General Composition and Universal Composability in Secure Multi-Party Computation[*]

Yehuda Lindell

IBM T.J.Watson Research
19 Skyline Drive, Hawthorne
New York 10532, USA
lindell@us.ibm.com

August 8, 2003

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 is equivalent to a relaxed variant of universal composability (where the only difference relates to the order of quantifiers in the definition). An important corollary of this theorem is that existing impossibility results for universal composability (or actually its relaxed variant) are inherent in any definition achieving security under concurrent general composition. 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 (almost) implies the definition itself. This indicates that the security definition of universal composability is not overly restrictive.

Keywords: Secure multi-party computation, protocol composition, universal composability.

[*]An extended abstract of this paper will appear in the 44*th FOCS*, 2003. This is a preliminary full version.

1 Introduction

This paper considers the composition of protocols for secure multi-party computation.

Secure computation. In the setting of multi-party 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 requirements on a secure multi-party 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* multi-party probabilistic polynomial-time functionality can be securely computed [24, 16, 4, 12]. 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 three different important parameters: the *context* in which the protocol runs, the *participating parties* and the *scheduling*.

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 *self compose* 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 (e.g., [13, 23]).

   (b) General composition: In this type of composition, many different protocols are run together in the network. Furthermore, these protocols may have been designed independently of one another. 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 [20, 5, 6].

   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. Typically, when self composition is considered, this also means that the same party assumes the same role in each of the executions (e.g., in the case of zero knowledge, it is assumed that the same party plays the "prover" in all executions).

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

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).

Typically, in the setting of composition, a protocol must remain secure for *any polynomial number* of executions. However, sometimes we will consider a model where the number of secure protocol executions may be bounded. This will be explicitly stated wherever it is the case.

Universal Composability (UC). Recently, a robust notion of security, called universal composability, was put forward [6]. 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 universally composable framework, 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

[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.

way throughout the computation. A protocol is said to UC realize a given functionality f if for any 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 *concurrent general composition with arbitrary sets of parties* [6]. 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 multi-party functionality [6] (building on [4, 22]). Furthermore, in the *common reference string model*,[2] any functionality can be UC realized, for any number of corrupted parties [10]. On the negative side, it has also been shown that in the plain model (i.e., without a common reference string), it is impossible to obtain UC secure protocols for a large class of *two-party* functionalities [7, 6, 9]. (Loosely speaking, this class includes, for example, functionalities that depend on both parties' inputs and preserve a non-trivial level of privacy. A specific example of a two-party functionality that cannot be UC realized is Yao's classic millionaires problem [24].)

General composition and UC security. In this paper, we study the relationship between general composition and UC security. As shown in [6], UC security implies concurrent general composition. However, it is not known whether or not the reverse also holds. Resolving this question is of importance for the following two reasons:

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 realized 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".

[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.

We note that the question of whether or not concurrent general composition implies UC security was first posed in [15].

Before proceeding, it is worthwhile to consider why one would 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 [9] 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 and that the 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.

The main result. 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. 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.
- 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 ℓ secure protocols are run concurrently with arbitrary other protocols. (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).

The main contribution of this paper is a proof that specialized-simulator UC security is equivalent to concurrent $O(1)$-bounded general composition (where only a constant number of secure protocols are run in the network). That is:

Theorem 1.1 (main theorem – informally stated): *A protocol is secure under concurrent $O(1)$-bounded general composition if and only if it is secure under specialized-simulator UC.*

Observe that Theorem 1.1 refers to a relaxed variant of UC security (specifically, specialized-simulator UC) and a relaxed variant of general composition (specifically, $O(1)$-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".

Corollary 1 – impossibility for concurrent general composition. In order to use Theorem 1.1 to derive impossibility results for concurrent general composition, we first note that many of the impossibility results of [9] for UC security hold with respect to specialized-simulator UC. This is explicitly stated in [9]. Thus, combining Theorem 1.1 and [9], we obtain the following corollary:

Corollary 1.2 (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 concurrent $O(1)$-bounded general composition.*

We note that in Theorem 1.1 what we actually prove is that security under concurrent **1**-bounded general composition (where only a single secure protocol is run concurrently with arbitrary other protocols) implies specialized-simulator UC. Thus, applying the results of [9], we actually obtain impossibility even for concurrent **1**-bounded general composition. This is important because it means that impossibility holds even for a *very restricted* type of general composition. We conclude that there do *not* exist weaker definitions of security that imply concurrent general composition, but do not suffer from the impossibility results of UC security.

We stress that Corollary 1.2 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 *parallel 1-bounded 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 [18, 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 1.1 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 general composition is to be achieved. This means that at least some flavor of the UC definition is *necessary* in any security definition that achieves this level of composition. In particular, the feature of the UC definition where the simulator interacts with the environment in a black-box manner, and without rewinding, is essential. (Recall that this feature is also a part of the definition of specialized-simulator UC.)

