messages that are sent externally, ρ is internally simulated by $\mathcal{S}$ using $\mathcal{S}_\rho$). More specifically, when $\mathcal{A}$ outputs a ρ-message to be sent to an honest party, $\mathcal{S}$ hands this message to $\mathcal{S}_\rho$. Likewise, $\mathcal{S}_\rho$'s reply is handed back to $\mathcal{A}$ as if coming from an honest party. In addition, whenever $\mathcal{S}_\rho$ sends an input to the trusted party computing f, simulator $\mathcal{S}$ sends the same input. Likewise, outputs sent to $\mathcal{S}$ from the trusted party are forwarded to $\mathcal{S}_\rho$. (We note that the π execution continues by just forwarding π-messages between $\mathcal{A}$ and the honest parties, even while the simulation of ρ takes place. Notice that the real execution of π can continue concurrently with the simulation of ρ because $\mathcal{S}_\rho$ is *straight-line,* and so it never needs to rewind.[9]) Whenever $\mathcal{A}$ halts, simulator $\mathcal{S}$ copies $\mathcal{A}$'s output to its own output tape and halts.

We now prove that for every arbitrary protocol π (with a single ideal call to f)

[9]If $\mathcal{S}_\rho$ were to rewind $\mathcal{A}$, then a problem would arise in the case that a π-message that $\mathcal{A}$ sent before rewinding is changed by $\mathcal{A}$ after rewinding. In particular, since these messages are sent *externally* to honest parties, they cannot be changed at a later stage.

and every real adversary $\mathcal{A}$,

$$\left\{\textsc{hybrid}^{f}_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(\mathbf{x})\right\}_{z,\mathbf{x}\in\{0,1\}^*} \equiv \left\{\textsc{real}_{\pi^\rho,\mathcal{A}(z)}(\mathbf{x})\right\}_{z,\mathbf{x}\in\{0,1\}^*}. \tag{5.1}$$

Assume, by contradiction, that this is not the case. This implies that there exists a protocol π, an adversary $\mathcal{A}$, and a vector of inputs $\mathbf{x}$ such that (5.1) does not hold.

First, assume that the output of each honest party in π includes its local input to and output from the ideal call to f (or likewise, its local input to and output from the execution of ρ). Clearly (5.1) does not hold here as well, because we only provided more output. Next, modify the protocol π to π' so that the honest parties output their local input to and output from f (or ρ) only, and not their original π-output. This is the only difference between π and π'. In particular, the honest parties run π to the end as before, following the same instructions. We now claim that (5.1) also does not hold with respect to π'. That is,

$$\left\{\textsc{hybrid}^{f}_{\pi',\mathcal{S}^{\mathcal{A}(z)}}(\mathbf{x})\right\} \not\equiv \left\{\textsc{real}_{\pi'^\rho,\mathcal{A}(z)}(\mathbf{x})\right\}. \tag{5.2}$$

This requires justification because in π' the output distribution is different (and contains less "information"). In order to see that (5.2) holds, we show that if this were not the case, then (5.1) would hold. This follows from the fact that the only difference between $\textsc{hybrid}^{f}_{\pi,\mathcal{S}}(\mathbf{x})$ and $\textsc{real}_{\pi^\rho,\mathcal{A}}(\mathbf{x})$ for the honest parties is that the output from f is obtained from the trusted party in HYBRID and from the execution of ρ in REAL. Furthermore, protocol π refers only to the input to ρ and output from ρ and thus only the input to and output from ρ influence the honest parties' output in π. This implies that if the *joint input/output distribution* to $\boldsymbol{\rho}$ are identically distributed in HYBRID and REAL, then the honest parties' outputs from $\boldsymbol{\pi}$ are identically distributed in HYBRID and REAL.[10] One subtle point that needs to be clarified is that the output of the honest parties in π is also influenced by the messages that they receive from the adversary. However, by the definition of $\mathcal{S}$, the messages received by the honest parties *within the execution of* π are identical in the HYBRID and REAL executions; indeed, $\mathcal{S}$ merely forwards these messages unmodified between $\mathcal{A}$ and the honest parties. This contradicts the assumption that (5.1) does not hold, and so we conclude that (5.2) holds.

