General Composition and Universal Composability in Secure Multiparty Computation[*]

Yehuda Lindell[†]

Department of Computer Science
Bar-Ilan University
Ramat Gan 52900, Israel
lindell@cs.biu.ac.il

November 28, 2007

Abstract

Concurrent general composition relates to a setting where a secure protocol is run in a network concurrently with other, arbitrary protocols. Clearly, security in such a setting is what is desired, or even needed, in modern computer networks where many different protocols are executed concurrently. Canetti (FOCS 2001) introduced the notion of *universal composability*, and showed that security under this definition is sufficient for achieving concurrent general composition. However, it is not known whether or not the opposite direction also holds.

Our main result is a proof that security under concurrent general composition, when interpreted in the natural way under the simulation paradigm, is equivalent to a variant of universal composability, where the only difference relates to the order of quantifiers in the definition. (In newer versions of universal composability, these variants are equivalent.) An important corollary of this theorem is that existing impossibility results for universal composability (for all its variants) are inherent for definitions that imply security under concurrent general composition, as formulated here. In particular, there are large classes of two-party functionalities for which *it is impossible* to obtain protocols (in the plain model) that remain secure under concurrent general composition. We stress that the impossibility results obtained are not "black-box", and apply even to non-black-box simulation.

Our main result also demonstrates that the definition of universal composability is somewhat "minimal", in that the composition guarantee provided by universal composability implies the definition itself. This indicates that the security definition of universal composability is not overly restrictive.

[*]An extended abstract of this work appeared in the *44th FOCS*, 2003.
[†]Most of this work was carried out while the author was at the IBM T.J.Watson Research Center.

1 Introduction

This paper considers the security of protocols for multiparty computation under composition.

1.1 Background – Secure Computation and Protocol Composition

Secure computation. In the setting of multiparty computation, a set of parties $P_1, \ldots, P_m$ with private inputs $\overline{x} = (x_1, \ldots, x_m)$, wish to jointly compute a functionality $f(\overline{x}) = (f_1(\overline{x}), \ldots, f_m(\overline{x}))$, such that each party P_i receives $f_i(\overline{x})$ for output. This functionality may be probabilistic, in which case $f(\overline{x})$ is a random variable, and it may be reactive, in which case the inputs and outputs are provided over a number of stages. Loosely speaking, the basic requirements on a secure multiparty protocol are that parties learn nothing from the protocol execution other than their output (privacy), and that the output is distributed according to the prescribed functionality (correctness). These security requirements must hold in the face of a malicious adversary who can corrupt a subset of the parties and have them arbitrarily deviate from the protocol specification. Powerful feasibility results have been shown for this problem, demonstrating that *any* multiparty probabilistic polynomial-time functionality can be securely computed [38, 20, 6, 15]. However, these feasibility results relate only to the stand-alone setting, where a *single* set of parties run a *single* execution.

Protocol composition. In general, the notion of "protocol composition" refers to a setting where the participating parties are involved in many protocol executions. Furthermore, the honest parties participate in each execution as if it is running in isolation (and therefore obliviously of the other executions taking place). In particular, this means that honest parties are not required to coordinate between different executions or keep track of the history of past executions. Requiring parties to coordinate between executions is highly undesirable and sometimes may even be impossible (e.g., it is hard to imagine successful coordination between protocols that are designed independently of each other). We note that in contrast to the honest parties, the adversary may coordinate its actions between the protocol executions. This asymmetry is due to the fact that some level of coordination is clearly possible. Thus, although it is undesirable to rely on it in the construction of protocols, it would be careless to assume that the adversary cannot utilize it to some extent. This is especially true since the adversary's strategy may be designed, given full knowledge of the protocols that will be run in the network.

Types of protocol composition. As we have described, the notion of protocol composition relates to settings in which many protocol executions take place. However, there are actually many ways to interpret this notion. We single out four different important parameters: the *context* in which the protocol runs, the *participating parties*, the *scheduling* and the *number of executions*.

1. **The context.** This refers to the question of *which protocols* are being run together in the network, or in other words, with which protocols should the protocol in question compose. There are two contexts that have been considered, defining two classes of composition:

   (a) Self composition: A protocol is said to be secure under *self composition* if it remains secure when it alone is executed many times in a network. We stress that in this setting, there is only one protocol that is being run. As we have mentioned, the honest parties run each protocol execution obliviously of the other executions (even though the same protocol is run each time). This is the type of composition considered, for example, in the entire body of work on concurrent zero-knowledge (cf., [17, 37]).

(b) General composition: In this type of composition, many different protocols are run together in a network. Furthermore, these protocols may have been designed independently of one another, and some may be secure while others are not. A protocol is said to maintain security under *general composition* if its security is maintained even when it is run along with other arbitrary protocols. This type of composition has been explicitly considered in [30, 7, 34, 8].

We stress a crucial difference between self and general composition. In self composition, the protocol designer has control over everything that is being run in the network. However, in general composition, the other protocols being run may even have been designed maliciously after the secure protocol is fixed.

2. **The participating parties.** Another parameter which must be considered is whether or not the same set of parties is involved in all executions:

   (a) A single set of parties: In this setting, the same set of parties participates in all executions. A further distinction when considering a single set of parties relates to the roles the parties play within each execution. (For example, in zero-knowledge there are two roles: the prover and the verifier.) We have the following two settings:

      i. Fixed roles: Here, the parties play the same roles in all executions (if a party plays the prover in one zero-knowledge execution then it always plays the prover).

      ii. Arbitrary roles: Here, the parties' roles are not fixed and may vary from execution to execution.

   (b) Arbitrary sets of parties: In this setting, arbitrary (and possibly intersecting) sets of parties run each protocol execution. In this setting, arbitrary roles is always considered.

3. **The scheduling.** There are three main types of scheduling that appear in the literature:

   (a) Sequential: each new execution begins strictly after the previous one terminates.

   (b) Parallel: all executions begin at the same time and proceed at the same rate (i.e., in a synchronous fashion).

   (c) Concurrent: the scheduling of the protocol executions, including when they start and the rate at which they proceed, is maliciously determined by the adversary. That is, the adversary has full control over when messages sent by the parties are delivered (as is the case in an asynchronous network).

   Another variant of scheduling is the timing model where, loosely speaking, the adversary is somewhat limited in how long it can delay the delivery of messages.

4. **The number of executions.** This parameter refers to the question of how many times a secure protocol should run. There are two main variants:

   (a) Unbounded: the number of executions of the secure protocol may be any polynomial in the security parameter. This is the default notion.

   (b) Bounded: in this model, an explicit a priori bound on the number of concurrent executions is known. Furthermore, a protocol only needs to remain secure if the number of concurrent executions does not exceed this bound (and so, the protocol design may depend on the specific bound). When this bound is m, we say that a protocol is secure under m-bounded composition.

We stress that this bound refers to the number of times the *secure protocol* is run. Thus, in the case of general composition, there is no bound on the arbitrary protocol running alongside the secure protocol.

Universal Composability (UC). Recently, a robust notion of security, called universal composability, was put forward [8]. This definition follows the standard paradigm of comparing a real protocol execution to an ideal execution where a trusted third party helps the participating parties carry out the computation.[1] However, it also differs in a very important way. The traditional model considered for secure computation includes the parties running the protocol, plus an adversary $\mathcal{A}$ that controls a set of corrupted parties. However, in the framework of universal composability, an additional adversarial entity called the environment $\mathcal{Z}$ is introduced. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to UC compute a given functionality f if for every real-model adversary $\mathcal{A}$, there exists an ideal-model adversary $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that are running in the ideal model for f. (In a sense, $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and an ideal execution involving a trusted party.) The importance of this new definition is due to a "composition theorem" that states that any universally composable protocol remains secure under *unbounded concurrent general composition with arbitrary sets of parties* [8]. Note that throughout this paper, when we refer to "universal composability" we mean the above *security definition*, and not the composition operation (which we call general composition). In order to emphasize this, from here on we refer to the definition of universal composability as "UC security" (or, alternatively, as the "UC definition").

It has been shown that in the case of an *honest majority*, UC secure protocols exist for any multiparty functionality [8] (building on [6, 36]). Furthermore, in the *common reference string model*,[2] any functionality can be UC computed, for any number of corrupted parties [13]. On the negative side, it has also been shown that in the plain model (i.e., without a common reference string or any other trusted setup phase), it is impossible to obtain UC-secure protocols for a large class of *two-party* functionalities [10, 8, 12]. (This class includes, for example, functionalities that do not completely reveal the parties' inputs. A specific example of a two-party functionality that cannot be UC computed is Yao's classic millionaires problem [38].)

General composition and UC security. In this paper, we study the relationship between general composition and UC security. As shown in [8], UC security implies security under concurrent general composition. However, it is not known whether or not the reverse also holds. Of course, the answer to this question depends on the specific formulation of concurrent general composition that is considered. Nevertheless, our definition here is aimed to be as minimalistic (i.e., weak) as possible, while still remaining within the standard ideal/real simulation paradigm. Resolving this question is of importance for the following two reasons:

[1] This well-established paradigm of defining security can be described as follows. An ideal execution is first defined. In such an execution, the parties hand their inputs to a trusted third party, who simply computes the desired functionality, and hands each party its designated output. Clearly, in this ideal model, the adversary's ability to carry out an attack is severely limited. Security is then formulated by requiring that the adversary should not be able to do any more harm in a real execution of the protocol than in the above ideal execution. More formally, an ideal-model adversary (or simulator) should be able to emulate a real execution of the protocol for a real adversary.

[2] In the common reference string model, all parties are given access to a common string that is ideally chosen from some efficiently samplable distribution. Essentially, this means that a trusted setup phase is assumed.

1. *Feasibility for concurrent general composition:* As we have mentioned, it has been shown that a large class of two-party functionalities cannot be UC computed in the plain model. Now, on the one hand, if security under concurrent general composition implies UC security, then we would immediately derive broad impossibility results for *any definition* that implies security under concurrent general composition. On the other hand, if security under concurrent general composition does not imply UC security, then this would mean that the question of feasibility for concurrent general composition is still wide open.

2. *Optimality of the UC definition:* In general, security definitions must be strong enough to guarantee the desired security properties, but without being so restrictive that protocols that are actually secure are ruled out. (On one extreme, one can define all protocols to be not secure, and then all adversarial attacks on "secure protocols" are vacuously thwarted.) Thus, a natural question to ask is whether or not the UC definition is *overly* restrictive. If security under concurrent general composition implies UC security, then this would prove that the UC definition is optimal (in the sense that any definition achieving security under concurrent general composition implies UC security). We note that overly restrictive definitions place an unnecessary burden on protocol design, and may have ramifications, for example, on the efficiency that can be achieved. Thus, this question is of interest independently of the question of feasibility (discussed above). That is, even when working in a model where the feasibility of UC security has been demonstrated (like the common reference string model), it is still important to know that the UC definition is "optimal".

We note that the question of whether or not concurrent general composition implies UC security was first posed in [19].

Before proceeding, it is worthwhile to consider why one may think that the UC definition is possibly too strict. Two important features of the UC definition are that the ideal-model simulator has only black-box access to the environment, and furthermore, the environment cannot be "rewound". Since the real-model adversary and environment can maliciously work together in attacking a protocol, this means that the simulator has to be able to work while being given only black-box access and no rewinding capability. However, for black-box simulation, rewinding is essential. This observation was used by [12] to demonstrate broad impossibility results for obtaining UC security in the two-party case. Thus, the UC definition is significantly more restrictive than previous definitions. Furthermore, the known impossibility results for UC security are due to these restrictions. It is therefore natural to wonder whether or not the two stringencies of *black-box access* and *no rewinding* in the UC definition are really essential for obtaining security under concurrent general composition.

1.2 Our Results

In order to describe our results, we first informally define two notions:

- Specialized-simulator UC: This is a relaxed variant of UC security that is exactly the same as the original definition of UC, except that the order of quantifiers between the environment and ideal-model simulator is reversed. That is, the UC definition requires the existence of a universal simulator that can simulate for all environments. In contrast, in specialized-simulator UC, a *different simulator* is allowed for every environment (the idea of reversing quantifiers in order to relax security requirements was studied in a different context in [16]). We stress that apart from the reversal of quantifiers, all other aspects of the definition remain unchanged. In particular, under the specialized-simulator definition, the simulator is still given only "on-line" access to the environment (i.e., black-box access without rewinding), and must work for every auxiliary input that the environment may receive. Note that the fact that the simulator can be different for every environment means that it essentially knows the environment's code. However, since it has only black-box access to the environment, it cannot see the environment's auxiliary input or its random tape (both of which are given to nonblack-box simulators).

- ℓ-bounded general composition: We define a restricted version of general composition, where the number of secure protocols that are executed is bounded. That is, denote by ℓ-bounded general composition the setting where at most $\ell(n)$ secure protocols are run concurrently with arbitrary other protocols, and n denotes the security parameter. (As we have mentioned above, standard general composition considers the case that ℓ can be any polynomial.) We stress that ℓ bounds only the number of executions of secure protocols; the number of arbitrary protocols running in the network is unbounded (thus, the bound refers to how many times the secure protocol can be composed).