Theorem 1.1 is therefore an *indication* that the definition of UC security is not overly restrictive. It is only an indication (and not a proof) because "minimality" is proven with respect to specialized-simulator UC, and not with respect to the UC definition itself. Nevertheless, we note that there are no known proofs of security that use non-universal simulation (this includes the proofs of [1, 2] which are universal, even though they are not black-box).[4] Thus, at least with respect to currently

[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 our 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 open.

[4]We are very wary of using such an argument, especially since until recently, black-box simulation was conjectured to be the only way of proving security. (This conjecture was refuted by [1, 2].)

known techniques, there is "no difference" between specialized-simulator UC and the UC definition itself.

We remark that in Section 3.2.3, we show that a natural strengthening of the definition of concurrent 1-bounded general composition implies UC security, without a reversal of quantifiers. This "stronger" definition for general composition requires the existence of a single (or universal) simulator for all protocols that run concurrently to the secure protocol.

Specialized-simulator UC as a security definition. As we have mentioned, Theorem 1.1 demonstrates that specialized-simulator UC is a minimal definition, if security under concurrent general composition is desired. Unfortunately, we do not know if specialized-simulator UC implies concurrent general composition (where any polynomial number of secure protocols can run concurrently in the network), which is really the desired composition operation. Nevertheless, Theorem 1.1 does state that security under concurrent $O(1)$-bounded general composition *is* guaranteed for any protocol that is secure under specialized-simulator UC. This is a significantly weaker composition theorem than that guaranteed by UC security (which ensures security under unbounded general composition). However, as we discuss at the end of Section 3.3, it can still be used to obtain meaningful security guarantees in a real network setting.

The common reference string model. Another possible conclusion from our results relates to the setup assumptions that are needed for secure computation. Specifically, it is well accepted that authenticated channels (typically implemented using a PKI and digital signatures) are needed for obtaining even stand-alone secure computation. Similarly, Corollary 1.2 justifies the use of some additional setup assumption, like that of a common reference string, in order to obtain concurrent general composition. As we have mentioned, universally composable protocols exist for essentially any functionality in the common reference string model [10].

Related work. Most of the research on the topic of concurrent composition of protocols has been for the *self composition* of *specific* problems. The bulk of this work has been on the question of concurrent zero-knowledge; e.g., see [13, 23, 8]. Recently, work on the question of concurrent self composition of general secure computation has also been studied [19, 21]. In this paper, we focus on the *general composition* of *general* secure multi-party computation. Apart from the work on universal composability [6], general composition of multi-party protocols has only been studied in the case of sequential executions. In contrast to the concurrent and parallel cases, stand-alone security implies security under sequential general composition [20, 5]. Unfortunately, sequential composition does not suffice for obtaining security in modern networks.

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 [17, 20, 3, 5]. Our presentation assumes basic familiarity with these definitions of secure multi-party computation.

Notation and conventions. The security parameter is denoted by n (and often, for simplicity, we assume that this is also the length of a party's input); all parties and adversaries run in time that is polynomial in n. Computational indistinguishability is denoted by $\overset{c}{\equiv}$, and when we say that two ensembles $\{X(n,a)\}_{n\in\mathsf{N},a\in\{0,1\}^*}$ and $\{Y(n,a)\}_{n\in\mathsf{N},a\in\{0,1\}^*}$ are computationally indistinguishable, we mean that this holds for all a and for all sufficiently large n's (this is in contrast to where quantification is over all sufficiently large a's as well). 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).

2.1 Security Under General Composition

We now present a definition of security for multi-party 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 to protocol designers who need to prove security in a stand-alone setting only, and can then derive security under composition by just applying the given composition theorem. However, 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.[5] 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 [5, 6].

Motivating discussion. In the setting of 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 are 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 "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

[5]An example of where such an "overshoot" occurred was regarding the *sequential* general composition of secure multi-party protocols. Specifically, [20] presented a definition for stand-alone security and showed that sequential general composition holds for any protocol meeting this definition. However, the definition of [20] 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 [5] 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 [5] (and thus compose sequentially), and yet are not secure by the definition of [20]. Thus, at least as far as sequential composition is concerned, the definition of [20] is "overly restrictive".

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.

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.[6] Furthermore, our equivalence between concurrent general composition and UC security depends heavily on this stringency. Nevertheless, as argued above, this stringency is important for modelling the security needs of modern networks.

Multi-party computation. A multi-party protocol problem for a set of parties $P_1, \ldots, P_m$ is cast by specifying a (probabilistic polynomial-time) multi-party 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.[7] 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.

The hybrid model. Let π be an arbitrary protocol that utilizes ideal interaction with a trusted party computing a multi-party 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 call to $\mathcal{F}$.

As we have mentioned, the adversary controls the scheduling of all messages, including both standard and ideal messages. As usual, we assume that the parties are connected via authenticated channels. Therefore, the adversary can read all standard messages, and may use this knowledge to decide when, if ever, to deliver a message. (We remark that the adversary cannot, however, modify

[6]This issue should not be confused with the issue of "stateless" versus "stateful" composition. We consider stateless composition here, and each execution of ρ is run 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.