Now, let $\mathbf{x}_\rho$ be a vector of inputs to ρ such that in the event that the inputs determined by π' to f (or equivalently to ρ) equal $\mathbf{x}_\rho$, the $\textsc{hybrid}^{f}_{\pi',\mathcal{S}}(\mathbf{x})$ and $\textsc{real}_{\pi'^\rho,\mathcal{A}}(\mathbf{x})$ distributions are not identical. Such a vector of inputs $\mathbf{x}_\rho$ to ρ must exist, because otherwise the HYBRID and REAL distributions would be identical. (More formally, we can break up the distributions of $\textsc{hybrid}^{f}_{\pi',\mathcal{S}}(\mathbf{x})$ and $\textsc{real}_{\pi'^\rho,\mathcal{A}}(\mathbf{x})$ according to all possible sets of inputs to f and ρ in an execution of π' with $\mathbf{x}$. Then, it must be that for at least one of these "subdistributions," the HYBRID and REAL distributions are not identical.)

We are now ready to construct an adversary $\mathcal{A}_\rho$ that attacks a *stand-alone execution of* ρ, and succeeds when the input vector to ρ equals $\mathbf{x}_\rho$. Adversary $\mathcal{A}_\rho$ works by internally emulating all the honest parties in the execution of π', and externally running the ρ execution. Specifically, $\mathcal{A}_\rho$ sets the inputs of the (internally-emulated) parties running π' to $\mathbf{x}$, and also chooses uniformly distributed random tapes for these parties. Then, $\mathcal{A}_\rho$ invokes $\mathcal{A}$ and perfectly emulates an execution of $\textsc{real}_{\pi'^\rho,\mathcal{A}}(\mathbf{x})$,

[10] This is not necessarily true for the adversary's output. However, the adversary's output is unchanged in π and π'.

by passing all π'-messages between $\mathcal{A}$ and the internally-emulated honest parties. In contrast, when the execution of ρ is reached, $\mathcal{A}_\rho$ sends $\mathcal{A}$'s ρ-messages externally to the real honest parties running ρ. Likewise, ρ-messages received externally by $\mathcal{A}_\rho$ are internally handed back to $\mathcal{A}$. We note that the internal emulation of π continues concurrently to the external execution of ρ (according to the scheduling determined by $\mathcal{A}$). At the end of the external execution of ρ, adversary $\mathcal{A}_\rho$ guesses the outputs of the honest parties (from the set of all possible outputs) and uses them in the continuation of the internal emulation of π.[11] $\mathcal{A}_\rho$ continues this internal emulation of π' until $\mathcal{A}$ halts. At this point $\mathcal{A}_\rho$ outputs $\mathcal{A}$'s output, as well as its guesses for the honest parties' inputs to ρ and their outputs. $\mathcal{A}_\rho$ then halts.

First, observe that with nonzero probability, the inputs of the internally-emulated honest parties to ρ equal $\mathbf{x}_\rho$ (and therefore match the real inputs of the external parties). This must hold because otherwise $\mathbf{x}_\rho$ would not have been chosen above (an $\mathbf{x}_\rho$ which appears with zero probability cannot contribute to a statistical difference between the REAL and HYBRID distributions). Furthermore, the guess of $\mathcal{A}_\rho$ with respect to the honest parties' outputs from ρ is also correct with nonzero probability. Now, conditioned on these guesses being correct, the output of the honest parties from ρ along with the portion of $\mathcal{A}_\rho$'s output that is copied from $\mathcal{A}$'s output is distributed exactly according to $\text{REAL}_{\pi'^\rho,\mathcal{A}(z)}(\mathbf{x})$. Furthermore, in an ideal-model simulation by $\mathcal{S}_\rho$, once again conditioned on the guesses being correct, the output of the (ideal) honest parties along with the appropriate portion of $\mathcal{S}_\rho$'s output is distributed exactly according to $\text{HYBRID}^f_{\pi',\mathcal{S}^{\mathcal{A}(z)}}(\mathbf{x})$.