Before proceeding, we remark that the difference between specialized-simulator UC and regular UC depends on some subtle details regarding how polynomial-time is defined. Indeed, the latest version of the UC paper [9] proposes a different notion of polynomial-time (see Appendix A) and proves the following:

Claim 1.1 (proven in [9]): *According to the notion of polynomial-time in [9], a protocol is secure under specialized-simulator UC if and only if it is UC secure.*

As we will see, our results are not sensitive to the complexity class of the parties, as long as *all parties* including the adversary and environment belong to the same class (as is the case in all UC versions). We have therefore chosen to present our results with respect to specialized-simulator UC. This gives the most generality because our results then hold whether or not specialized-simulator UC is equivalent to UC. In our results below we will refer to robust complexity classes $\mathcal{C}$. This just means that if a machine M is in the class $\mathcal{C}$ and runs in time $t(|x|)$ on input x, then a machine M' that runs in time $p(t(|x|))$ for some polynomial $p(\cdot)$ is also in $\mathcal{C}$. This clearly holds for any reasonable notion of polynomial-time. In particular, it holds for the notion of polynomial-time in the length of the input, the notion of polynomial-time in the security parameter (as defined here in Section 2), and the new notion of polynomial-time defined in [9].

The main contribution of this paper is a proof that security under 1-bounded concurrent general composition (where only a single secure protocol is run in the network) implies security under specialized-simulator UC security. That is:

Theorem 1.2 (main theorem – informally stated): *Let $\mathcal{C}$ be any robust complexity class and consider definitions in which all parties (honest and adversarial) are in $\mathcal{C}$. Then, any protocol that is secure under* 1*-bounded concurrent general composition is also secure under specialized-simulator UC.*

We actually prove something more general: we show that Theorem 1.2 holds even when the parties are given access to an ideal functionality that can be used to model some trusted setup (like a common reference string), or authenticated channels. See below (under "optimality of the UC definition") as to why this is of importance.

The intuition as to why this implication holds, despite the stringencies of the UC definition described above, is as follows. The environment $\mathcal{Z}$ captures the behavior of the honest parties in all of the protocols being run except for the secure protocol itself. Now, in a real-world setting, the adversary clearly cannot have any control over the honest parties, or else all meaning of security is lost. Thus the adversary can only interact with the honest parties according to the standard method of communication (send messages and receive back responses). This method of communication is exactly "straight-line black box". This explains why the adversary also interacts with the environment $\mathcal{Z}$ in the UC definition in a straight-line black-box manner. Our proof of Theorem 1.2 demonstrates this intuition in a formal way.

Observe that Theorem 1.2 refers to a relaxed variant of UC security (specifically, specialized-simulator UC) and a relaxed variant of general composition (specifically, 1-bounded general composition). Nevertheless, as will be discussed below, it provides substantial progress in answering both of the above questions regarding "feasibility for general composition" and "optimality of the UC definition". We remark that our formal statement of Theorem 1.2 in Section 3.2 includes the case of setup (like a key registration functionality or common reference string). This is important with respect to the optimality of the UC definition (because by [12] many UC-secure protocols must use some form of setup).

Corollary 1 – impossibility for concurrent general composition. In order to use Theorem 1.2 to derive impossibility results for concurrent general composition, we first note that many of the impossibility results of [12] for UC security hold with respect to specialized-simulator UC. This is explicitly stated in [12]. Thus, combining Theorem 1.2 and [12], we obtain the following corollary:

Corollary 1.3 (impossibility of general composition – informal): *There exist large classes of two-party functionalities for which it is impossible to obtain protocols (in the plain model) that remain secure under* 1*-bounded concurrent general composition.*

Observe that this corollary relates to a *very restricted* type of concurrent general composition. This significantly strengthens our impossibility results. We stress that Corollary 1.3 holds unconditionally and for any type of simulation (i.e., it is *not* restricted to "black-box" simulation). However, we do mention the following caveat. The impossibility result holds for a *specific* security definition of concurrent general composition (that, as we show, implies specialized-simulator UC). This definition is a natural one, and seems to be as weak as possible while still capturing the notion of security under composition with arbitrary protocols (see Definition 1). Nevertheless, it is always possible that a different definition can be formulated for which the impossibility result will not hold. In particular, we do not consider definitions that are not formulated via the simulation paradigm.[3]

In addition to the impossibility result stated above, we also prove impossibility for 1*-bounded parallel general composition*. This may seem somewhat surprising since parallel composition is usually considered easier to achieve than concurrent composition (and this is definitely true for the case of self composition, as pointed out in [22, Section 6]). Nevertheless, we show that in the context of general composition, even parallel composition is impossible to achieve.

Corollary 2 – optimality of the UC definition. Theorem 1.2 explicitly states that any definition that guarantees security under concurrent general composition implies specialized-simulator UC security. Thus, specialized-simulator UC is a *minimal* definition, if security under concurrent

[3]We note that it is easy to come up with a security definition for which security is preserved under concurrent general composition; simply take the definition that *all* protocols are secure. However, such a definition provides no security guarantees whatsoever. Therefore, the question is whether or not it is possible to provide a "meaningful" definition for which the impossibility results do not hold. We rule out this possibility for definitions which provide the security guarantees of the ideal/real model simulation paradigm, as defined in Definition 1. Whether or not it is possible to provide a meaningful, weaker definition is still open, with some progress made by [35, 4, 29].

general composition is to be achieved. When considering polynomial-time parties as defined in [9], we can combine Theorem 1.2 with Claim 1.1 in order to obtain the following corollary:

Corollary 1.4 *When all parties run in polynomial-time as in [9], it holds that any protocol that is secure under 1-bounded concurrent general composition is also UC secure.*

Thus, it follows that UC security is *minimal*, at least when working with polynomial-time as defined in [9]. This also holds when some trusted setup is used, like a common reference string or key registration functionality. Thus, there is no "penalty" incurred by proving the security of protocols in the UC model.

Specialized-simulator UC implies bounded general composition. Theorem 1.2 states that 1-bounded concurrent general composition implies specialized-simulator UC security. But what about the opposite direction? It can be shown that for any robust complexity class, any protocol that is secure under specialized-simulator UC is secure under $O(1)$-bounded concurrent general composition (this was proven in previous versions of this work). Given the new results of Canetti in [9] (proving equivalence of specialized-simulator UC and regular UC for the new notion of polynomial-time), we have omitted the proof of this implication here.

Setup assumptions and alternative models for UC. Our results here show that in order to achieve UC security, the basic model must somehow be modified or augmented. This justifies the search for trusted setup assumptions (like the common reference string mentioned above) or alternative definitions in order to achieve security under concurrent general composition. Indeed, subsequently to the initial publication of this work [25], additional setup assumptions and modifications to the model were considered. These are discussed below.

1.3 Related and Subsequent Work

Until 2001, most of the research on the topic of concurrent composition of protocols considered the *self composition* of *specific* problems. A considerable bulk of this work studied concurrent zero-knowledge; e.g., see [17, 37, 11]. The question of concurrent self composition of general secure computation was studied in [24, 33, 27, 32, 29] but is not our subject here. In this paper, we focus on the *general composition* of *general* secure multiparty protocols. The first work to consider security under *concurrent* general composition was [34], who considered the case that a secure protocol is executed *once* concurrently with another arbitrary protocol. ([34] provided a definition and a composition theorem, but no proof of feasibility.) The unbounded case, where a secure protocol can be run *any* polynomial number of times in an arbitrary network, was then considered by the framework of universal composability [8]. Broad impossibility results for UC security in the plain model were demonstrated in [10, 12], thus opening the question as to whether the UC definition can be weakened.

Subsequent to this work, alternatives to UC security were considered with the aim of bypassing the impossibility results of concurrent general composition. Following [31] – who showed that problems of concurrency can be solved by providing the simulator with more time than the adversary – it was shown in [35] that by providing the simulator with "additional powers" it is possible to bypass the impossibility results for UC in the plain model. This work was later extended in [4] and [28]. We remark that such solutions provide weaker (and rather unclear) security guarantees. Other alternatives have been to look for different setup assumptions as in [3] who presented a PKI-type infrastructure, and to augment the model with time as in [23].

1.4 Organization

For the sake of clarity, we present our results in stages. In Section 3.1 we first prove a basic theorem stating that security under 1-bounded concurrent general composition implies specialized-simulator UC security, in the case of *static adversaries* that corrupt at least one party and where there are *ideally secure channels* that connect the parties. Considering static adversaries and ideal channels enables us to provide a simpler proof that highlights the essence of why the implication holds. Following this, in Section 3.2 we prove our main theorem that 1-bounded concurrent general composition implies specialized-simulator UC security for adaptive adversaries. Furthermore, we prove that this holds when the parties have access to an ideal functionality that can be used to provide some trusted setup (like a common reference string). Both of the above proofs consider parties that run in time that is polynomial in the security parameter. In Section 3.3 we show that the theorem holds also for robust complexity classes, as required for obtaining Corollary 1.4. Then in Section 3.4 we show that the implication can even be proved from 1-bounded parallel composition to a version of UC security for which the impossibility results of [12] still hold. Finally, in Section 4 we state the impossibility results for concurrent general composition that are derived by combining our proofs with the results of [12].

2 Definitions

In this section, we present a definition of security for concurrent general composition and provide an overview of the definition of universal composability. Both definitions follow the standard simulation-based paradigm for security by comparing a real execution to an ideal process [21, 30, 5, 7]. Our presentation assumes basic familiarity with these definitions of secure multiparty computation.

For the sake of clarity, and in order to be concrete, we will present a definition of security in which all parties (honest and adversarial) run in time that is *polynomial in the security parameter.* Nevertheless, the entire definition can trivially be repeated for any notion of polynomial-time, or any other complexity class desired. The only difference is the bounds on the running-time of the parties.

Notation and conventions. The security parameter is denoted by n, and all parties and adversaries run in time that is polynomial in n. (Formally, all parties have an additional "security parameter tape" upon which n is written in unary. Furthermore, all parties run in time that is polynomial in the length of the input on the security parameter tape and not on their ordinary input tape.) A function $\mu(\cdot)$ is negligible in n (or just negligible) if for every polynomial $p(\cdot)$ there exists a value N such that for all $n > N$ it holds that $\mu(n) < 1/p(n)$. Let $X = \{X(n,a)\}_{n\in\mathsf{N},a\in\{0,1\}^*}$ and $Y = \{Y(n,a)\}_{n\in\mathsf{N},a\in\{0,1\}^*}$ be distribution ensembles. Then, we say that X and Y are computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$, if for every probabilistic polynomial-time distinguisher D there exists a function $\mu(\cdot)$ that is negligible in n, such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(n,a)) = 1] - \Pr[D(Y(n,a)) = 1]| < \mu(n)$$

When X and Y are equivalent distributions we write $X \equiv Y$, and when they are statistically close we write we write $X \stackrel{s}{\equiv} Y$. Note that the length of a is not bounded; however, by our above convention all parties are polynomial in n (even if they receive a as input and a is of super-polynomial length).

The definition of computational indistinguishability provided here is *uniform* and in our formal treatment we deal with uniform machines only. All of our results carry over to the non-uniform model as well.

2.1 Security Under General Composition

We now present a definition of security for multiparty computation that is based *directly* on the general composition operation. That is, we define a protocol to be secure if it remains secure under concurrent general composition. This is in contrast to the methodology whereby some stand-alone definition of security is presented, and then a general composition theorem is shown to hold for this definition. This latter approach is advantageous because it makes proving security easier: it suffices to prove security in a stand-alone setting only, security under composition is automatic. However, as we have mentioned, such an approach has the danger of possibly overshooting the security requirements. Specifically, there may exist protocols that are secure under the desired composition operation, and yet are not secure for the stand-alone definition.[4] By defining a protocol to be secure if it is secure under concurrent general composition, we avoid this potential pitfall. The formalization of general composition used in our definition is based on [7, 8].

Motivating discussion. In the setting of concurrent general composition, a secure protocol ρ is run concurrently (possibly many times) with an arbitrary other protocol π. This protocol π represents *all the activity* in the network in which ρ is being run. In addition, π may determine the inputs to ρ and use its outputs. This is consistent with the fact that the outcome of some protocols is likely to influence the inputs of other protocols. (For example, if a party wins in an auction protocol, but payed far more than it originally thought necessary, then this party may agree to bid less in a subsequent auction. Likewise, the outcome of one election can have a significant influence on our choice of candidate in another election.) This "transfer" of inputs and outputs between protocols is very strong, and is a crucial ingredient in proving the equivalence of concurrent general composition and UC security. However, we believe that it accurately models the behavior of real networks (or, at least, it would be highly undesirable to guarantee security only in the case that parties decide future inputs independently of past outputs).

One way to model this setting is to directly consider the concurrent composition of the two protocols π and ρ. An alternative way to model this is to consider π to be a "calling" or "controlling" protocol which, among other things, contains "subroutine calls" to the protocol ρ (or to the functionality that ρ computes). The calling protocol π determines the inputs to ρ and uses the resulting outputs. We denote such a composition of π with ρ by π^ρ. Note that messages of π may be sent concurrently to the execution of ρ (even though π "calls" ρ). This specific formalization has the appearance of π being a protocol that has been designed to perform a given task, and that uses subroutine calls to ρ in order to complete this task. We stress that although this is one interpretation, it also models the case that π is merely anarchic activity in a network in which ρ is being executed.

[4] An example of where such an "overshoot" occurred was regarding the *sequential* general composition of secure multiparty protocols. Specifically, [30] presented a definition for stand-alone security and showed that sequential general composition holds for any protocol meeting this definition. However, the definition of [30] is very restrictive; among other things, it requires "one-pass black-box simulation" (i.e., black-box simulation without rewinding). Later, it was shown in [7] that sequential composition holds for a less restrictive definition (where simulation need not be black-box). Furthermore, there exist protocols that are secure by the definition of [7] (and thus compose sequentially), and yet are not secure by the definition of [30]. Thus, at least as far as sequential composition is concerned, the definition of [30] is "overly restrictive".

We reiterate the fact that inputs and outputs are transferred between the protocols π and ρ and that this is stronger than a case where π and ρ have predetermined inputs and the honest parties do not transfer any information between the executions.[5] Furthermore, our equivalence between concurrent general composition and UC security depends heavily on this stringency. Nevertheless, as argued above, this stringency is important for modeling the security needs of modern networks.