[7]This is a rather non-standard way of defining static adversaries. Specifically, the adversary cannot choose who to corrupt at the onset of the computation as a function of its auxiliary input. We rely on this more stringent notion of static adversaries in one of our results. We remark that this stringent definition has been used in the context of two-party computation; see [14].

messages or insert messages of its own.) In contrast, the channels connecting the participating parties and the trusted third party are both authenticated *and* private. Thus, the adversary cannot read the ideal messages, even though it delivers them.

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. 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 tape, 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 for every i, $x_i \in \{0,1\}^n$). Then, the hybrid execution of π with ideal functionality $\mathcal{F}$, denoted $\textsc{hybrid}^{\mathcal{F}}_{\pi,\mathcal{S}}(n, \overline{x}, z)$, 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 multi-party 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 $\textsc{real}_{\pi^\rho,\mathcal{A}}(n, \overline{x}, z)$, 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 realizes $\mathcal{F}$ under concurrent general composition *if for every 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:*

$$\left\{\mathrm{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}}(n,\overline{x},z)\right\}_{n\in\mathsf{N};\overline{x}\in(\{0,1\}^n)^m;z\in\{0,1\}^*} \stackrel{\mathrm{c}}{\equiv} \left\{\mathrm{REAL}_{\pi^\rho,\mathcal{A}}(n,\overline{x},z)\right\}_{n\in\mathsf{N};\overline{x}\in(\{0,1\}^n)^m;z\in\{0,1\}^*}$$

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

Note that non-uniformity of the adversary follows from the fact that the quantification is over all inputs, including the auxiliary input z received by the adversary.

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 for the following reasons. First, 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., concurrent 1-bounded general composition).

Another reason that we consider concurrent bounded general composition is that we show a full equivalence between concurrent $O(1)$-bounded general composition and specialized-simulator UC. In contrast, we do not know the full relationship between concurrent (unbounded) general composition and UC security.

2.2 Universal Composability (UC) and Specialized-Simulator UC

In this paper we show that security under 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 [6].

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" $\mathcal{Z}$.

Formal definitions of the real and ideal model executions can be found in [6]. 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 $\textsc{uc-real}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)$. 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 $\textsc{uc-ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)$. 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 multi-party protocol. Then:*

- (UC security [6]): *We say that ρ* UC realizes *$\mathcal{F}$ 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 $\mathcal{Z}$,*

$$\left\{\textsc{uc-ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N};z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\textsc{uc-real}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N};z\in\{0,1\}^*}$$

- (specialized-simulator UC): *We say that ρ* securely realizes *$\mathcal{F}$* 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,*

$$\left\{\textsc{uc-ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N};z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\textsc{uc-real}_{\rho,\mathcal{A},\mathcal{Z}}(n,z)\right\}_{n\in\mathsf{N};z\in\{0,1\}^*}$$

"Universal" composition. As we have mentioned, a protocol that is universally composable is secure under concurrent general composition with arbitrary sets of parties [6]. Thus, the UC definition provides security under the "strongest" type of composition.

3 The Main Theorem

In this section, we prove our main theorem, stating that security under concurrent $O(1)$-bounded general composition is equivalent to specialized-simulator UC security. Formally,

Theorem 3 *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ securely realizes $\mathcal{F}$ under concurrent $O(1)$-bounded general composition with arbitrary sets of parties* if and only if *ρ securely realizes $\mathcal{F}$ under specialized-simulator UC. This holds for both adaptive and static adversaries.*

3.1 1-Bounded General Composition Implies Specialized-Simulator UC

We begin by proving the direction of Theorem 3 that states that any protocol that is secure under concurrent $O(1)$-bounded general composition is also secure under under specialized-simulator UC. Our proof here is for adaptive adversaries; the static adversarial case is demonstrated in Section 3.2.1. Following the proof, we also consider other variants for slightly different settings.

Notice that in the lemma statement below we refer to concurrent 1-bounded general composition (rather than $O(1)$-bounded as in Theorem 3). This only strengthens the theorem for this direction.

Lemma 3.1 *Let ρ be a protocol and $\mathcal{F}$ a functionality. If ρ securely realizes $\mathcal{F}$ under concurrent* 1-*bounded general composition with arbitrary sets of parties and adaptive adversaries, then ρ securely realizes $\mathcal{F}$ under specialized-simulator UC with adaptive adversaries.*