Notice finally that because $\mathcal{A}_\rho$ outputs its guesses for the inputs and outputs of the honest parties (and whether or not they are correct can be derived from comparing $\mathcal{A}_\rho$'s output to the real inputs and outputs of the honest parties), the "subdistribution" of the guesses of $\mathcal{A}_\rho$ being correct is disjoint from the case that the guesses of $\mathcal{A}_\rho$ are incorrect. Since it follows from the contradicting assumption that this subdistribution is not identically distributed in the real and ideal executions, we conclude that

$$\left\{\text{IDEAL}^f_{\mathcal{S}_\rho^{\mathcal{A}_\rho(z)}}(\mathbf{x}_\rho)\right\} \not\equiv \left\{\text{REAL}_{\rho,\mathcal{A}_\rho(z)}(\mathbf{x}_\rho)\right\},$$

in contradiction to the perfect security of ρ with the black-box simulator $\mathcal{S}_\rho$.

The "furthermore" in the theorem statement follows from the fact that the simulator $\mathcal{S}$ that we constructed merely invokes $\mathcal{A}$ and the simulator $\mathcal{S}_\rho$ for ρ. Thus, if $\mathcal{S}_\rho$ runs in time that is polynomial in the running time of $\mathcal{A}$, so does $\mathcal{S}$. This completes the proof. ☐

The computational setting. In the information-theoretic setting, ideal secure channels are assumed to exist. However, in the computational model only authenticated channels are used. Thus, information-theoretic protocols cannot be used directly in the computational setting. Fortunately, this is easy to fix; it suffices to use computationally secure channels as formulated in [9] (these can be constructed from any CCA2-secure public-key encryption scheme, or any public-key encryption scheme

[11]This guess can be made efficient by guessing the honest parties' inputs and random tapes. In the case that the messages sent by the real honest parties are consistent with the guessed inputs and random tapes, $\mathcal{A}_\rho$ sets their outputs according to these values and the messages sent by $\mathcal{A}$ to the honest parties (if the guess of $\mathcal{A}_\rho$ is correct here, it follows that $\mathcal{A}_\rho$ holds the view of the honest parties and so can compute their exact outputs). Otherwise, $\mathcal{A}_\rho$ sets their outputs to some default values. Notice that if $\mathcal{A}_\rho$'s input and random-tape guesses are correct, then so are the output guesses. Furthermore, the input and random-tape guesses are correct with nonzero probability.

when assuming authenticated channels. We denote by ρ' the protocol which is identical to ρ except that all messages are sent encrypted, according to the scheme shown in [9]. This yields the following corollary.

COROLLARY 5.2. *Assume the existence of public-key encryption schemes. Let f be a function, and let ρ be a protocol that computes f with perfect security and abort in the stand-alone model and has a black-box straight-line simulator. Then, protocol ρ' defined above securely computes f under concurrent general composition in the computational setting.*

6. Straight-line black-box simulation and start synchronization.

Theorem 4.1 states that statistical security and black-box straight-line simulation do not suffice for achieving security under concurrent general composition. In this section, we show that a mild additional requirement can be made on the protocol so that it does suffice. Namely, we prove that if a secure protocol has a black-box straight-line simulator, and no party sends any message that depends on its input or random tape until all parties have announced that they have started, then the protocol *is* secure under concurrent general composition. This result holds for all levels of security, even computational security.

Start synchronization. We say that a protocol ρ has start synchronization if it begins with the following steps for *all* parties:

1. Send "begin ρ" to all parties.
2. Wait until "begin ρ" messages are received from *all* parties. Then, continue the execution of ρ.

We stress that the inputs to the protocol are read by each party before it sends its begin ρ message, and are therefore fixed before the actual execution of ρ begins (or, more exactly, before any honest party sends a message that depends on its input or random tape). We prove the following theorem.