Multiparty computation. A multiparty protocol problem for a set of parties $P_1, \ldots, P_m$ is cast by specifying a (probabilistic polynomial-time) multiparty ideal functionality machine $\mathcal{F}$ that receives inputs from parties and provides outputs. Note that $\mathcal{F}$ can also be reactive, in which case, inputs and outputs are provided in a number of stages. The aim of the computation is for the parties to jointly compute the functionality $\mathcal{F}$.

Adversarial behavior. In this work we consider malicious adversaries. That is, the adversary controls a subset of the parties who are said to be corrupted. The corrupted parties follow the instructions of the adversary in their interaction with the honest parties, and may arbitrarily deviate from the protocol specification. The adversary also receives the view of the corrupted parties at every stage of the computation. We consider both static and adaptive corruptions. In the static (or non-adaptive) adversarial model, the set of corrupted parties is fixed for any given adversary. In contrast, an adaptive adversary may corrupt parties during the computation and as a function of what it sees. Finally, we consider a model where the adversary has full control over the scheduling of the delivery of all messages. Thus, the network is asynchronous. This adversarial control over the message scheduling also implies that we consider concurrent composition.

We remark that our way of defining static adversaries is rather non-standard. We define that the set of corrupted parties is fixed for the adversary. That is, the set of corrupted parties $I \subseteq [m]$ is fixed and does not depend on its auxiliary input or random tape (this is the way two-party computation with static adversaries is defined in [18]). We rely on this more stringent notion of static adversaries in order to prove the equivalence between concurrent general composition and UC security with a *single set of parties*. Nevertheless, when considering *arbitrary sets of parties*, our results hold for adaptive adversaries (and thus for the standard definition of static adversaries where the same adversary can be associated with different sets of corrupted parties).

The hybrid model. Let π be an arbitrary protocol that utilizes ideal interaction with a trusted party computing a multiparty functionality $\mathcal{F}$ (recall that π actually models arbitrary network activity). This means that π contains two types of messages: standard messages and ideal messages. A standard message is one that is sent between two parties that are participating in the execution of π, using the point-to-point network (or broadcast channel, if assumed). An ideal message is one that is sent by a participating party to the trusted third party, or from the trusted third party to a participating party. This trusted party runs the code for $\mathcal{F}$ and associates all ideal messages with $\mathcal{F}$. Notice that the computation of π is a "hybrid" between the ideal model (where a trusted party carries out the entire computation) and the real model (where the parties interact with each other only). Specifically, the messages of π are sent directly between the parties, and the trusted party is only used in the ideal calls to $\mathcal{F}$.

[5] This issue should not be confused with the issue of "stateless" versus "stateful" composition. We consider stateless composition here, and honest parties run each execution of ρ independently of other executions of ρ and independently of messages obtained in π. However, the *input* to ρ (and only the input) may be influenced by prior network activity. Likewise, the output of an execution of ρ (and only the output) may influence later activity. Of course, all of this holds only for the honest parties; adversarial parties may act as they wish.

As we have mentioned, the adversary controls the scheduling of all messages, including both standard and ideal messages. In our main proof, we do not assume anything about the communication channels connecting the parties (i.e., we do not assume that they are private or even authenticated). Thus, the standard messages may be tampered with by the adversary. Nevertheless, the theorem is stated also when the parties have access to an ideal functionality as above and this ideal functionality can be used to achieve authenticated communication between the parties. In contrast, the channels connecting the participating parties and the trusted third party computing $\mathcal{F}$ are both *authenticated and private.* Thus, the adversary cannot read the ideal messages, even though it delivers them. Actually, sometime it is useful to define each ideal message as having a public header and a private body (see [13, 26]), but this is of no consequence to our results here.

Computation in the hybrid model proceeds as follows. In the static corruption model, the computation begins with the adversary receiving the inputs and random tapes of the corrupted parties. Throughout the execution, the adversary controls these parties and can instruct them to send any standard and ideal messages that it wishes. In the adaptive corruption model, the adversary can choose to corrupt parties throughout the computation. Upon corruption, the adversary receives the party's internal state and then controls the party for the remainder of the computation. In addition to controlling the corrupted parties, the adversary delivers all the standard and ideal messages by copying them from outgoing communication tapes to incoming communication tapes. Formally, we assume that each pair of parties (including the trusted party) has a matching pair of outgoing and ingoing communication types. Then, when a party P_i writes a message on the outgoing communication tape that is dedicated to P_j, the adversary delivers this message to P_j by copying it to the ingoing communication tape of P_j that is dedicated to P_i. Another technicality (and one that can be ignored later) is that we assume that the number and length of messages from all parties is known, and that honest parties only read the expected number of messages of the expected length. This is necessary to prevent the adversary from bombarding a party with so many messages that it expends all of its (a priori fixed polynomial) running time processing these garbage messages. As we have mentioned, this can be ignored from here on.

The honest parties always follow the specification of protocol π. Specifically, upon receiving a message (delivered by the adversary), the party reads the message, carries out a local computation as instructed by π, and writes standard and/or ideal messages to its outgoing communication tapes, as instructed by π. At the end of the computation, the honest parties write the output value prescribed by π on their output tapes, the corrupted parties output a special "corrupted" symbol and the adversary outputs an arbitrary function of its view. Let n be the security parameter, let $\mathcal{S}$ be an adversary for the hybrid model with auxiliary input z, and let $\overline{x}$ be the vector of the parties' inputs to π (note that $\overline{x} = (x_1, \ldots, x_m)$ and each $x_i \in \{0,1\}^*$). Then, the hybrid execution of π with ideal functionality $\mathcal{F}$, denoted $\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}}(n, \overline{x}, z)$, is defined as the output vector of all parties and $\mathcal{S}$ from the above hybrid execution.

The real model – general composition. Let ρ be a multiparty protocol for computing the functionality $\mathcal{F}$. Intuitively, the composition of protocol π with ρ is such that ρ takes the place of the ideal call to $\mathcal{F}$. Formally, each party holds a separate probabilistic interactive Turing machine (ITM) that works according to the specification of the protocol ρ for that party. When π instructs a party to send an ideal message α to the ideal functionality $\mathcal{F}$, the party writes α on the input tape of its ITM for ρ and invokes the machine. Any message that it receives that is marked for ρ, it forwards to this ITM, and all other messages are answered according to π. Finally, when the execution of ρ concludes and a value β is written on the output tape of the ITM, the party copies β to the incoming communication tape for π, as if β is an ideal message (i.e., output) received from

$\mathcal{F}$. This composition of π with ρ is denoted π^ρ and takes place without any trusted help. Thus, the computation proceeds in the same way as in the hybrid model, except that all messages are standard. (Note that like in the hybrid model, the adversary controls message delivery and can also read messages sent, but cannot modify or insert messages.) Let n be the security parameter, let $\mathcal{A}$ be an adversary for the real model with auxiliary input z, and let $\overline{x}$ be the vector of the parties' inputs to π. Then, the real execution of π with ρ, denoted $\text{REAL}_{\pi^\rho,\mathcal{A}}(n,\overline{x},z)$, is defined as the output vector of all the parties and $\mathcal{A}$ from the above real execution.

Security as emulation of a real execution in the hybrid model. Having defined the hybrid and real models, we can now define security of protocols. Loosely speaking, the definition asserts that for any context, or calling protocol π, the real execution of π^ρ emulates the hybrid execution of π which utilizes ideal calls to $\mathcal{F}$. This is formulated by saying that for every real-model adversary there exists a hybrid-model adversary for which the output distributions are computationally indistinguishable. The fact that the above emulation must hold for *every* protocol π that utilizes ideal calls to $\mathcal{F}$, means that *general composition* is being considered (recall that π represents arbitrary network activity). In the definition, we distinguish between the case that π is not restricted and may thus utilize any polynomial number of calls to $\mathcal{F}$, and the case that π utilizes only a bounded number of calls to $\mathcal{F}$. We also distinguish between the case that all parties running π also run ρ, and the case that only a subset of the parties run ρ. The case that only a subset of the parties run ρ accurately models the setting of modern networks. In such a setting, π represents all of the network activity outside of the secure protocol, and ρ is the execution of the secure protocol by some subset of the parties within the network.

Definition 1 (security under concurrent general composition): *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* securely computes $\mathcal{F}$ under concurrent general composition *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideal calls to $\mathcal{F}$ and every probabilistic polynomial-time real-model adversary $\mathcal{A}$ for π^ρ, there exists a probabilistic polynomial-time hybrid-model adversary $\mathcal{S}$ such that for every $\overline{x}, z \in \{0,1\}^*$:*

$$\left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}}(n,\overline{x},z)\right\}_{n\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi^\rho,\mathcal{A}}(n,\overline{x},z)\right\}_{n\in\mathbb{N}}$$

If we restrict the protocols π to those that utilize at most ℓ ideal calls *to $\mathcal{F}$, then ρ is said to securely compute $\mathcal{F}$ under* ℓ-bounded concurrent general composition.

If both π and ρ are defined for m parties $P_1,\ldots,P_m$, then security is said to hold for a single set of parties. *If π is defined for m parties and ρ may be defined for less than m parties, then security is said to hold for* arbitrary sets of parties.

Remark. Arguably, the notion of concurrent general composition is best captured when π can utilize any number of ideal calls to $\mathcal{F}$. Indeed, this provides a more reasonable security guarantee. However, we introduce the restricted notion of *bounded* concurrent composition because in order to strengthen our impossibility results, we wish to capture the notion of general composition in the *weakest way possible* (while still being meaningful). Thus, we will consider the case that the calling protocol π utilizes only a *single* call to $\mathcal{F}$ (i.e., 1-bounded concurrent general composition).

2.2 Universal Composability (UC) and Specialized-Simulator UC

In this paper we show that security under concurrent general composition implies a relaxed variant of the definition of universal composability. The only difference between the original and relaxed definitions is with respect to the order of quantifiers between the ideal-model adversary and the environment. Specifically, in the relaxed variant a different simulator is allowed for each environment. We therefore call this definition specialized-simulator UC. In this section, we present a brief outline of the UC definition; for a full definition, see [8]. We remark that there are many possible variants of this definition that depend on the specific order of activations and the types of channels that connect the parties. Nevertheless, our results and the impossibility results of [12] that we use later are robust and hold for all known variants. Thus, we focus on a specific one here with the understanding that our concrete choices are of no importance.

The definition of universal composability follows the standard simulation paradigm based on comparing a real execution with an ideal execution involving a trusted third party. As above, the algorithm run by the trusted party is called the ideal functionality. Then, a protocol ρ for computing a functionality $\mathcal{F}$ is secure if the result of a real execution can be emulated in an ideal execution where parties interact with the ideal functionality only. Notice that only a single stand-alone execution of ρ is considered in this definition (security under composition is derived via a composition theorem).

As described in the introduction, the notion of ideal-model emulation in the UC framework includes the real-model adversary $\mathcal{A}$, an ideal-model adversary or simulator $\mathcal{S}$ and an environment $\mathcal{Z}$. The environment generates the inputs to all parties, reads all outputs, and interacts with the adversary throughout the computation. It is required that for every real-model adversary $\mathcal{A}$ attacking a real protocol execution, there exists an ideal-model adversary $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and the ideal functionality for $\mathcal{F}$. An important point to notice with respect to the UC definition is that the ideal-model adversary $\mathcal{S}$ interacts with $\mathcal{Z}$ throughout the computation in the same way that $\mathcal{A}$ does. That is, $\mathcal{S}$ has no control over $\mathcal{Z}$, who is actually an external, real party. In particular, this means that $\mathcal{S}$ cannot "rewind" $\mathcal{Z}$. We consider the case where all communication is sent via the adversary (as with our definition of general composition above) with the exception of inputs that $\mathcal{Z}$ writes to the parties' input tapes and outputs that $\mathcal{Z}$ reads from the parties' output tapes. (This is not communication in the sense of communication tapes and so is not carried out via the adversary.) In addition, the order of activations of parties is such that the environment $\mathcal{Z}$ begins the computation. Then, the next party to be activated is the party to receive an input or an incoming message. This works by transferring control as soon as the environment writes an input or the adversary delivers a message (or likewise, as soon as the environment or adversary send a message to each other).

Formal definitions of the real and ideal model executions can be found in [8]. Let n be the security parameter, let $\mathcal{Z}$ be an environment with input z, and let $\mathcal{A}$ be a real-model adversary (a vector $\overline{x}$ of inputs for the parties is not defined here because $\mathcal{Z}$ interactively provides the parties with their inputs). Then, a real-model execution of a protocol ρ with $\mathcal{Z}$, z and $\mathcal{A}$ is denoted $\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n, z)$. Likewise, let $\mathcal{S}$ be an ideal-model adversary. Then, an ideal-model execution of $\mathcal{F}$ with $\mathcal{Z}$, z and $\mathcal{S}$ is denoted $\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n, z)$. The UC definition requires that for every $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish the real and ideal executions. Thus, $\mathcal{S}$ is a *universal simulator*, where the universality is with respect to all environments. In contrast, we consider a variant of UC where a different $\mathcal{S}$ is allowed for every $\mathcal{Z}$. Thus, we reverse the quantifiers between the ideal-model simulator and environment. This reversal of quantifiers also requires a change in the convention regarding the output of the environment. In the UC definition, the environment outputs a single bit only (and this is the only output from the entire experiment). When a universal simulator is considered, this is equivalent to the case that the environment can output an arbitrary string. However, when the order of quantifiers is reversed, equivalence may no longer hold. We therefore allow the environment to output a string of arbitrary length.