Proof: The intuition behind the proof of this lemma is as follows. If a protocol is secure under concurrent 1-bounded general composition, then it can be composed with any protocol. In particular, it can be composed with a protocol in which one of the parties plays the role of the UC environment $\mathcal{Z}$. 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 realizes $\mathcal{F}$ under concurrent 1-bounded general composition with arbitrary sets of parties. We wish to show that ρ securely realizes $\mathcal{F}$ under specialized-simulator UC. This involves constructing a UC-type simulator $\mathcal{S}$ for every adaptive adversary $\mathcal{A}$ and environment $\mathcal{Z}$. Thus, fix an adaptive adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. In order to construct $\mathcal{S}$, we use the fact that ρ is secure under concurrent 1-bounded 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 $\mathcal{Z}$. We now define such a protocol π, involving parties $P_{\mathcal{Z}}, P_{\text{adv}}$ and $P_1, \ldots, P_m$. Loosely speaking, $P_{\mathcal{Z}}$ will play the role of the UC environment $\mathcal{Z}$ and P_{adv}, when corrupted, will play the role of the UC (real-model or ideal-model) adversary. We note that in the definition of protocol π, one-time pads are used for communicating between $P_{\mathcal{Z}}$ and $P_1, \ldots, P_m$. This is because in the UC model, $\mathcal{Z}$ and the parties communicate values that are kept secret from the adversary (specifically, this communication consists of $\mathcal{Z}$ writing on the parties' input tapes and reading their output tapes). 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. (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 and $2m$ strings $r_1^a, r_1^b, \ldots, r_m^a, r_m^b$, each uniformly distributed in $\{0,1\}^n$. For every i, party P_i receives the pair of strings r_i^a and r_i^b. (Thus, $P_{\mathcal{Z}}$ shares 2 one-time pads with each P_i.[8])

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 $P_{\mathcal{Z}}$ 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 sends the pair $(\mathsf{input}, \alpha_i)$ to P_i. During the execution, $P_{\mathcal{Z}}$ receives tuples $(\mathsf{output}, P_i, \beta_i)$ from the parties $P_1, \ldots, P_m$ (see step (3) below). When $\mathcal{Z}$ wishes to read the output tape of a party P_i, then if $P_{\mathcal{Z}}$ has received such an output message from P_i, it internally hands $\mathcal{Z}$ the value $y_i = \beta_i \oplus r_i^b$. 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).

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 $1 \le i \le m$): When party P_i receives a message $(\mathsf{input}, \alpha_i)$ from $P_{\mathcal{Z}}$, it computes $x_i = \alpha_i \oplus r_i^a$ and sends x_i to the ideal functionality $\mathcal{F}$. When P_i receives its output y_i from $\mathcal{F}$, it computes $\beta_i = y_i \oplus r_i^b$ and sends the tuple $(\mathsf{output}, P_i, \beta_i)$ to $P_{\mathcal{Z}}$.

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 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_{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 $\mathcal{Z}$.[9] 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. Finally, messages between $P_{\mathcal{Z}}$ and the parties $P_1, \ldots, P_m$ 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 $P_{\mathcal{Z}}$ 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

[8]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. Thus, we can assume that the one-time pads are indeed random.

[9]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.

$\mathcal{A}$. That is, for a given value z, define the following distribution over parties' inputs:

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

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

$$\left\{\text{REAL}_{\pi^\rho, \mathcal{A}_\pi}(n, \overline{X}_n(z), \lambda)\,|_{P_\mathcal{Z}}\right\}_{n\in\mathsf{N}; z\in\{0,1\}^*} \equiv \left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n, z)\right\}_{n\in\mathsf{N}; z\in\{0,1\}^*} \tag{1}$$

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, $P_\mathcal{Z}$ 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 identical to the inputs of the parties to ρ in the UC setting (recall that $P_\mathcal{Z}$ 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 $P_\mathcal{Z}$ 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-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 concurrent 1-bounded general composition and arbitrary sets of parties, we have that there exists a hybrid-model simulator $\mathcal{S}_\pi$ such that

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

In particular, it follows that the output of $P_\mathcal{Z}$ in the HYBRID setting is computationally indistinguishable to its output in the REAL setting. That is, for every input vector $\overline{x}$, every auxiliary input z for $\mathcal{A}_\pi$ and all sufficiently large n's

$$\left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n, \overline{x}, z)\,|_{P_\mathcal{Z}}\right\} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{x}, z)\,|_{P_\mathcal{Z}}\right\} \tag{2}$$

Of course, a special case of Eq. (2) is for the case that $\overline{x}$ is chosen according to $\overline{X}_n(z)$ as defined 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 $\mathcal{S}_\pi$ interacts with parties $P_\mathcal{Z}, P_{\text{adv}}, P_1, \ldots, P_m$. Furthermore, $\mathcal{S}$ interacts in an ideal execution with $\mathcal{F}$, whereas $\mathcal{S}_\pi$ interacts in a hybrid execution of $\pi^\mathcal{F}$. 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 $\mathcal{S}_\pi$ can also not corrupt $P_\mathcal{Z}$.[10] (This holds because the set of corrupted parties can

[10]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 $P_\mathcal{Z}$.

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 REAL and HYBRID executions.) Thus, $\mathcal{S}$ does not need to deal with the case that $P_\mathcal{Z}$ is corrupted by $\mathcal{S}_\pi$.