THEOREM 6.1. *Let f be a function, and let ρ be a protocol that computes f with computational (resp., statistical or perfect) security and abort in the stand-alone model, and with a black-box straight-line simulator. Furthermore, assume that ρ has start synchronization. Then, ρ computes f with computational (resp., statistical or perfect) security under concurrent general composition. This holds for both static and adaptive adversaries.*

Proof. The case of perfect security follows from Theorem 5.1, even without start synchronization. It therefore remains to prove the theorem for the computational and statistical cases. We present the proof for the computational setting; only minor modifications are needed for the statistical case. Also, as in the proof of Theorem 5.1, we provide the proof only for the case of static adversaries; dealing with adaptive corruptions here is straightforward. The intuition behind this theorem comes from the counterexample of section 4. Specifically, notice that the security of the protocol there is compromised by having one party's input depend on messages sent by the other parties in ρ itself. When there is start synchronization, this cannot happen. Technically, the proof utilizes start synchronization by the following observation. Assume that all of the parties' inputs to ρ in a concurrent execution with some protocol π, depend only on π (as is indeed the case when there is start synchronization). Then, all of the computation of π can be thrown into an auxiliary input for an adversary attacking a stand-alone execution of ρ. Thus, we can use the stand-alone security of ρ to derive security under concurrent general composition. (Note that an auxiliary input can be correlated to the parties' inputs to ρ but not to their random tapes in ρ. Thus, this strategy cannot work if there is no start synchronization, as is demonstrated in the counterexample of section 4.)

We begin the proof in a very similar fashion to the proof of Theorem 5.1. That is, we use the simulator $\mathcal{S}_\rho$ that exists for ρ (by the assumption that it is secure in the stand-alone model) in order to construct a simulator $\mathcal{S}$ for the setting of concurrent general composition. As in the proof of Theorem 5.1, the simulator $\mathcal{S}$ works by forwarding all of the π-messages untouched (where π is an arbitrary protocol that contains an ideal call to f), and using $\mathcal{S}_\rho$ to deal with all of the ρ-messages. More specifically, let f be a function, let ρ be a protocol that computes f with computational security, and let $\mathcal{S}_\rho$ be the black-box straight-line ideal-model simulator for ρ that is assumed to exist. Then, we construct a simulator $\mathcal{S}$ for the setting of concurrent general composition as follows. Let $\mathcal{A}$ be a real-model adversary, and let π be an arbitrary protocol that contains a single ideal call to f (as discussed in the proof of Theorem 5.1, considering a single call to f suffices). Then, $\mathcal{S}$ invokes the real adversary $\mathcal{A}$ and forwards all π-messages untouched between $\mathcal{A}$ and the honest parties (recall that $\mathcal{S}$ runs a real execution of π with external parties). In contrast, when $\mathcal{A}$ begins the ρ execution, $\mathcal{S}$ invokes a copy of $\mathcal{S}_\rho$ and forwards all ρ-messages between $\mathcal{A}$ and $\mathcal{S}_\rho$ (in contrast to the π-messages that are sent externally, ρ is internally simulated by $\mathcal{S}$ using $\mathcal{S}_\rho$). More specifically, when $\mathcal{A}$ outputs a ρ-message to be sent to an honest party, $\mathcal{S}$ hands this message to $\mathcal{S}_\rho$. Likewise, $\mathcal{S}_\rho$'s reply is handed back to $\mathcal{A}$ as if coming from an honest party. In addition, whenever $\mathcal{S}_\rho$ sends an input to the trusted party computing f, simulator $\mathcal{S}$ sends the same input. Likewise, outputs sent to $\mathcal{S}$ from the trusted party are forwarded to $\mathcal{S}_\rho$. We note that the π execution continues by just forwarding π-messages between $\mathcal{A}$ and the honest parties, even while the simulation of ρ takes place. Notice that the real execution of π can continue concurrently with the simulation of ρ because $\mathcal{S}_\rho$ is *straight-line,* and so never needs to rewind (this is where the straight-line property of $\mathcal{S}_\rho$ is used). Whenever $\mathcal{A}$ halts, simulator $\mathcal{S}$ copies $\mathcal{A}$'s output to its own output tape and halts.