In order to stress the difference between UC security and specialized-simulator UC, we present both definitions (while relying on the formal definitions of the real and ideal models that are not presented here):

Definition 2 *Let $\mathcal{F}$ be an ideal functionality and let ρ be a multiparty protocol. Then:*

- (UC security [8]): *We say that ρ* UC computes *$\mathcal{F}$ if for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a probabilistic polynomial-time ideal-process adversary $\mathcal{S}$ such that for any probabilistic polynomial-time environment $\mathcal{Z}$ and every $z \in \{0,1\}^*$,*

$$\left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}}$$

- (specialized-simulator UC): *We say that ρ* securely computes *$\mathcal{F}$* under specialized-simulator UC *if for every probabilistic polynomial-time adversary $\mathcal{A}$ and every probabilistic polynomial-time environment $\mathcal{Z}$, there exists a probabilistic polynomial-time ideal-process adversary $\mathcal{S}$ such that for every $z \in \{0,1\}^*$,*

$$\left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}}$$

We use the same convention of polynomial-time here as for general composition. That is, each party (including the environment and adversary) runs in time that is polynomial in the security parameter n. As we have discussed, the latest version of the UC framework [9] considers a notion of polynomial-time that is suited for dynamic networks and is based on parties being polynomial in the length of their input. According to this definition, it follows that *UC security is equivalent to specialized-simulator UC*; see Appendix A for more about this notion of polynomial-time.

"Universal" composition. As we have mentioned, a protocol that is universally composable is secure under concurrent general composition with arbitrary sets of parties [8]. Thus, the UC definition provides security under the "strongest" type of composition.

3 General Composition Implies Universal Composability

3.1 The Basic Theorem

In this section we prove that security under 1-bounded concurrent general composition implies specialized-simulator UC security for the case of *static adversaries that corrupt at least one party* and where *ideal secure channels* are assumed to exist between the parties. (According to our formalization all messages go via the adversary who schedules when to deliver them. In this case of ideally secure channels the adversary doesn't see the content of these messages and cannot modify them. However, it still schedules delivery.) We also consider the basic case that all parties run in time that is polynomial in the security parameter. The reason for considering this model is that the proof of equivalence is simpler and highlights the main reason for equivalence. In Section 3.2 below, we prove a more general version of this result, and in Sections 3.3 and 3.4 we present further extensions.

Theorem 3 *Let ρ be a protocol and $\mathcal{F}$ a functionality, and consider a model whereby all parties are connected via ideally secure channels. If ρ securely computes $\mathcal{F}$ under* 1*-bounded concurrent general composition with a single set of parties and static adversaries who corrupt at least one party, then ρ securely computes $\mathcal{F}$ under specialized-simulator UC with adaptive adversaries.*

Proof: The intuition behind the proof of this theorem is as follows. If a protocol is secure under 1-bounded concurrent general composition, then it can be composed with any protocol. In particular, it can be composed with a protocol in which an honest party plays the role of the UC environment $\mathcal{Z}$ and a corrupted party plays the role of the adversary $\mathcal{A}$. The ability to simulate this protocol will then imply the existence of a UC simulator for the environment $\mathcal{Z}$. We now proceed with the proof.

Let ρ be a protocol for parties $P_1, \ldots, P_m$ and $\mathcal{F}$ a functionality such that ρ securely computes $\mathcal{F}$ under 1-bounded concurrent general composition with a single set of parties and in the presence of static adversaries who corrupt at least one party. Note that if an adversary corrupts all parties then security holds by triviality. Thus, from here on we also assume that the adversary does not corrupt all parties. We wish to show that ρ securely computes $\mathcal{F}$ under specialized-simulator UC. This involves constructing a UC-type simulator $\mathcal{S}$ for every static adversary $\mathcal{A}$ of the form above and every environment $\mathcal{Z}$. Thus, fix an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. Furthermore, without loss of generality, let P_1 be a party who is corrupted by $\mathcal{A}$ and let P_m be a party who is not corrupted by $\mathcal{A}$. In order to construct $\mathcal{S}$, we use the fact that ρ is secure under 1-bounded concurrent general composition, and is thus secure when composed with any protocol. In particular, it is secure when composed with a protocol π that includes emulation of the environment $\mathcal{Z}$ and adversary $\mathcal{A}$. We now define such a protocol π, involving parties $P_1, \ldots, P_m$. Loosely speaking, P_m who is not corrupted will play the role of the UC environment $\mathcal{Z}$, and P_1 who is corrupted will play the role of the UC (real-model or ideal-model) adversary $\mathcal{A}$.

Protocol π for the $\mathcal{F}$-hybrid model is defined as follows:

1. Inputs: Party P_m receives a value z for input; all other parties receive no input.

2. Instructions for P_m: Upon input z, party P_m internally invokes the environment $\mathcal{Z}$ with input z. When $\mathcal{Z}$ wishes to send a message α to the adversary that it interacts with, party P_m sends the message $(\mathcal{Z}, \mathcal{A}, \alpha)$ to P_1; likewise, when P_m receives a message of the form $(\mathcal{A}, \mathcal{Z}, \beta)$ from P_1, party P_m internally hands β to $\mathcal{Z}$ as if it is from the adversary that $\mathcal{Z}$ interacts with. When $\mathcal{Z}$ writes a value x_i intended for the input tape of a party P_i $(i \neq m)$, then party P_m sends the message (input, x_i) to P_i. During the execution, P_m receives tuples of the form $(\mathsf{output}, P_i, y_i)$ from the parties $P_1, \ldots, P_{m-1}$ (see step (3) below). When $\mathcal{Z}$ wishes to read the output tape of a party P_i, then if P_m has received such an output message from P_i, it internally hands $\mathcal{Z}$ the value y_i. Otherwise, it internally hands $\mathcal{Z}$ a blank message λ (to be interpreted as if P_i has not yet written to its output tape).

   In addition to the above, when $\mathcal{Z}$ writes a value x_m intended for the input tape of P_m, party P_m sends x_m to the ideal functionality $\mathcal{F}$. When it receives back its output y_m, it stores it and gives it to $\mathcal{Z}$ whenever it wishes to read the output tape of P_m.

3. Instructions for P_i (for every $1 \leq i \leq m-1$): When party P_i receives a message (input, x_i) from P_m, it sends x_i to the ideal functionality $\mathcal{F}$. When P_i receives its output y_i from $\mathcal{F}$, it sends the message $(\mathsf{output}, P_i, y_i)$ to P_m. (Recall that these messages go through the adversary who controls delivery.)

4. Output: Party P_m outputs whatever $\mathcal{Z}$ outputs. All other parties output the empty string λ.

Recall that in the composed protocol π^ρ, the parties $P_1, \ldots, P_m$ run an execution of ρ instead of sending their inputs to the functionality $\mathcal{F}$.

Given the protocol π, we define a probabilistic polynomial-time adversary $\mathcal{A}_\pi$ who attacks the composed, real-model protocol π^ρ. The idea of $\mathcal{A}_\pi$ is to set up a scenario which perfectly emulates the UC real-model setting. $\mathcal{A}_\pi$ corrupts party P_1 and in addition to running the instructions of P_1 described above, it internally runs the code of the UC adversary $\mathcal{A}$ (as fixed above), and uses the ability of P_1 to communicate with P_m to effectively enable communication between $\mathcal{A}$ and $\mathcal{Z}$. That is, when the internal $\mathcal{A}$ wishes to send a message α to $\mathcal{Z}$, adversary $\mathcal{A}_\pi$ instructs P_1 to send $(\mathcal{A}, \mathcal{Z}, \alpha)$ to P_m. Likewise, when P_1 receives a message of the form $(\mathcal{Z}, \mathcal{A}, \beta)$ from P_m, adversary $\mathcal{A}_\pi$ passes β to $\mathcal{A}$ as if $\mathcal{A}$ received it from $\mathcal{Z}$. Regarding the execution of ρ, whenever $\mathcal{A}$ instructs a corrupted party P_i to send a message to some party P_j, adversary $\mathcal{A}_\pi$ instructs P_i to send the same message. In addition, $\mathcal{A}_\pi$ delivers messages between the parties whenever $\mathcal{A}$ would deliver them. Finally, messages of the form (input, x_i) sent by P_m and $(\mathsf{output}, P_i, y_i)$ sent to P_m are all delivered by $\mathcal{A}_\pi$ *immediately*. This completes the definition of $\mathcal{A}_\pi$.

We now claim that for every UC adversary $\mathcal{A}$ and UC environment $\mathcal{Z}$, the output of P_m from an execution of π^ρ with $\mathcal{A}_\pi$ is distributed exactly like the output of $\mathcal{Z}$ after an execution of ρ with $\mathcal{A}$ in the UC model. That is, for every $z \in \{0,1\}^m$ let $\overline{z}$ be a vector of inputs where for every $i \neq m$ we have $z_i = \lambda$ (the empty input), and $z_m = z$. Then, we claim for every $z \in \{0,1\}^*$,

$$\left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n, z)\right\}_{n\in\mathsf{N}} \equiv \left\{\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{z}, \lambda)\,|_{P_m}\right\}_{n\in\mathsf{N}} \tag{1}$$

where $\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{x}, z)\,|_P$ denotes the output of the party P from the REAL execution of π^ρ. Eq. (1) follows from the following observations. First, P_m perfectly emulates the environment $\mathcal{Z}$. Therefore, the inputs of the parties $P_1, \ldots, P_m$ to ρ in the real execution of π^ρ are identical to the inputs of the parties to ρ in the UC setting (recall that P_m provides these inputs in π^ρ whereas $\mathcal{Z}$ provides these inputs in the UC setting). Second, the view of $\mathcal{A}$ in the emulation by $\mathcal{A}_\pi$ is identical to its view in the UC setting. Therefore, the behavior of the corrupted parties is identical in both settings. Finally, the honest parties in π^ρ behave exactly the same as in an execution of ρ. Putting this together, we have that the view of $\mathcal{Z}$ in the emulation by P_m in π^ρ is identical to its view in the UC setting. We conclude that the distribution over P_m's output is identical to the distribution over $\mathcal{Z}$'s output.

Having established the connection between a UC-REAL execution of ρ and a REAL execution of π^ρ, we proceed to show how a UC-type simulator is obtained for ρ. First, by the security of ρ under 1-bounded concurrent general composition with a single set of parties and static adversaries corrupting at least one party, we have that there exists a hybrid-model simulator $\mathcal{S}_\pi$ such that for all $\overline{x}, w \in \{0,1\}^*$,

$$\left\{\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{x}, w)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n, \overline{x}, w)\right\}_{n\in\mathsf{N}}$$

In particular, it follows that the output of P_m in the HYBRID setting is computationally indistinguishable to its output in the REAL setting. That is, for every $\overline{x}, w \in \{0,1\}^*$ we have

$$\left\{\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{x}, w)\,|_{P_m}\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n, \overline{x}, w)\,|_{P_m}\right\}_{n\in\mathsf{N}} \tag{2}$$

Of course, a special case of Eq. (2) is for the case that $\overline{x} = \overline{z}$ and $w = \lambda$ as defined above for Eq. (1).

We are now ready to show how to construct an ideal-model adversary/simulator $\mathcal{S}$ for the UC setting with $\mathcal{Z}$ and $\mathcal{A}$, from the hybrid-model adversary $\mathcal{S}_\pi$ for π and $\mathcal{A}_\pi$. $\mathcal{S}$ internally invokes $\mathcal{S}_\pi$ and emulates a hybrid execution of π for $\mathcal{S}_\pi$. Notice that $\mathcal{S}$ interacts with $P_1, \ldots, P_m$ and an external environment $\mathcal{Z}$; whereas $\mathcal{S}_\pi$ just interacts with parties $P_1, \ldots, P_m$. Furthermore, $\mathcal{S}$

interacts in an ideal execution with $\mathcal{F}$, whereas $\mathcal{S}_\pi$ interacts in a hybrid execution of $\pi^{\mathcal{F}}$. As we have already seen, the distributions over the messages sent are identical. Therefore all $\mathcal{S}$ needs to do is to "redirect" messages appropriately (e.g., a message of the form $(\mathcal{A}, \mathcal{Z}, \alpha)$ sent by $\mathcal{S}_\pi$ to P_m needs to be sent by $\mathcal{S}$ to $\mathcal{Z}$ directly). In general, $\mathcal{S}$ works as follows:

- *Dealing with* $(\mathcal{A}, \mathcal{Z}, \alpha)$ *and* $(\mathcal{Z}, \mathcal{A}, \beta)$ *messages:* Whenever $\mathcal{S}_\pi$ instructs P_1 to send a message of the form $(\mathcal{A}, \mathcal{Z}, \alpha)$ to P_m, simulator $\mathcal{S}$ externally sends α to $\mathcal{Z}$. Likewise, whenever $\mathcal{S}$ receives a message β from $\mathcal{Z}$, it internally hands P_1 the message $(\mathcal{Z}, \mathcal{A}, \beta)$ as if sent from P_m.

- *Dealing with* (input, x_i) *and* $(\mathsf{output}, P_i, y_i)$ *messages:* When $\mathcal{Z}$ writes to the input tape of an uncorrupted party P_i, simulator $\mathcal{S}$ emulates P_m sending an input message to P_i (this is a message between two uncorrupted parties and so due to the fact that the channels are ideally secure $\mathcal{S}_\pi$ sees nothing but the fact that a message was sent).[6] When an output is delivered from $\mathcal{F}$ to a party P_i (see below under "delivery of messages" as to when this occurs), simulator $\mathcal{S}$ emulates P_i sending the appropriate output message to P_m (not that this message is not necessarily delivered by $\mathcal{S}_\pi$ immediately).