- *Emulation of $P_{\rm adv}$:* First, note that until the point that $P_{\rm adv}$ is corrupted, it does nothing. Thus, the only emulation required is when $\mathcal{S}_\pi$ corrupts $P_{\rm adv}$. In this case, all $\mathcal{S}$ needs to do is to hand $\mathcal{S}_\pi$ the series of messages that $P_{\rm adv}$ would have received so far. However, by the construction of π, party $P_{\rm adv}$ 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 $P_{\rm adv}$ 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 $P_{\rm adv}$. 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 $(\mathsf{input}, \alpha_i)$ to P_i, for $\alpha_i \in_R \{0,1\}^n$. Likewise, when $\mathcal{Z}$ reads the output tape of an uncorrupted party P_i, simulator $\mathcal{S}$ emulates P_i sending $(\mathsf{output}, P_i, \beta_i)$ to $P_\mathcal{Z}$, for $\beta_i \in_R \{0,1\}^n$.[11]

  If $\mathcal{S}_\pi$ corrupts party P_i before $\mathcal{Z}$ writes a value on its input tape, then $\mathcal{S}$ chooses a random pair (r_i^a, r_i^b) and hands this 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 $(\mathsf{input}, x_i \oplus r_i^a)$ to P_i. Likewise, when P_i receives its output y_i from $\mathcal{F}$, simulator $\mathcal{S}$ emulates P_i sending $(\mathsf{output}, P_i, y_i \oplus r_i^b)$ to $P_\mathcal{Z}$. 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.

  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 $(\mathsf{input}, \alpha_i)$ to P_i. 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 $r_i^b \in_R \{0,1\}^n$, and hands $\mathcal{S}_\pi$ the pair (r_i^a, r_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. 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:* $\mathcal{S}$ deliver a messages between $\mathcal{F}$ and a party P_i whenever $\mathcal{S}_\pi$ would deliver the corresponding message between $\mathcal{F}$ and P_i in the hybrid execution of π. When $\mathcal{S}_\pi$ wishes to deliver a message from a corrupted party P_i to $\mathcal{F}$, simulator $\mathcal{S}$ sends $\mathcal{F}$ the value written on the outgoing communication tape of the emulated P_i, as intended for $\mathcal{F}$.

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 $P_\mathcal{Z}$ runs the code of $\mathcal{Z}$. Therefore, the messages sent from $P_\mathcal{Z}$ to $P_{\rm adv}$ and $P_1, \ldots, P_m$ are the same 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 identical in both scenarios. This is because the

[11]The above description assumes that $\mathcal{S}$ knows when $\mathcal{Z}$ writes to a party's input tape and reads from its output tape. The fact that $\mathcal{S}$ knows when $\mathcal{Z}$ writes to a party's input tape is due to the following. 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. Regarding the time that $\mathcal{Z}$ reads a party's output tape, without loss of generality, we can assume that $\mathcal{Z}$ reads the tape as soon as it is written to. Now, in the UC ideal model, when a party receives its output from the ideal functionality, it is activated and immediately copies this output to its output tape. Since $\mathcal{S}$ delivers all outputs from the ideal functionality to the parties, it thus knows when outputs are written.

honest parties receive their inputs from $P_{\mathcal{Z}}$ in π and from $\mathcal{Z}$ in the ideal execution with $\mathcal{S}$; they are therefore the same. Likewise, the corrupted parties in π are instructed by $\mathcal{S}_\pi$ regarding their inputs to $\mathcal{F}$, and $\mathcal{S}$ forwards these same instructions. This all implies that $\mathcal{Z}$'s view in the ideal execution of $\mathcal{F}$ with $\mathcal{S}$ is identical to $P_{\mathcal{Z}}$'s view in the $\mathcal{F}$-hybrid execution of π with $\mathcal{S}_\pi$. Thus,

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

where $\overline{x}$ is chosen according to $\overline{X}_n(z)$, as defined in Eq. (1). The lemma is obtained by combining Equations (1) to (3). ■

Remark – specialized simulation. According to Definition 1 (security under general composition), a different simulator $\mathcal{S}_\pi$ may be provided for every protocol π. Therefore, in the proof of Lemma 3.1, 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 Lemma 3.1 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 Lemma 3.1 for Alternative Settings

Before proving the other direction of Theorem 3, we consider alternative settings that yield variants of Lemma 3.1. These variants will be used later to obtain stronger impossibility results for general composition. (Unfortunately the placement of this section here interrupts the natural flow of the proof of Theorem 3. Despite this, we present this section here as it refers closely to the proof of Lemma 3.1.)