Let k be the security parameter (i.e., all parties are assumed to run in time that is polynomial in k). Recall that we denote by $\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(k,\mathbf{x})$ a real execution of the protocol π with real protocol ρ, where $\mathcal{A}$ has auxiliary input z, and the vector of the parties' inputs equals $\mathbf{x}$. Likewise, we denote by $\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(k,\mathbf{x})$ the hybrid execution of π with ideal calls to f, where the parties' inputs are $\mathbf{x}$, and the ideal-world adversary/simulator is given oracle (i.e., black-box) access to $\mathcal{A}(z)$.

We now prove that for every arbitrary protocol π (with a single ideal call to f) and every real adversary $\mathcal{A}$,

$$\left\{\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(k,\mathbf{x})\right\}_{\mathbf{x},z\in\{0,1\}^*;k\in\mathsf{N}} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(k,\mathbf{x})\right\}_{\mathbf{x},z\in\{0,1\}^*;k\in\mathsf{N}}. \tag{6.1}$$

Assume, by contradiction, that this is not the case. This implies that there exists a protocol π, a distinguisher D, an adversary $\mathcal{A}$, a polynomial p, and a vector of inputs $\mathbf{x}$ and auxiliary input z such that for infinitely many k's,

$$\left|\Pr\left[D\left(\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(k,\mathbf{x})\right)=1\right]-\Pr\left[D(\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(k,\mathbf{x}))\right]\right|>\frac{1}{p(k)}.$$

We use this to contradict the security of ρ. That is, we show that there exists a distinguisher D_ρ, real-world adversary $\mathcal{A}_\rho$, polynomial q, and an auxiliary input z_ρ and vector of inputs $\mathbf{x}_\rho$ such that for infinitely many k's,

$$\left|\Pr\left[D_\rho\left(\text{IDEAL}^f_{\mathcal{S}_\rho^{\mathcal{A}_\rho(z_\rho)}}(k,\mathbf{x}_\rho)\right)=1\right]-\Pr\left[D(\text{REAL}_{\rho,\mathcal{A}_\rho(z_\rho)}(k,\mathbf{x}_\rho))\right]\right|>\frac{1}{q(k)}, \tag{6.2}$$

in contradiction to the assumption that $\mathcal{S}_\rho$ is a "good simulator."

As we have mentioned above, the key point in the proof is to use the auxiliary input z_ρ for $\mathcal{A}_\rho$ in a stand-alone execution of ρ in place of the execution of π together with ρ. Specifically, a secure protocol must remain secure when the adversary receives any auxiliary input, even one that is correlated with the honest parties' inputs. In particular, for every vector of inputs $\mathbf{x}_\rho$ to ρ, security must hold when the auxiliary input z_ρ consists of a vector of inputs $\mathbf{x}$ and random coins $\mathbf{r}$ for the honest parties (and the adversary) such that when π is run with the adversary $\mathcal{A}$, inputs $\mathbf{x}$, and random tapes $\mathbf{r}$, the honest parties' inputs to f (or equivalently to ρ) are those given in $\mathbf{x}_\rho$. (Note that the random tapes $\mathbf{r}$ in z_ρ are actually chosen uniformly from all those meeting the requirement.) Before proceeding, we stress that defining such an auxiliary input is possible only if the inputs of the parties in ρ are *independent* of the ρ-execution itself. This is ensured because ρ begins with start synchronization.

Next, we describe an adversary $\mathcal{A}_\rho$ attacking ρ. Adversary $\mathcal{A}_\rho$ internally invokes $\mathcal{A}$ and perfectly simulates the honest parties' actions in π by running the π-instructions of the honest parties on the inputs and random tapes specified in its auxiliary input z_ρ. In contrast to the π-messages that are internally emulated by $\mathcal{A}_\rho$ for $\mathcal{A}$, all of the messages belonging to ρ are forwarded by $\mathcal{A}_\rho$ from $\mathcal{A}$ to the external honest parties and back. At the end of the execution of ρ, adversary $\mathcal{A}_\rho$ halts, outputting $\mathcal{A}$'s current view.