  When $\mathcal{Z}$ writes a value x_i on the input tape of a corrupted party P_i, simulator $\mathcal{S}$ emulates P_m sending the message (input, x_i) to P_i. Likewise, when $\mathcal{S}_\pi$ instructs a corrupted party P_i to send $(\mathsf{output}, P_i, y_i)$ to P_m, simulator $\mathcal{S}$ instructs the corrupted party P_i to write y_i on its output tape (to be read by $\mathcal{Z}$). Note that since P_i is corrupted, $\mathcal{S}$ obtains all of its values. Therefore, $\mathcal{S}$ knows x_i and y_i, as required for carrying out the above emulation.

  Before proceeding, note that the above covers all types of messages in π except for the messages sent to $\mathcal{F}$.

- *Dealing with messages sent by* $\mathcal{S}_\pi$*:* Whenever $\mathcal{S}_\pi$ instructs a corrupted party P_i to send a message α to $\mathcal{F}$, simulator $\mathcal{S}$ instructs P_i to send α to $\mathcal{F}$ in the ideal execution (this involves having P_i write α to its outgoing communication tape for $\mathcal{F}$).

- *Delivery of messages:* Simulator $\mathcal{S}$ delivers a message from a party P_i to the ideal functionality $\mathcal{F}$ whenever $\mathcal{S}_\pi$ would deliver the corresponding message in the hybrid execution of π. In contrast, $\mathcal{S}$ delivers a message from $\mathcal{F}$ to P_i only when $\mathcal{S}_\pi$ would deliver the corresponding output message from P_i to P_m (note that this may be after $\mathcal{S}_\pi$ would deliver the message from $\mathcal{F}$ to P_i in π; the reason for this is explained below).

This completes the description of $\mathcal{S}$.

The proof is concluded by showing that $\mathcal{S}$'s emulation for $\mathcal{S}_\pi$ is perfect. First, recall that in an execution of π, party P_m runs the code of $\mathcal{Z}$. Therefore, the messages of the form $(\mathcal{Z}, \mathcal{A}, \alpha)$ sent from P_m to P_1 are distributed identically in the emulation by $\mathcal{S}$ and in a hybrid-model execution of π. Furthermore, the inputs that all parties send to $\mathcal{F}$ are also identically distributed in both scenarios. This is because the honest parties receive their inputs from P_m in π (who runs $\mathcal{Z}$) and from $\mathcal{Z}$ itself in the ideal execution with $\mathcal{S}$. Likewise, the inputs sent to $\mathcal{F}$ by the corrupted parties in π are those instructed by $\mathcal{S}_\pi$, and $\mathcal{S}$ sends these same inputs. Finally, we claim that the delivery of messages is the same in both of the executions. This is clearly the case for all of the messages between the parties and the ideal functionality $\mathcal{F}$. However, an issue arises with respect to the input and output messages. In the UC setting, $\mathcal{Z}$ writes inputs directly to the parties' input tapes and likewise directly reads their outputs. Thus, the adversary has no control over the scheduling

[6]This assumes that $\mathcal{S}$ knows when $\mathcal{Z}$ writes to a party's input tape. This holds because in the UC model, when $\mathcal{Z}$ writes to a party's input tape, that party is activated next. Then, in a UC ideal execution, that party immediately writes a message on its outgoing communication tape for $\mathcal{F}$. Since $\mathcal{S}$ is responsible for the delivery of messages, it knows whenever a party writes to its outgoing communication tape.

of these messages. In contrast, in our protocol π these input/output messages are sent by P_m and scheduled by $\mathcal{S}_\pi$. Since we have no control over how $\mathcal{S}_\pi$ works (we are just guaranteed its existence by the assumption that ρ is secure), we don't know when it delivers these messages. In order to see that this is not a problem, first note that in the UC setting after receiving an input from $\mathcal{Z}$ an honest party immediately sends the input to the ideal functionality $\mathcal{F}$. This communication between an honest party and $\mathcal{F}$ *can be* scheduled by $\mathcal{S}$ who delivers it whenever $\mathcal{S}_\pi$ delivers the analogous message in π. Thus, it doesn't matter when $\mathcal{Z}$ writes the input value; what is important is when this is received by $\mathcal{F}$ and this is the same in both settings. Regarding the output values, as above, $\mathcal{Z}$ can read these outputs as soon as the honest parties receive them from $\mathcal{F}$. However, in π, party P_m can only read them after $\mathcal{S}_\pi$ delivers the output messages. If the delivery of these messages is scheduled differently then this can make the view of the real $\mathcal{Z}$ in the UC setting different from the view of the environment $\mathcal{Z}$ that is internally emulated by P_m. However, notice that by the specification of $\mathcal{S}$, it only delivers the outputs from $\mathcal{F}$ to the honest parties when $\mathcal{S}_\pi$ delivers the output messages from the parties to $\mathcal{Z}$. Without loss of generality, we assume that $\mathcal{Z}$ reads all the outputs tapes every time it is activated. This means that $\mathcal{Z}$ obtains the output the first time it is activated after an honest party receives its output from $\mathcal{F}$. Thus, once again, these messages are effectively delivered at the same time (in the UC model $\mathcal{S}$ delivers the output from $\mathcal{F}$ which is then immediately read by $\mathcal{Z}$ when $\mathcal{S}_\pi$ delivers the corresponding output message from the honest party to P_m who hands it to the internally emulated $\mathcal{Z}$). This all implies that $\mathcal{Z}$'s view in the ideal execution of $\mathcal{F}$ with $\mathcal{S}$ is identical to P_m's view in the $\mathcal{F}$-hybrid execution of π with $\mathcal{S}_\pi$. Thus, for every $z \in \{0,1\}^*$ we have

$$\left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n,\overline{z},\lambda)\,|_{P_m}\right\}_{n\in\mathsf{N}} \equiv \left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}} \tag{3}$$

where $\overline{z}$ is as defined in Eq. (1). The theorem is obtained by combining Equations (1) to (3). ■

Remark – static adversaries. The proof of Theorem 3 is sensitive to the specific formulation of static adversaries used. That is, we rely heavily on the fact that the set of corrupted parties is fixed for a given adversary and that the set cannot depend on the input or random coin tosses. This is due to the fact that the protocol defines the instructions of P_1 and P_m and the emulation only works if P_1 is always corrupted and P_m is always honest. Despite this, we remark that the impossibility results of [12] all have this property (i.e., use an adversary for which the corrupted party is a priori fixed). In addition, in the next section we will present a proof for the case of adaptive adversaries which clearly covers the case of static adversaries. The only difference is that we need to assume that ρ is secure for *arbitrary sets of parties* rather than just for a single set of parties as we assume here.

Remark – specialized simulation. According to Definition 1 (security under concurrent general composition), a different simulator $\mathcal{S}_\pi$ may be provided for every protocol π. Therefore, in the proof of Theorem 3, it is crucial that π is fixed before $\mathcal{S}_\pi$ and thus $\mathcal{S}$ are obtained. Since π emulates the environment $\mathcal{Z}$, this means that $\mathcal{Z}$ must be fixed before the UC simulator $\mathcal{S}$ is obtained. This explains why we can only prove that *specialized-simulator* UC is implied, and not the original UC definition itself.

Discussion – UC stringencies. The proof of Theorem 3 shows why the two stringencies of the UC definition, as discussed in the Introduction, are essential. These two stringencies are (1) that $\mathcal{S}$ has only black-box access to $\mathcal{Z}$, and (2) that $\mathcal{S}$ cannot rewind $\mathcal{Z}$. Now, in the protocol π that calls ρ, an honest party $P_{\mathcal{Z}}$ runs the code of $\mathcal{Z}$. Intuitively, this means that the environment $\mathcal{Z}$ in the UC definition models the code that is run by honest parties outside of the secure protocol ρ. The key point is that the ideal-model adversary/simulator clearly cannot be given control over the honest parties in the network, or all meaning of security is lost. In other words, in order for security to be obtained, the ideal-model adversary can only have (1) black-box access to honest parties, and (2) cannot rewind these parties. Since the environment $\mathcal{Z}$ models these parties, the UC simulator $\mathcal{S}$ must also be given only black-box access to $\mathcal{Z}$, without the possibility of rewinding.

3.2 A More General Theorem

In this section, we prove a more general theorem, stating that security under 1-bounded concurrent general composition implies specialized-simulator UC security. Our proof here is for adaptive adversaries and does not assume ideal secure channels or even authenticated channels. Of course, it is assumed that the communication between the parties and the ideal functionality *is* authenticated and private; however, no such assumption is made with respect to standard communication between the parties. In addition, we prove this implication when the parties have access to an ideal functionality $\mathcal{G}$ in both the setting of concurrent general composition and UC security. The importance of this is that it shows that even when some setup assumption is used (e.g., a common reference string that is given by $\mathcal{G}$), then UC security is still minimal. We stress that $\mathcal{G}$ can be used to model setup as above, and also authenticated channels (yielding the standard real model where the parties communicate via authenticate, but not necessarily private, channels). Of course, setting $\mathcal{G}$ to be a functionality that does nothing yields a completely plain model. Thus, this formalization of the theorem is very general and demonstrates our result for a variety of models. As for Theorem 3 we consider the case that all parties run in time that is polynomial in the security parameter; the general case of arbitrary robust complexity classes is considered in Section 3.3 below.

Summing up, there are three differences between the theorem that we prove here and Theorem 3 proved in Section 3.1:

1. We prove that concurrent general composition implies specialized-simulator UC security in the $\mathcal{G}$-hybrid model, for any ideal functionality $\mathcal{G}$.

2. We do not assume ideally secure or authenticated channels.

3. We consider adaptive adversaries and arbitrary sets of parties. This immediately implies security for static adversaries and arbitrary sets of parties (simply consider adversaries that corrupt only at the onset). Using the ideas from the proof of Theorem 3 the proof can be modified to hold for static adversaries and fixed sets of parties, as long as at least one party is corrupted.

Before proceeding to the proof, we remark that we use the notion of one-time information-theoretic message authentication codes. Loosely speaking, a message authentication code (MAC) is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ where $\mathsf{Gen}(1^n)$ outputs a key k that can be used to authenticate messages of length n, Mac receives a key k and some message $x \in \{0,1\}^n$ and outputs a MAC-tag t, and Vrfy receives k, m and t and outputs 1 or 0. The correctness requirement of a MAC is that for every n, every k in the range of $\mathsf{Gen}(1^n)$ and every $x \in \{0,1\}^n$ it holds that $\mathsf{Vrfy}_k(x, \mathsf{Mac}_k(x)) = 1$. The security requirement of *one-time information-theoretic MACs* is that every (not necessarily efficient) adversary $\mathcal{A}$ succeeds in the following experiment with at most negligible probability:

1. Upon input 1^n, adversary $\mathcal{A}$ outputs a message $x \in \{0,1\}^n$

2. A key $k \leftarrow \mathsf{Gen}(1^n)$ is chosen, and a tag $t = \mathsf{Mac}_k(x)$ is computed and handed to $\mathcal{A}$

3. $\mathcal{A}$ outputs a pair (x', t')

4. $\mathcal{A}$ is said to have succeeded if $x' \neq x$ and $\mathsf{Vrfy}_k(x', t') = 1$.

We remark that one-time information-theoretic MACs can be constructed without any computational assumptions (e.g., it suffices to use any pairwise-independent hash function).

We now proceed to the proof. (Although much of the proof is the same as for Theorem 3, due to the numerous differences we present a full proof here. However, we assume that the reader has read the proof of Theorem 3 and so we do not repeat intuitive explanations.)

Theorem 4 *Let ρ be a protocol and let $\mathcal{F}$ and $\mathcal{G}$ be functionalities. If ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under 1-bounded concurrent general composition with arbitrary sets of parties and adaptive adversaries, then ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under specialized-simulator UC with adaptive adversaries.*

Proof: Let ρ be a protocol for parties $P_1, \ldots, P_m$, and let $\mathcal{F}$ and $\mathcal{G}$ be functionalities such that ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under 1-bounded concurrent general composition with arbitrary sets of parties and adaptive adversaries. We wish to show that ρ securely computes $\mathcal{F}$ under specialized-simulator UC. Fix an adaptive adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. We define a protocol π, involving parties $P_{\mathcal{Z}}, P_{\text{adv}}$ and $P_1, \ldots, P_m$. Loosely speaking, party $P_{\mathcal{Z}}$ will play the role of the UC environment $\mathcal{Z}$, and party P_{adv} – when corrupted – will play the role of the UC (real-model or ideal-model) adversary. This is a major point that is different to the proof of Theorem 3; since we are considering adaptive adversaries here, none of the parties $P_1, \ldots, P_m$ can be guaranteed to be corrupted or uncorrupted. We therefore introduce new parties who play this role (below we show how $P_{\mathcal{Z}}$ can be guaranteed to be honest). Another problem that we have to deal with is that there are no secure channels here, but $P_{\mathcal{Z}}$ must be able to communicate secretly and reliably with the parties $P_1, \ldots, P_m$ in order to hand them inputs and receive back their outputs. We achieve this by using one-time pads (for secrecy) and information-theoretic MACs (for integrity) in the specification of protocol π. For the sake of simplicity, in the description below we assume that $\mathcal{Z}$ writes at most n bits of input on each party's input tape and that each party generates at most n bits of output. This assumption is of course incorrect for reactive functionalities. However, it is easy to extend the proof below by simply using long enough random pads and keys for information-theoretic MACs that can authenticate messages of the required length. (Recall that $\mathcal{Z}$ is polynomial-time and fixed before π is constructed. Therefore, there exists an a priori polynomial bound on the length of all communication between $\mathcal{Z}$ and the parties.) Protocol π for the $\mathcal{F}$-hybrid model is defined as follows:

1. Inputs: Party $P_{\mathcal{Z}}$ receives a value z for input, $2m$ strings $r_1^a, r_1^b, \ldots, r_m^a, r_m^b$ that are each uniformly distributed in $\{0,1\}^n$, and $2m$ keys $k_1^a, k_1^b, \ldots, k_m^a, k_m^b$ chosen independently according to $\mathsf{Gen}(1^n)$. For every i, party P_i receives the two pairs (r_i^a, r_i^b) and (k_i^a, k_i^b). (Thus, $P_{\mathcal{Z}}$ shares 2 one-time pads and 2 information-theoretic MAC keys with each P_i.[7])

2. Instructions for $P_{\mathcal{Z}}$: Party $P_{\mathcal{Z}}$, upon input z, internally invokes the environment $\mathcal{Z}$ with input z. When $\mathcal{Z}$ wishes to send a message to the adversary that it interacts with, party $P_{\mathcal{Z}}$

[7]Since indistinguishability of the REAL and HYBRID executions is required for all inputs, it implies indistinguishability also when some of the inputs are uniformly distributed or chosen according to Gen. Thus, we can assume that the one-time pads are indeed random and the MAC keys chosen according to Gen.

sends the message to P_{adv}; likewise, when $P_{\mathcal{Z}}$ receives a message from P_{adv} then $P_{\mathcal{Z}}$ internally hands this to $\mathcal{Z}$ as if it is from the adversary that $\mathcal{Z}$ interacts with. When $\mathcal{Z}$ writes a value x_i intended for the input tape of a party P_i, then party $P_{\mathcal{Z}}$ computes $\alpha_i = x_i \oplus r_i^a$ and $t_i^a = \mathsf{Mac}_{k_i^a}(\alpha_i)$ and sends the tuple $(\mathsf{input}, \alpha_i, t_i^a)$ to P_i. During the execution, $P_{\mathcal{Z}}$ receives tuples $(\mathsf{output}, P_i, \beta_i, t_i^b)$ from the parties $P_1, \ldots, P_m$ (see step (3) below). Upon receiving such a message, $P_{\mathcal{Z}}$ checks that $\mathsf{Vrfy}_{k_i^b}(\beta_i, t_i^b) = 1$; if yes, it stores the tuple $(\mathsf{output}, P_i, y_i)$ where $y_i = \beta_i \oplus r_i^b$; if no, it outputs $\perp$ and halts immediately. When $\mathcal{Z}$ wishes to read the output tape of a party P_i, party $P_{\mathcal{Z}}$ checks that has stored an output tuple for P_i; if yes, it internally hands $\mathcal{Z}$ the value y_i in the tuple; if no, it hands $\mathcal{Z}$ a blank message λ (to be interpreted as if P_i has not yet written to its output tape).

3. Instructions for P_{adv}: Party P_{adv} has no instructions (this may seem strange, but we will always consider a scenario that P_{adv} is corrupt and so anyway follows the adversary's instructions).

4. Instructions for P_i (for every $1 \leq i \leq m$): When party P_i receives a message $(\mathsf{input}, \alpha_i, t_i^a)$ from $P_{\mathcal{Z}}$, it first checks that $\mathsf{Vrfy}_{k_i^a}(\alpha_i, t_i^a) = 1$ and if yes, it computes $x_i = \alpha_i \oplus r_i^a$ and sends x_i to the ideal functionality $\mathcal{F}$. Otherwise, it outputs $\perp$ and halts. When P_i receives its output y_i from $\mathcal{F}$, it computes $\beta_i = y_i \oplus r_i^b$ and $t_i^b = \mathsf{Mac}_{k_i^b}(\beta_i)$ and sends the tuple $(\mathsf{output}, P_i, \beta_i, t_i^b)$ to $P_{\mathcal{Z}}$. (Recall that these messages go through the adversary who controls delivery.)

5. Output: Party $P_{\mathcal{Z}}$ outputs whatever $\mathcal{Z}$ outputs. All other parties output the empty string λ.

Recall that in the composed protocol π^ρ, the parties $P_1, \ldots, P_m$ run an execution of ρ in the $\mathcal{G}$-hybrid model instead of sending their inputs to the functionality $\mathcal{F}$.

Given the protocol π, we define a probabilistic polynomial-time adversary $\mathcal{A}_\pi$ who attacks the composed, $\mathcal{G}$-hybrid-model protocol π^ρ. The idea of $\mathcal{A}_\pi$ is to set up a scenario which perfectly emulates the UC real-model setting with adversary $\mathcal{A}$ and environment $\mathcal{Z}$. $\mathcal{A}_\pi$ corrupts party P_{adv} and internally runs the code of the UC adversary $\mathcal{A}$ (as fixed above), using P_{adv} to communicate with $P_{\mathcal{Z}}$. That is, when the internal $\mathcal{A}$ wishes to send a message to $\mathcal{Z}$, adversary $\mathcal{A}_\pi$ instructs P_{adv} to send that message to $P_{\mathcal{Z}}$. Likewise, when P_{adv} receives a message from $P_{\mathcal{Z}}$, adversary $\mathcal{A}_\pi$ passes the message to $\mathcal{A}$ as if $\mathcal{A}$ received it from $\mathcal{Z}$.[8] Whenever $\mathcal{A}$ wishes to corrupt a party P_i, adversary $\mathcal{A}_\pi$ corrupts the party P_i and provides $\mathcal{A}$ with the internal state that it expects to receive (i.e., $\mathcal{A}_\pi$ provides $\mathcal{A}$ with the internal state of P_i's ITM for running ρ only). $\mathcal{A}_\pi$ also instructs P_{adv} to send $P_{\mathcal{Z}}$ the information that $\mathcal{Z}$ would receive upon such a corruption. Regarding the execution of ρ, whenever $\mathcal{A}$ instructs a corrupted party P_i to send a message to some party P_j, adversary $\mathcal{A}_\pi$ instructs P_i to send the same message. In addition, $\mathcal{A}_\pi$ delivers messages between the parties whenever $\mathcal{A}$ would deliver them. Likewise, $\mathcal{A}_\pi$ delivers messages between the parties $P_1, \ldots, P_m$ and the functionality $\mathcal{G}$ whenever $\mathcal{A}$ would deliver such messages. Finally, messages between $P_{\mathcal{Z}}$ and the parties $P_1, \ldots, P_m$ are all delivered by $\mathcal{A}_\pi$ *immediately* and *unmodified*. This completes the definition of $\mathcal{A}_\pi$.

We now claim that for every UC adversary $\mathcal{A}$ and UC environment $\mathcal{Z}$, the output of $P_{\mathcal{Z}}$ from an execution of π^ρ with $\mathcal{A}_\pi$ in the $\mathcal{G}$-hybrid model is distributed exactly like the output of $\mathcal{Z}$ after an execution of ρ with $\mathcal{A}$ in the UC $\mathcal{G}$-hybrid model. That is, for a given value z, define the following random variable for the parties' inputs:

$$\overline{X}_n(z) = ((z, r_1^a, k_1^a, r_1^b, k_1^b, \ldots, r_m^a, k_m^a, r_m^b, k_m^b), \lambda, (r_1^a, k_1^a, r_1^b, k_1^b), \ldots, (r_m^a, k_m^a, r_m^b, k_m^b))$$

[8] P_{adv} is used in π because in the setting of general composition, the adversary can only communicate with honest parties, like $P_{\mathcal{Z}}$, by instructing a corrupted party to send some message. In contrast, in the UC framework, the adversary can communicate with $\mathcal{Z}$ even if no parties are corrupted. Thus, in the general composition setting, P_{adv} is always corrupted, allowing the adversary to send messages to $P_{\mathcal{Z}}$ even if none of $P_1, \ldots, P_m$ are corrupted.

where all r_i^a and r_i^b's are uniformly distributed in $\{0,1\}^n$ and all k_i^a and k_i^b are all chosen according to $\mathsf{Gen}(1^n)$. In other words, consider the input case that $P_{\mathcal{Z}}$ receives $(z, r_1^a, k_1^a, r_1^b, k_1^b, \ldots, r_m^a, k_m^a, r_m^b, k_m^b)$ for input, P_{adv} receives the empty input λ, and every party P_i receives $(r_i^a, k_i^a, r_i^b, k_i^b)$, where all r_i^a and r_i^b's are random and k_i^a, k_i^b are MAC-keys chosen according to $\mathsf{Gen}(1^n)$. Then, we claim that for all $z \in \{0,1\}^*$,

$$\left\{\text{UC-HYBRID}^{\mathcal{G}}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}} \equiv \left\{\text{HYBRID}^{\mathcal{G}}_{\pi^\rho,\mathcal{A}_\pi}(n,\overline{X}_n(z),\lambda)\,|_{P_{\mathcal{Z}}}\right\}_{n\in\mathsf{N}} \tag{4}$$

where $\text{HYBRID}^{\mathcal{G}}_{\pi^\rho,\mathcal{A}_\pi}(n,\overline{x},z)\,|_P$ denotes the output of the party P from the execution of π^ρ in the $\mathcal{G}$-hybrid model. Eq. (4) follows from the following observations. First, $P_{\mathcal{Z}}$ perfectly emulates the environment $\mathcal{Z}$. Therefore, the inputs of the parties $P_1,\ldots,P_m$ to ρ in the real execution of π^ρ are identically distributed to the inputs of the parties to ρ in the UC setting. Note that $P_{\mathcal{Z}}$ provides these inputs in π^ρ whereas $\mathcal{Z}$ provides these inputs in the UC setting. However, $\mathcal{A}_\pi$ delivers these input messages immediately and unmodified. Therefore there is no difference. Second, the view of $\mathcal{A}$ in the emulation by $\mathcal{A}_\pi$ is identical to its view in the UC setting. Therefore, the behavior of the corrupted parties is identical in both settings. Finally, the honest parties in π^ρ behave exactly the same as in an execution of ρ (note that this includes when they are activated because $\mathcal{A}_\pi$ delivers the inputs immediately). Putting this together, we have that the view of $\mathcal{Z}$ in the emulation by $P_{\mathcal{Z}}$ in π^ρ is identical to its view in the UC setting. We conclude that the distribution over $P_{\mathcal{Z}}$'s output is identical to the distribution over $\mathcal{Z}$'s output.

Having established the connection between a UC-HYBRID$^{\mathcal{G}}$ execution of ρ and a HYBRID$^{\mathcal{G}}$ execution of π^ρ, we proceed to show how a UC-type simulator is obtained for ρ. First, by the security of ρ in the $\mathcal{G}$-hybrid model under 1-bounded concurrent general composition and arbitrary sets of parties, we have that there exists a hybrid-model simulator $\mathcal{S}_\pi$ such that for every vector of inputs $\overline{x} \in \{0,1\}^*$ and every auxiliary-input $w \in \{0,1\}^*$ for $\mathcal{A}_\pi$,

$$\left\{\text{HYBRID}^{\mathcal{G}}_{\pi^\rho,\mathcal{A}_\pi}(n,\overline{x},w)\right\}_{n\in\mathsf{N}} \stackrel{c}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n,\overline{x},w)\right\}_{n\in\mathsf{N}}$$

In particular, it follows that the output of $P_{\mathcal{Z}}$ in an execution of π in the $\mathcal{F}$-hybrid model is computationally indistinguishable from its output in an execution of π^ρ in the $\mathcal{G}$-hybrid model. Furthermore, since it holds for all input vectors $\overline{x}$ and auxiliary inputs z for $\mathcal{A}_\pi$, it holds when $\overline{x}$ is chosen according to $\overline{X}_n(z)$ as defined for Eq. (4) and when $w = \lambda$. Thus, for every $z \in \{0,1\}^*$,

$$\left\{\text{HYBRID}^{\mathcal{G}}_{\pi^\rho,\mathcal{A}_\pi}(n,\overline{X}_n(z),\lambda)\,|_{P_{\mathcal{Z}}}\right\}_{n\in\mathsf{N}} \stackrel{c}{\equiv} \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n,\overline{X}_n(z),\lambda)\,|_{P_{\mathcal{Z}}}\right\}_{n\in\mathsf{N}} \tag{5}$$

We are now ready to show how to construct an ideal-model adversary/simulator $\mathcal{S}$ for the UC setting with $\mathcal{Z}$ and $\mathcal{A}$, from the hybrid-model adversary $\mathcal{S}_\pi$ for π and $\mathcal{A}_\pi$. $\mathcal{S}$ internally invokes $\mathcal{S}_\pi$ and emulates a hybrid execution of π for $\mathcal{S}_\pi$. Notice that $\mathcal{S}$ interacts with $P_1,\ldots,P_m$ and an external environment $\mathcal{Z}$; whereas $\mathcal{S}_\pi$ interacts with parties $P_{\mathcal{Z}}, P_{\text{adv}}, P_1,\ldots,P_m$. Furthermore, $\mathcal{S}$ interacts in a UC-ideal execution with $\mathcal{F}$, whereas $\mathcal{S}_\pi$ interacts in a hybrid execution of $\pi^{\mathcal{F}}$. Therefore, $\mathcal{S}$ must emulate for $\mathcal{S}_\pi$ the additional parties $P_{\mathcal{Z}}$ and P_{adv} and the additional messages belonging to π.