3.2.1 Static adversaries

Lemma 3.1 refers to adaptive adversaries. However, we can actually prove a stronger result when considering static (or non-adaptive) adversaries. Specifically, we can show that 1-bounded general composition for *a single set of parties* implies specialized-simulator UC. Note that in the proof of Lemma 3.1, we use the fact that there are arbitrary sets of parties in an essential way. Specifically, we define *additional* parties $P_{\mathcal{Z}}$ and P_{adv}, and it is crucial that $P_{\mathcal{Z}}$ is not corrupted and that P_{adv} is corrupted. ($P_{\mathcal{Z}}$ cannot be corrupted because then the simulator $\mathcal{S}$ would have to provide $\mathcal{S}_\pi$ with the appropriate internal state of $\mathcal{Z}$, which it cannot do. Likewise, P_{adv} must be corrupted in order to simulate communication between $\mathcal{A}$ and $\mathcal{Z}$, even when no parties are corrupted.) Now, if $P_{\mathcal{Z}}$ was played by a party P_i then the simulation would fail in the case that the adaptive adversary corrupts party P_i; likewise for P_{adv}.

Proposition 4 *Let ρ be a protocol and $\mathcal{F}$ a functionality. If ρ securely realizes $\mathcal{F}$ under concurrent* 1*-bounded general composition with a single set of parties and* static *adversaries, then ρ securely realizes $\mathcal{F}$ under specialized-simulator UC with* static *adversaries.*

Proof Sketch: The proof of this proposition is very similar to the proof of Lemma 3.1; the only differences are as follows. As above, we start with a protocol ρ for parties $P_1, \ldots, P_m$, an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, and we define a protocol π. However, in contrast to the proof of Lemma 3.1, we do not introduce additional parties $P_{\mathcal{Z}}$ and P_{adv}. Rather, protocol π is defined for the same set of parties $P_1, \ldots, P_m$, and we have two of the parties P_i and P_j play the roles of P_{adv} and $P_{\mathcal{Z}}$. Recall that in the proof of Lemma 3.1, protocol π is defined after both $\mathcal{Z}$ and $\mathcal{A}$ are fixed. Now, since $\mathcal{A}$ is a static adversary, the set of corrupted parties is fixed. Therefore, we can single out one corrupted party P_i and one honest party P_j.[12] In the definition of protocol π, party P_i will play the role of P_{adv} (as well as P_i) and party P_j will play the role of $P_{\mathcal{Z}}$ (as well as P_j). Since we are guaranteed that P_i is corrupted and P_j is not, everything else in the proof remains the same. ■

Caveat. The proof of Proposition 4 is *very* 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. For example, consider the definition of the static adversarial model where all the corruptions take place before any parties are activated (as defined in [6]). In this case, the set of corrupted parties can depend on $\mathcal{Z}$'s input z and the proof does not work.

Arbitrary and single sets of parties with static adversaries. An interesting corollary from Proposition 4 and the other direction of Theorem 3 to be proved below, is the fact that when static adversaries are considered, security for arbitrary sets of parties is equivalent to security for a single set of parties. This can be seen as follows. Proposition 4 states that security under concurrent general composition with static adversaries and a *single* set of parties implies specialized-simulator UC with static adversaries. Furthermore, Lemma 3.2 (the other direction of Theorem 3) states that specialized-simulator UC with static adversaries implies security under concurrent general composition with static adversaries and *arbitrary* sets of parties. Equivalence between a single set and arbitrary sets of parties therefore follows. We do not know if the same is true for adaptive adversaries.

3.2.2 Parallel general composition

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.

[12]This holds only if $\mathcal{A}$ corrupts a proper, non-empty subset of the parties. That is, if $\mathcal{A}$ corrupts all parties, then no party can play $P_{\mathcal{Z}}$, and if $\mathcal{A}$ corrupts no parties, then no party can play P_{adv}. This is solved as follows. If $\mathcal{A}$ corrupts all parties, then simulation is straightforward. Specifically, all $\mathcal{S}$ needs to do is forward messages between $\mathcal{A}$ and $\mathcal{Z}$. In the other case where $\mathcal{A}$ corrupts no parties, this is solved by having P_{adv} (or in this setting some P_i) run $\mathcal{A}$'s code, in the same way that $P_{\mathcal{Z}}$ is defined so that it runs $\mathcal{Z}$'s code. Note that we could have also done this in the proof of Lemma 3.1, instead of having $\mathcal{A}_\pi$ internally run $\mathcal{A}$'s code.

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 in 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.[13] 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 [6] (for concurrent composition) and the impossibility results of [9] also hold in this case. We now show that *parallel* 1-bounded 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 5 *Let ρ be a protocol and $\mathcal{F}$ a functionality. If ρ securely realizes $\mathcal{F}$ under* parallel *1-bounded general composition for a single set of parties and static adversaries, then ρ securely realizes $\mathcal{F}$ 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 [9] hold also for synchronized specialized-simulator UC.[14] Thus, as we will show in Section 4, Proposition 5 implies impossibility for parallel general composition.