Given the above, we show that by the contradicting assumption, there exists at least one vector of inputs $\mathbf{x}_\rho$ to ρ with associated auxiliary input z_ρ such that a real execution of ρ with adversary $\mathcal{A}_\rho$, auxiliary input z_ρ, and inputs $\mathbf{x}_\rho$, can be distinguished from an ideal execution of f with adversary/simulator $\mathcal{S}_\rho^{\mathcal{A}_\rho(z_\rho)}$ and inputs $\mathbf{x}_\rho$. We prove this by showing that if this is not the case (i.e., if (6.2) does *not* hold), then this would imply that (6.1) holds (in contradiction to the assumption). To see this, notice that an efficient distinguisher D_ρ who receives the adversary's view after the execution of ρ is completed (including its outputs from ρ or f), the auxiliary input z_ρ of the adversary, and the honest parties' outputs from ρ or f, can complete the execution of π itself and obtain the outputs of $\mathcal{A}$ and all the honest parties. Now, if D_ρ receives outputs from a real execution of ρ, the output distribution that it will internally compute will be exactly $\text{REAL}_{\pi^\rho,\mathcal{A}(z)}(k,\mathbf{x})$. In contrast, if D_ρ receives outputs from an ideal execution of f, the output distribution that it will internally compute will be exactly $\text{HYBRID}^f_{\pi,\mathcal{S}^{\mathcal{A}(z)}}(k,\mathbf{x})$. (This holds due to the construction of $\mathcal{S}$ that runs $\mathcal{S}_\rho$ for the ρ part of the execution, and everything else is run according to $\mathcal{A}$.) We conclude that if (6.1) does not hold, then there exist $\mathbf{x}_\rho$ and z_ρ such that D_ρ can distinguish between IDEAL and REAL as in (6.2). This contradicts the assumed security of ρ. □

7. Black-box straight-line simulation and fixed inputs.

Theorem 4.1 states that black-box straight-line simulation does not suffice for achieving security under concurrent general composition. In this section, we show that it does however suffice for achieving security under concurrent general composition with *fixed inputs*. Recall that in the setting of *fixed inputs*, each party P_i receives a pair of inputs (x_i, y_i) before the execution begins. Then, party P_i uses x_i as its input to π and it uses y_i as its input to ρ.

THEOREM 7.1. *Let f be a probabilistic polynomial-time function, and let ρ be a protocol that computes f with computational (resp., statistical/perfect) security and abort in the stand-alone model under black-box straight-line simulation. Then, ρ computes f with computational (resp., statistical/perfect) security under concurrent gen-*

eral composition with fixed inputs.

The proof of this theorem is a simple, special case of the proof of Theorem 6.1, where instead of using the auxiliary input to fix the inputs to ρ, they are already fixed ahead of time.

Acknowledgment. We would like to thank the anonymous referees for their very helpful comments.

REFERENCES

[1] M. Backes, J. Müller-Quade, and D. Unruh, *On the necessity of rewinding in secure multiparty computation*, in Proceedings of the 4th IACR TCC, Lecture Notes in Comput. Sci. 4392, Springer-Verlag, Berlin, 2007, pp. 157–173.

[2] D. Beaver, *Multiparty protocols tolerating half faulty processors*, in Proceedings of the CRYPTO'89, Lecture Notes in Comput. Sci. 435, Springer-Verlag, New York, 1990, pp. 560–572.

[3] D. Beaver, *Foundations of secure interactive computing*, in Proceedings of the CRYPTO'91, Lecture Notes in Comput. Sci. 576, Springer-Verlag, New York, 1991, pp. 377–391.

[4] M. Ben-Or, R. Canetti, and O. Goldreich, *Asynchronous secure computation*, in Proceedings of the 25th Annual ACM STOC, San Diego, 1993, pp. 52–61.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson, *Completeness theorems for noncryptographic fault-tolerant distributed computations*, in Proceedings of the 20th Annual ACM STOC, Chicago, 1988, pp. 1–10.

[6] G. Bracha, *An asynchronous $(n-1)/3$-resilient consensus protocol*, in Proceedings of the 3rd ACM PODC, Vancouver, 1984, pp. 154–162.

[7] R. Canetti, *Security and composition of multiparty cryptographic protocols*, J. Cryptology, 13 (2000), pp. 143–202.

[8] R. Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, in Proceedings of the 42nd IEEE FOCS, Las Vegas, 2001, pp. 136–145; also available online from http://eprint.iacr.org/2000/067.