- *Emulation of $P_{\mathcal{Z}}$:* The emulation of $P_{\mathcal{Z}}$ works by redirecting messages intended for $P_{\mathcal{Z}}$ to the external environment $\mathcal{Z}$ and vice versa. That is, whenever $\mathcal{S}_\pi$ instructs P_{adv} to send a message to $P_{\mathcal{Z}}$, simulator $\mathcal{S}$ externally sends this message to $\mathcal{Z}$; likewise, all messages that $\mathcal{S}$ receives from $\mathcal{Z}$ are internally handed to P_{adv} as if sent from $P_{\mathcal{Z}}$. Note that since $\mathcal{A}_\pi$ does not corrupt

$P_{\mathcal{Z}}$, simulator $\mathcal{S}_\pi$ cannot corrupt $P_{\mathcal{Z}}$.[9] (This holds because the set of corrupted parties can be discerned from the global output. Therefore, if $\mathcal{S}_\pi$ were to corrupt a different set of parties to $\mathcal{A}_\pi$, it would immediately be possible to distinguish the $\textsc{hybrid}^{\mathcal{G}}_{\pi^\rho}$ and $\textsc{hybrid}^{\mathcal{F}}_{\pi}$ executions.) Thus, $\mathcal{S}$ does not need to deal with the case that $P_{\mathcal{Z}}$ is corrupted by $\mathcal{S}_\pi$.

- *Emulation of P_{adv}:* First, note that until the point that P_{adv} is corrupted, it does nothing. Thus, the only emulation required is when $\mathcal{S}_\pi$ corrupts P_{adv}. In this case, all $\mathcal{S}$ needs to do is to hand $\mathcal{S}_\pi$ the series of messages that P_{adv} would have received so far. However, by the construction of π, party P_{adv} receives messages from $P_{\mathcal{Z}}$ only and these messages are exactly the messages that $\mathcal{S}$ receives from $\mathcal{Z}$. Therefore, $\mathcal{S}$ just hands $\mathcal{S}_\pi$ the messages that $\mathcal{S}$ received from $\mathcal{Z}$ so far. The emulation of P_{adv} following its corruption relates only to its communication with $P_{\mathcal{Z}}$; this has already been described above under the emulation of $P_{\mathcal{Z}}$.

- *Emulation of π-messages and parties $P_1, \ldots, P_m$:* $\mathcal{S}$ internally forwards all messages between $\mathcal{S}_\pi$ and the simulated parties $P_{\mathcal{Z}}$ and P_{adv}. Next, when $\mathcal{Z}$ writes a value on the input tape of an uncorrupted party P_i, simulator $\mathcal{S}$ emulates $P_{\mathcal{Z}}$ sending $(\mathsf{input}, \alpha_i, t_i^a)$ to P_i, for $\alpha_i \in_R \{0,1\}^n$ and $t_i^a = \mathsf{Mac}_{k_i^a}(\alpha_i)$, where k_i^a is chosen according to $\mathsf{Gen}(1^n)$. Likewise, when $\mathcal{Z}$ reads the output tape of an uncorrupted party P_i, simulator $\mathcal{S}$ emulates P_i sending $(\mathsf{output}, P_i, \beta_i, t_i^b)$ to $P_{\mathcal{Z}}$, for $\beta_i \in_R \{0,1\}^n$ and $t_i^a = \mathsf{Mac}_{k_i^a}(\alpha_i)$, where k_i^a is chosen according to $\mathsf{Gen}(1^n)$. (This assumes that $\mathcal{S}$ knows when $\mathcal{Z}$ writes to a party's input tape and reads from its output tape; see Footnote 6 as to why this is the case.)

  If $\mathcal{S}_\pi$ corrupts party P_i before $\mathcal{Z}$ writes a value on its input tape, then $\mathcal{S}$ chooses random (r_i^a, r_i^b) and $k_i^a, k_i^b \leftarrow \mathsf{Gen}(1^n)$ and hands the values to $\mathcal{S}_\pi$ as P_i's input. Then, when $\mathcal{Z}$ writes a value x_i on P_i's input tape, $\mathcal{S}$ emulates $P_{\mathcal{Z}}$ sending $(\mathsf{input}, x_i \oplus r_i^a, t_i^a)$ to P_i, where $t_i^a = \mathsf{Mac}_{k_i^a}(x_i \oplus r_i^a)$. Likewise, when P_i receives its output y_i from $\mathcal{F}$, simulator $\mathcal{S}$ emulates P_i sending $(\mathsf{output}, P_i, y_i \oplus r_i^b, t_i^b)$ to $P_{\mathcal{Z}}$, where $t_i^a = \mathsf{Mac}_{k_i^b}(y_i \oplus r_i^b)$. Note that since P_i is corrupted, $\mathcal{S}$ obtains all of its values. Therefore, $\mathcal{S}$ knows x_i and y_i, as required for carrying out the above emulation.

  If $\mathcal{S}_\pi$ corrupts a party P_i after $\mathcal{Z}$ writes a value on its input tape (and before it receives its output), then $\mathcal{S}$ has already simulated $P_{\mathcal{Z}}$ sending $(\mathsf{input}, \alpha_i, t_i^a)$ to P_i (see above). Therefore, $\mathcal{S}$ corrupts P_i and obtains the input value x_i that $\mathcal{Z}$ wrote on P_i's input tape. Then, $\mathcal{S}$ sets $r_i^a = \alpha_i \oplus x_i$, chooses a random $r_i^b \in_R \{0,1\}^n$, computes $k_i^b \leftarrow \mathsf{Gen}(1^n)$, and hands $\mathcal{S}_\pi$ the pairs $(r_i^a, r_i^b), (k_i^a, k_i^b)$ as P_i's input. The emulation from this point on is the same as above.

  Finally, if $\mathcal{S}_\pi$ corrupts P_i after its input and output values have been sent, then both α_i and β_i have already been fixed, as have k_i^a and k_i^b. Thus, $\mathcal{S}$ corrupts P_i and obtains x_i and y_i. Next, $\mathcal{S}$ defines $r_i^a = \alpha_i \oplus x_i$ and $r_i^b = \beta_i \oplus y_i$. The rest of the emulation is as above.

- *Delivery of messages:* Simulator $\mathcal{S}$ delivers a message from a party P_i to the ideal functionality $\mathcal{F}$ whenever $\mathcal{S}_\pi$ would deliver the corresponding message in the hybrid execution of π. In contrast, $\mathcal{S}$ delivers a message from $\mathcal{F}$ to P_i only when $\mathcal{S}_\pi$ would deliver the corresponding output message from P_i to $P_{\mathcal{Z}}$.

This completes the description of $\mathcal{S}$.

The proof is concluded by showing that $\mathcal{S}$'s emulation for $\mathcal{S}_\pi$ is perfect except with negligible probability. This follows from an identical argument to the proof of Theorem 3 with the following additions. Here, instead of ideally secure channels, messages are encrypted using a one-time pad

[9] It is crucial that $\mathcal{S}_\pi$ not corrupt $P_{\mathcal{Z}}$ because were it to do this, $\mathcal{S}$ would be unable to carry out the simulation. This is because $\mathcal{S}$ cannot corrupt its environment $\mathcal{Z}$ and so cannot provide $\mathcal{S}_\pi$ with the appropriate internal state of $P_{\mathcal{Z}}$.

(that achieves perfect secrecy) and verified using an information-theoretic MAC. Thus, the only possible difference between the two is the negligible probability that $\mathcal{S}_\pi$ can have an honest party or $P_\mathcal{Z}$ accept a value that is different from what was sent (the fact that $\mathcal{S}$ sends uniform values α and β makes no difference whatsoever due to the properties of one-time pad encryption). By the security requirement of an information theoretic MAC, this can happen with at most negligible probability. In addition to the above, we note that the view of $\mathcal{S}_\pi$ due to corruptions is clearly the same. We conclude that for every $z \in \{0,1\}^*$,

$$\left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n, \overline{X}_n(z), \lambda)\,|_{P_\mathcal{Z}}\right\}_{n\in\mathsf{N}} \stackrel{\text{s}}{\equiv} \left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N}} \tag{6}$$

where $\overline{X}_n(z)$ is as defined in Eq. (4). The theorem is obtained by combining Equations (4) to (6). ■

3.3 Theorem 4 for Robust Complexity Classes

Notice that in the proof of Theorem 4, all that is needed is for the parties $P_\mathcal{Z}$ and P_{adv} to internally emulate $\mathcal{Z}$ and $\mathcal{A}$, to encrypt the messages sent between $P_\mathcal{Z}$ and the parties P_i using a one-time pad, and to compute an information-theoretic MAC. The encryptions are linear in the length of the messages sent and the MAC can be quadratic. Thus, the complexity of these parties in the setting of concurrent general composition is at most quadratic in the complexity of the analogous parties in the setting of universal composability. Thus, if $\mathcal{Z}$ and $\mathcal{A}$ are in some robust complexity class $\mathcal{C}$, then $P_\mathcal{Z}$ and P_{adv} are also in $\mathcal{C}$. We therefore obtain the following theorem:

Theorem 5 *Consider variants of Definitions 1 and 2 where all parties belong to a robust complexity class $\mathcal{C}$. Let ρ be a protocol and let $\mathcal{F}$ and $\mathcal{G}$ be functionalities. If ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under 1-bounded concurrent general composition with arbitrary sets of parties and adaptive adversaries, then ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under specialized-simulator UC with adaptive adversaries.*

Noting that the notion of polynomial-time in [9] is trivially a robust complexity class (see below), we have the following corollary:

Corollary 6 *Consider variants of Definitions 1 and 2 where all parties run in polynomial time as defined in [9]. Then, a protocol ρ securely computes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under 1-bounded concurrent general composition with arbitrary sets of parties and adaptive adversaries if and only if ρ UC computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model with adaptive adversaries.*

This corollary follows from the fact that under polynomial time in [9], specialized-simulator UC is equivalent to UC security. Therefore, our proof actually demonstrates that 1-bounded concurrent general composition implies UC security. (The other direction follows from the UC composition theorem.) For the sake of completeness, in Appendix A we describe the definition of polynomial-time of [9] and show that it is a robust complexity class.

3.4 Theorem 4 for the Parallel Setting

In this section, we consider the case of *parallel* general composition. In this case, protocols π and ρ begin at the same time and proceed in a synchronized fashion. That is, the π and ρ messages are delivered together, and in order.

We do not know how to show that parallel general composition implies specialized-simulator universal composition, as defined. Rather, we show the implication for a slightly modified definition, where the modification relates to the process of activations in the UC real and ideal models. Specifically, we limit the interaction between the environment and adversary in the real model to be one message per round of the protocol (no limitation is placed on the ideal model). That is, in each round, the environment sends one message to the adversary and this message must contain all corrupt and message-delivery instructions.[10] Likewise, the adversary replies to the environment with a single message. This is in contrast to the UC definition which allows unlimited interaction between the environment and adversary. We call this definition synchronized UC as the adversary and environment are somewhat synchronized.

We remark that the universal composition theorem of [8] (for concurrent composition) and the impossibility results of [12] also hold in this case. We now show that 1-bounded *parallel* general composition implies specialized-simulator *synchronized* UC. We state the proposition for the case of static adversaries and a single set of parties. However, it also holds for adaptive adversaries and arbitrary sets of parties.

Proposition 7 *Let ρ be a protocol and let $\mathcal{F}$ and $\mathcal{G}$ be functionalities. If ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under* 1*-bounded* parallel *general composition for a single set of parties and static adversaries, then ρ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model under specialized-simulator* synchronized *UC with static adversaries.*

Proof Sketch: This proposition follows from the fact that in synchronized UC, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ exchange a single pair of messages each round. Therefore, a single round of π can emulate the communication between $\mathcal{Z}$ and $\mathcal{A}$. Notice that this would not hold were $\mathcal{Z}$ and $\mathcal{A}$ to exchange multiple messages for each round of π. ■

We note that the impossibility results of [12] hold also for synchronized specialized-simulator UC. Although this was not explicitly stated in [12], it is straightforward to verify that the environment and real-model adversary constructed by [12] in order to prove their results communicate only once each round. Thus, as we will show in Section 4, Proposition 7 implies impossibility for parallel general composition.

4 Impossibility Results for General Composition

An important ramification of Theorems 3 and 4 and Proposition 7, is that known impossibility results for specialized-simulator UC apply to 1-bounded concurrent (and even parallel) general composition. As we will see, this rules out the possibility of obtaining security under this type of composition for large classes of two-party functionalities. We stress that the impossibility results are *unconditional*. That is, they hold without any complexity assumptions.

Impossibility for specialized-simulator UC. The following impossibility results for specialized-simulator UC and static adversaries were shown in [12] (although the focus of [12] was to prove impossibility results for the UC definition, they do explicitly state which of their results also apply when the quantifiers are reversed as in specialized-simulator UC):

1. Let $X \subseteq \{0,1\}^*$ be a domain, let $f : X \to \{0,1\}^*$ be a (deterministic) function and let $\mathcal{F}_f$ be a two-party ideal functionality that receives $x \in X$ from P_1 and sends $f(x)$ to P_2. If f is weakly

[10] The order of activations must be redefined in this case. However, this can be done in a straightforward way.

one-way,[11] then $\mathcal{F}_f$ cannot be securely computed under specialized-simulator UC. We note that the zero-knowledge functionality [8] over (moderately) hard-on-the-average languages is weakly one-way.

2. Let $f : X \times X \rightarrow \{0,1\}^*$ be a (deterministic) function and let $\mathcal{F}_f$ be a two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2 respectively, and hands both parties $f(x_1, x_2)$. If f depends on both parties' inputs, then $\mathcal{F}_f$ cannot be securely computed under specialized-simulator UC. (Formally, a function f depends on both inputs if there does not exist a function $g : X \rightarrow \{0,1\}^*$ such that $g(x) = f(x, x')$ for every x', or $g(x) = f(x', x)$ for every x'. If such a g exists, then the output is determined solely on the basis of one of the parties' inputs.)

3. Let $f : X \times X \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a (deterministic) function and denote $f = (f_1, f_2)$ (where $f_i : X \times X \rightarrow \{0,1\}^*$ for each f_i). Furthermore, let $\mathcal{F}_f$ be a two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2 respectively, and hands $f_1(x_1, x_2)$ to P_1 and $f_2(x_1, x_2)$ to P_2. If f is not completely revealing, then $\mathcal{F}_f$ cannot be securely computed under specialized-simulator UC.[12] See Appendix B for the definition of completely revealing functionalities.