3.2.3 A stronger variant of general composition that implies full UC security

In Lemma 3.1 we showed that concurrent 1-bounded general composition implies specialized-simulator UC. In this section, we show that there is a natural strengthening of 1-bounded general composition that implies UC security, *without* any reversal of quantifiers. The strengthening that we consider is with respect to the simulator. Specifically, assume that a protocol ρ is proven secure by providing a single simulator that works for all protocols π. That is, the security of ρ is proven by showing that for every real-model adversary $\mathcal{A}$ there exists an ideal-model adversary $\mathcal{S}$ such that for every protocol π in the $\mathcal{F}$-hybrid model that utilizes a single call to $\mathcal{F}$,

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

Such a simulator is called protocol-universal because it works for every protocol π. (Notice that a different simulator is still allowed for each real-model adversary $\mathcal{A}$. Thus, "universality" is with respect to protocols π and not with respect to real adversaries $\mathcal{A}$.) Now, recall that in the proof of Lemma 3.1, party $P_{\mathcal{Z}}$ played the role of $\mathcal{Z}$ in protocol π. If the simulator $\mathcal{S}_\pi$ works for every protocol π, then this implies that the UC simulator $\mathcal{S}$ that is obtained from $\mathcal{S}_\pi$ works for every environment $\mathcal{Z}$. This implies that for every UC adversary $\mathcal{A}$, there exists a UC simulator $\mathcal{S}$ so that no environment $\mathcal{Z}$ can distinguish between the IDEAL and REAL executions. In other words, protocol ρ is UC secure. We therefore obtain the following proposition:

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

[14] This was not explicitly stated in [9], but is not difficult to see. Specifically, the environment and real-model adversary constructed by [9] in order to prove their results communicate only once each round.

Proposition 6 *Let ρ be a protocol and $\mathcal{F}$ a functionality, and consider an adversarial model with adaptive corruptions. If ρ securely realizes $\mathcal{F}$ under concurrent 1-bounded general composition with arbitrary sets of parties and is proven using* protocol-universal simulation, *then ρ UC realizes $\mathcal{F}$.*

3.3 Specialized-Simulator UC Implies $O(1)$-Bounded General Composition

We now prove the other direction of Theorem 3, stating that any protocol that is secure under specialized-simulator UC is also secure under concurrent $O(1)$-bounded general composition. This is therefore a *composition theorem* for specialized-simulator UC. Following the proof, we discuss how $O(1)$-bounded general composition can be used to obtain meaningful security. (Recall that by Lemma 3.1, specialized-simulator UC is a *minimal* definition for obtaining security under concurrent general composition. It is therefore of interest to see what security guarantees can be provided by this definition.)

Lemma 3.2 *Let ρ be a protocol and $\mathcal{F}$ a probabilistic polynomial-time functionality. If ρ securely realizes $\mathcal{F}$ under specialized-simulator UC, then ρ securely realizes $\mathcal{F}$ under concurrent $O(1)$-bounded general composition with arbitrary sets of parties. This holds for both static and adaptive adversaries.*

Proof: We prove the lemma for adaptive adversaries; the static adversarial case is easily derived. In order to simplify the presentation, we first prove the lemma for concurrent **1**-bounded general composition. This is significantly simpler than the $O(1)$-bounded case, while demonstrating the key issues in the proof. Intuitively, the proof works by having the UC environment $\mathcal{Z}$ internally simulate the entire π-portion of the π^ρ execution. In this way, the security of ρ under specialized-simulator UC can be used to derive its security in the setting of concurrent general composition. More specifically, a simulator $\mathcal{S}$ for this $\mathcal{Z}$ in the UC setting can be used to obtain a simulator $\mathcal{S}_\pi$ for the composed protocol π^ρ.

Let $\mathcal{F}$ be a functionality and ρ a protocol, such that ρ securely realizes $\mathcal{F}$ under specialized-simulator UC. Now, let π be a protocol that utilizes a *single* call to $\mathcal{F}$ and let $\mathcal{A}_\pi$ be a real-model adversary attacking the composed protocol π^ρ. We wish to construct an $\mathcal{F}$-hybrid model adversary $\mathcal{S}_\pi$ such that

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

In order to do this, we first construct a real-model adversary $\mathcal{A}$ and environment $\mathcal{Z}$ for protocol ρ in the *specialized-simulator UC* setting. In the description below, we refer to emulated internal interaction and real external interaction. In particular, the environment $\mathcal{Z}$ will internally emulate the honest parties for the π part of the π^ρ execution. In contrast, the ρ part of the π^ρ execution is executed by the external parties. We will distinguish between π-messages (i.e., messages belonging to π) and ρ-messages (i.e., messages belonging to the sub-protocol ρ).