[9] R. Canetti and H. Krawczyk, *Universally composable notions of key exchange and secure channels*, in Proceedings of the EUROCRYPT 2002, Lecture Notes in Comput. Sci. 2332, Springer-Verlag, Berlin, 2002, pp. 337–351.

[10] D. Chaum, C. Crépeau, and I. Damgard, *Multiparty unconditionally secure protocols*, in Proceedings of the 20th Annual ACM STOC, Chicago, 1988, pp. 11–19.

[11] R. Cramer, I. Damgard, S. Dziembowski, M. Hirt, and T. Rabin, *Efficient multiparty computations with dishonest minority*, in Proceedings of the EUROCRYPT'99, Lecture Notes in Comput. Sci. 1592, Springer-Verlag, Berlin, 1999, pp. 311–326.

[12] Y. Dodis and S. Micali, *Parallel reducibility for information-theoretically secure computation*, in Proceedings of the CRYPTO 2000, Lecture Notes in Comput. Sci. 1880, Springer-Verlag, Berlin, 2000, pp. 74–92.

[13] D. Dolev, C. Dwork, and M. Naor, *Nonmalleable cryptography*, SIAM J. Comput., 30 (2000), pp. 391–437.

[14] C. Dwork, M. Naor, and A. Sahai, *Concurrent zero-knowledge*, J. ACM, 51 (2004), pp. 851–898.

[15] U. Feige, *Alternative Models for Zero Knowledge Interactive Proofs*, Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel, 1990; also available online from http://www.wisdom.weizmann.ac.il/∼feige.

[16] P. Feldman, *Asynchronous Byzantine Agreement in Constant Expected Time*, unpublished manuscript, 1989.

[17] O. Goldreich, *Foundations of Cryptography*, Vol. 1, Cambridge University Press, Cambridge, UK, 2001.

[18] O. Goldreich, *Foundations of Cryptography*, Vol. 2, Cambridge University Press, Cambridge, UK, 2004.

[19] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game, information-theoretically secure protocols and security under composition*, in Proceedings of the 38th Annual ACM STOC, Seattle, 2006, pp. 109–118.

[20] S. Goldwasser and L. Levin, *Fair computation of general functions in presence of immoral majority*, in Proceedings of the CRYPTO'90, Lecture Notes in Comput. Sci. 537, Springer-Verlag, New York, 1990, pp. 77–93.

[21] D. Hofheinz and D. Unruh, *Simulatable security and polynomially bounded concurrent composability*, in Proceedings of the 27th IEEE Symposium on Security and Privacy, Oakland, CA, 2006, pp. 169–183.

[22] E. Kushilevitz, Y. Lindell, and T. Rabin, *Information-theoretically secure protocols and security under composition*, in Proceedings of the 38th Annual ACM STOC, Seattle, 2006, pp. 109–118.

[23] Y. Lindell, *Bounded-concurrent secure two-party computation without setup assumptions*, in Proceedings of the 35th Annual ACM STOC, San Diego, 2003, pp. 683–692.

[24] Y. Lindell, *Lower bounds for concurrent self composition*, J. Cryptology, 21 (2008), pp. 200–249.

[25] Y. Lindell, *General composition and universal composability in secure multi-party computation*, J. Cryptology, 22 (2009), pp. 395–428.

[26] S. Micali and P. Rogaway, *Secure computation*, in Proceedings of the CRYPTO'91, Lecture Notes in Comput. Sci. 576, Springer-Verlag, New York, 1991, pp. 392–404.

[27] B. Pfitzmann and M. Waidner, *Composition and integrity preservation of secure reactive systems*, in Proceedings of the 7th ACM Conference on Computer and Communication Security, Athens, 2000, pp. 245–254.

[28] T. Rabin and M. Ben-Or, *Verifiable secret sharing and multiparty protocols with honest majority*, in Proceedings of the 21st Annual ACM STOC, Seattle, 1989, pp. 73–85.

[29] A. C.-C. Yao, *How to generate and exchange secrets*, in Proceedings of the 27th Annual IEEE FOCS, Toronto, 1986, pp. 162–167.