4. Let $f : X \times X \rightarrow \{0,1\}^*$ be a *probabilistic* function and let $\mathcal{F}_f$ be a two-party ideal functionality that receives x_1 and x_2 from P_1 and P_2 respectively, samples a value v from $f(x_1, x_2)$ and hands both parties v. Loosely speaking, we say that such a probabilistic function f is unpredictable for P_2 if there exists an input x_1 for P_1 such that for every input x_2 and every possible output value v, there is at least a non-negligible probability that $f(x_1, x_2)$ does not equal v. (That is, $f(x_1, x_2)$ is a random variable that does not almost always accept a single value v. In such a case, $f(x_1, x_2)$ defines a non-trivial distribution, irrespective of the value x_2 that is input by P_2.) Likewise, f is unpredictable for P_1 if there exists an x_2 such that for every x_1 and every v, with at least non-negligible probability $f(x_1, x_2)$ does not equal v. If is unpredictable for both P_1 and P_2, then $\mathcal{F}_f$ cannot be securely computed under specialized-simulator UC.

We stress that the above impossibility results are not due to fairness issues. In fact, output delivery is not guaranteed at all in the basic framework for UC security, and so fairness is anyway not required. A result of this is that a protocol that generates no output is actually universally composable by definition. Therefore, these impossibility results hold only for non-trivial protocols that generate output. Specifically, it is required that if no parties are corrupted and all messages are delivered by the adversary, then all parties receive their designated output.

Impossibility results for 1-bounded general composition. Let Φ be the set of ideal functionalities described above, that cannot be securely computed under specialized-simulator UC. Applying Theorems 3 and 4 to the results of [12], we obtain the following corollaries:

[11] A function is weakly one-way if there exists a polynomial $p(\cdot)$ such that for every efficient inverting machine M and all sufficiently large n's, $\Pr[M(f(Un)) \in f^{-1}(f(U_n))] < 1 - 1/p(n)$. Actually, for this impossibility result, it suffices that for every efficient machine M there exists a polynomial $p_{\mathrm{M}}(\cdot)$ such that M succeeds with probability at most $1 - 1/p_{\mathrm{M}}(n)$. Furthermore, any efficient distribution over the domain of f can be allowed, and not just the uniform distribution.

[12] We note that in the extended abstract of this paper [25], it was stated that if f_1 or f_2 has an insecure minor, then $\mathcal{F}_f$ cannot be securely computed under specialized-simulator UC. This statement is incorrect (and was also not proven in [12]).

Corollary 8 *Consider an adversarial model with adaptive or static corruptions. Then, the set of two-party functionalities Φ cannot be securely computed under 1-bounded concurrent general composition for arbitrary sets or a single set of parties. (In the case of static corruptions with a single set of parties, this holds as long as at least one party is corrupted.)*

We remark that technically Theorem 3 was only proven assuming ideally secure channels. However, using the one-time pads in the proof of Theorem 4 this assumption can be removed. The next corollary relates to parallel general composition:

Corollary 9 *Consider an adversarial model with adaptive or static corruptions. Then, the set of two-party functionalities Φ cannot be securely computed under 1-bounded parallel general composition for arbitrary sets or a single set of parties. (In the case of static corruptions with a single set of parties, this holds as long as at least one party is corrupted.)*

Corollary 9 follows from Proposition 7 and from the fact that the results of [12] also apply to specialized-simulator synchronized UC. Note that parallel *self* composition is easier to achieve than concurrent composition (see [22, Section 6]). Nevertheless, Corollary 9 demonstrates that in the context of *general* composition, parallel composition is also "hard" to achieve.

Acknowledgements

We would like to thank Ran Canetti, Oded Goldreich and Hugo Krawczyk for many helpful discussions and comments. We also thank the anonymous referees for their helpful comments.

References

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

[2] B. Barak. Constant-Round Coin-Tossing With a Man in the Middle or Realizing the Shared Random String Model. In *43rd FOCS*, pages 345–355, 2002.

[3] B. Barak, R. Canetti, J. Nielsen and R. Pass. Universally Composable Protocols with Relaxed Set-up Assumptions. In *45th FOCS,* pages 186–195, 2004.

[4] B. Barak and A. Sahai. How To Play Almost Any Mental Game Over The Net – Concurrent Composition via Super-Polynomial Simulation. In *46th FOCS*, pages 543–552, 2005.

[5] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[6] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th STOC,* pages 1–10, 1988.

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

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

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Cryptology ePrint Archive*, Report 2000/067. Version updated 2005.

[10] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO'01*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.

[11] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. *SIAM Journal on Computing*, 32(1):1–47, 2003.

[12] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universal Composable Two-Party Computation Without Set-Up Assumptions. *Journal of Cryptology,* 19(2):135-167, 2006.

[13] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Computation. In 34*th STOC*, pages 494–503, 2002.

[14] R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, Springer-Verlag (LNCS 2729), pages 265–281, 2003.

[15] D. Chaum, C. Crepeau and I. Damgard. Multi-party Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.

[16] C. Dwork, M. Naor, O. Reingold and L.J. Stockmeyer. Magic Functions. *Journal of the ACM*, 50(6):852–921, 2003.

[17] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. *Journal of the ACM*, 51(6):851–898, 2004.

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

[19] O. Goldreich. Cryptography and Cryptographic Protocols. In *Distributed Computing,* 16(2):177–199, 2003.

[20] 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.

[21] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.

[22] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. *Journal of Cryptology,* 18(3):247–287, 2005.

[23] Y. Kalai, Y. Lindell and M. Prabhakaran. Concurrent General Composition of Secure Protocols in the Timing Model. In the *37th STOC,* pages 644–653, 2005. To appear in the *Journal of Cryptology.*

[24] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In 35*th STOC*, pages 683–692, 2003.

[25] Y. Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. (Extended abstract of this paper.) In 44*th FOCS*, pages 394–403, 2003.

[26] Y. Lindell. *Composition of Secure Multi-Party Protocols – A Comprehensive Study.* Lecture Notes in Computer Science Vol. 2815, Springer-Verlag, 2003.

[27] Y. Lindell. Lower Bounds for Concurrent Self Composition. In the *1st Theory of Cryptography Conference* (TCC), Springer-Verlag (LNCS 2951), pages 203–222, 2004. To appear in the *Journal of Cryptology.*

[28] T. Malkin, R. Moriarty and N. Yakovenko. Generalized Environmental Security from Number Theoretic Assumptions. In the 3*rd TCC*, Springer-Verlag (LNCS 3876), pages 343–359, 2006.

[29] S. Micali, R. Pass and A, Rosen. Input-Indistinguishable Computation. In 47*th FOCS*, pages 367–378, 2006.

[30] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[31] R. Pass. Simulation in Quasi-Polynomial Time, and Its Application to Protocol Composition. In *Eurocrypt 2003*, Springer-Verlag (LNCS 2656), pages 160–176, 2003.

[32] R. Pass. Bounded-Concurrent Secure Multi-Party Computation with a Dishonest Majority. In 36*th STOC,* pages 232–241, 2004.

[33] R. Pass and A. Rosen Bounded-Concurrent Secure Two-Party Computation in a Constant Number of Rounds. In 44*th FOCS*, 2003.

[34] B. Pfitzmann and M. Waidner. Composition and Integrity Preservation of Secure Reactive Systems. In *7th ACM Conference on Computer and Communication Security*, pages 245–254, 2000.

[35] M. Prabhakaran and A. Sahai. New Notions of Security: Universal Composability Without Trusted Setup. In 36*th STOC,* pages 242–251, 2004.

[36] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989.

[37] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EUROCRYPT'99*, Springer-Verlag (LNCS 1592), pages 415–431, 1999.

[38] A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986.

A The Definition of Polynomial-Time Interactive Machines of [9]

In most of this paper we have defined polynomial-time in the standard way; namely, all parties are a priori bounded to run in at most $p(n)$ steps where p is a polynomial and n is the security parameter. This works in our setting where the set of parties and protocols they run is a priori fixed (and all communication is delivered by the adversary). In more general settings, strange things can happen. For example, consider the case that each interactive machine invokes another in an infinite series; each machine is polynomial but the overall system is unbounded (see [9] for more discussion regarding these issues). We avoid these technical problems because we consider a setting where all parties and protocols are a priori fixed. We remark that this implies that our model is somewhat limited. Indeed, in addition to the above, our model cannot be used to capture the standard notions of security for a number of primitives. For example, the standard notion of security of encryption is such that the adversary can choose messages to be encrypted. Since the bound on the running-time of the adversary can be any polynomial, it follows that the honest parties who encrypt these messages are not bounded by any single polynomial. Despite these limitations, we find that our formulation is simpler and sufficient to capture most of the key issues that arise.

An alternative notion of polynomial-time for interactive Turing machines was proposed in [9]. As we have mentioned in the paper, this definition has the advantage that specialized-simulator UC is equivalent to UC. Thus, under this definition, all of our results hold for the full UC definition. Due to its relevance to our results, we briefly describe this definition here. An interactive Turing machine M runs in polynomial-time if there exists a polynomial $p(\cdot)$ such that at any point in the computation, the total number of steps taken by M until this point of the computation is bounded by $p(\ell_{\text{IN}} - \ell_{\text{OUT}})$ where ℓ_{IN} is the total number of bits written on the input tape of M during the computation and ℓ_{OUT} is the total number of bits that M wrote to the input tapes of other interactive Turing machines.

This definition guarantees that the number of steps run in an overall system of polynomial-time interactive Turing machines is polynomial; see [9] for a proof of this. We also remark that this notion of polynomial clearly forms a robust complexity class (because if $t(\cdot)$ is a polynomial then for every polynomial $p(\cdot)$, so is $p(t(\cdot))$).

B Completely Revealing Functionalities

As we have stated in Section 4, functionalities that are not completely revealing cannot be securely computed under concurrent general composition, by any protocol. We now define the notion of completely revealing functionalities, as found in [12]. Loosely speaking, a functionality is completely revealing for party P_1, if party P_2 can choose an input so that the output of the functionality fully reveals P_1's input (for *all* possible choices of that input). That is, a functionality f is completely revealing for P_1 if there exists an input y for P_2 so that for every x, it is possible to derive x from $f(x, y)$. For example, let us take the maximum function for a given range, say $\{0, \ldots, n\}$. Then, party P_2 can input $y = 0$ and the result is that it will always learn P_1's exact input. In contrast, the less-than function is *not* completely revealing because for any input used by P_2, there will always be uncertainty about P_1's input (unless P_1's input is the smallest or largest in the range).

In this appendix, we will only present the definition of "completely revealing" for functionalities that have finite domains. This significantly simplifies the definition of "completely revealing"; the general definition can be found in [12].

We begin by defining what it means for two inputs to be "equivalent": Let $f : X \times Y \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a two-party functionality and denote $f = (f_1, f_2)$. Let $x_1, x_2 \in X$. We say that x_1 and x_2 are equivalent with respect to f_2 if for every $y \in Y$ it holds that $f_2(x_1, y) = f_2(x_2, y)$. Notice that if x_1 and x_2 are equivalent with respect to f_2, then x_1 can always be used instead of x_2 (at least without affecting P_2's output). We now define completely revealing functionalities:

Definition 10 (completely revealing functionalities over finite domains): *Let $f : X \times Y \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a deterministic two-party functionality such that the domain $X \times Y$ is finite, and denote $f = (f_1, f_2)$. We say that the function f_2* completely reveals P_1's input *if there exists a single input $y \in Y$ for P_2, such that for every two distinct inputs x_1 and x_2 for P_1 that are not equivalent with respect to f_2, it holds that $f_2(x_1, y) \neq f_2(x_2, y)$. Complete revealing for P_2's input is defined* *analogously. We say that a functionality f is* completely revealing *if f_1 completely reveals P_2's input* and *f_2 completely reveals P_1's input.*

If a functionality is completely revealing for P_1, then party P_2 can set its own input to be the "special value" y from the definition, and then P_2 will always obtain the exact input used by P_1. Specifically, given $v = f_2(x, y)$, party P_2 can traverse over all X and find the unique x for which it holds that $f_2(x, y) = v$ (where uniqueness here is modulo equivalent inputs x and x'). It then follows that x must be P_1's input (or at least is equivalent to it). Thus we see that P_1's input is completely revealed by f_2. In contrast, if f_2 is *not* completely revealing for P_1, then there does not exist such an input for P_2 that enables it to completely determine P_1's input. This is because for every y that is input by P_2, there exist two non-equivalent inputs x_1 and x_2 such that $f_2(x_1, y) = f_2(x_2, y)$. Therefore, if P_1's input happens to be x_1 or x_2, it follows that P_2 is unable to determine which of these inputs were used by P_1. Notice that if a functionality is not completely revealing, P_2 may still learn much of P_1's input (or even the exact input "most of the time"). However, there is a *possibility* that P_2 will not fully obtain P_1's input. As it turns out, the existence of this "possibility" suffices for proving impossibility results.

Note that we require that x_1 and x_2 be non-equivalent because otherwise, x_1 and x_2 are really the same input and so, essentially, both x_1 and x_2 are P_1's input. Technically, if we do not require this, then a functionality may not be completely revealing simply due to the fact that no y can have the property that $f_2(x_1, y) \neq f_2(x_2, y)$ when x_1 and x_2 are equivalent. This would therefore not capture the desired intuition.

As we have mentioned above, the "less than" function (otherwise known as Yao's millionaires' problem) is not completely revealing, as long as the range of inputs is larger than 2. This can easily be demonstrated.