The real-model adversary $\mathcal{A}$: The adversary $\mathcal{A}$, attacking an execution of ρ, cooperates with $\mathcal{Z}$ to simulate an execution of π^ρ for $\mathcal{A}_\pi$. When $\mathcal{A}$ is activated with auxiliary input z and inputs x_j for the corrupted parties, it internally invokes the adversary $\mathcal{A}_\pi$ and provides it with the auxiliary input z and the inputs x_j. $\mathcal{A}$ deals with π-messages and ρ-messages separately, as follows. When $\mathcal{A}_\pi$ wishes to send a π-message from a corrupted party P_j to an uncorrupted party P_i, then $\mathcal{A}$ sends this message to $\mathcal{Z}$ to be internally forwarded to P_i (recall that $\mathcal{Z}$ internally emulates all the honest parties for the π portion of the protocol). Likewise, when $\mathcal{A}$ receives a message from $\mathcal{Z}$ that is actually a π-message from an uncorrupted P_i to a corrupted P_j, then $\mathcal{A}$ forwards this message to $\mathcal{A}_\pi$. Finally, when $\mathcal{A}_\pi$ wishes to deliver a π-message from an uncorrupted party P_i

to another uncorrupted party $P_{i'}$, then $\mathcal{A}$ notifies $\mathcal{Z}$ to internally deliver this message. (As will be described below, $\mathcal{A}$ is notified by $\mathcal{Z}$ when a π-message is written by an uncorrupted party on its outgoing communication tape; $\mathcal{A}$ forwards all such notifications to $\mathcal{A}_\pi$.)

The above describes the emulation of π-messages. However, when $\mathcal{A}_\pi$ wishes to send a ρ-message from a corrupted P_j to an uncorrupted P_i, then $\mathcal{A}$ *externally* delivers this message from P_j to P_i. Likewise, when an external honest P_i writes a ρ-message on its outgoing communication tape for a corrupted party P_j, then $\mathcal{A}$ hands $\mathcal{A}_\pi$ this message. Finally, when an external uncorrupted P_i writes a ρ-message on its outgoing communication tape for another uncorrupted party $P_{i'}$, then $\mathcal{A}$ notifies $\mathcal{A}_\pi$ of this message. Then, when $\mathcal{A}_\pi$ wishes to deliver this message from P_i to $P_{i'}$, adversary $\mathcal{A}$ externally delivers it.

We conclude by describing how party corruptions are carried out. When $\mathcal{A}_\pi$ wishes to corrupt an honest party P_i, then $\mathcal{A}$ first corrupts the external P_i (thus receiving the private state of P_i from the execution of ρ). Next, $\mathcal{A}$ requests the state of the internally emulated P_i from $\mathcal{Z}$ (thus receiving the private state of P_i from π). $\mathcal{A}$ then combines these states into a single private state of P_i from the entire execution of π^ρ and hands it to $\mathcal{A}_\pi$. At the conclusion of the emulation, $\mathcal{A}$ hands $\mathcal{Z}$ whatever $\mathcal{A}_\pi$ outputs (this is given to $\mathcal{Z}$ and not output by $\mathcal{A}$ because in the UC setting, only $\mathcal{Z}$ has output).

The environment $\mathcal{Z}$: As described above, $\mathcal{Z}$ internally emulates the roles of the honest parties in the execution of π. In contrast, the execution of ρ is carried out externally. We consider the case that $\mathcal{Z}$'s input z' equals $(\overline{x}, z)$, where $\overline{x}$ is supposedly the parties' inputs to π and z is $\mathcal{A}_\pi$'s auxiliary input. Thus, upon its first activation, $\mathcal{Z}$ writes z on $\mathcal{A}$'s input tape. Furthermore, $\mathcal{Z}$ internally invokes each honest party P_i with input x_i (taken from $\overline{x}$), and externally sends the inputs x_j of the corrupted parties to $\mathcal{A}$.

When the instructions of an honest P_i are that it should write some π-message on its outgoing communication tape for some other party, then $\mathcal{Z}$ internally simulates this action and externally notifies $\mathcal{A}$ of the message that has been written. Furthermore, when $\mathcal{Z}$ externally receives a message from $\mathcal{A}$ that consists of a π-message to an honest P_i from a corrupted P_j, then $\mathcal{Z}$ internally invokes P_i on this message and writes its response. When $\mathcal{A}$ instructs $\mathcal{Z}$ that a π-message should be delivered from one honest party to another, then $\mathcal{Z}$ delivers the message in the internal emulation.

Now, when an internally emulated honest P_i obtains a value y_i that is to be used as input into ρ, the environment $\mathcal{Z}$ writes the value y_i to the *external* P_i's input tape (recall that this external party runs ρ only). Likewise, when the external P_i writes its ρ-output to its output tape, $\mathcal{Z}$ reads this output and hands it to the internal P_i as its output from ρ. $\mathcal{Z}$ continues the internal emulation of P_i until it concludes π.[15]

In addition to the above, when $\mathcal{A}$ instructs $\mathcal{Z}$ that an honest party P_i is being corrupted, then $\mathcal{Z}$ sends $\mathcal{A}$ the internal state of the internally emulated P_i. At the conclusion, $\mathcal{Z}$ outputs the output of $\mathcal{A}_\pi$ as received from $\mathcal{A}$, along with the outputs of all the honest parties that it internally emulated. This concludes the description of $\mathcal{Z}$.

It is clear that a UC real-model execution of ρ with the above-described $\mathcal{A}$ and $\mathcal{Z}$ perfectly emulates a real execution of the composed protocol π^ρ with adversary $\mathcal{A}_\pi$. That is,

$$\left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n, z')\right\} \equiv \left\{\text{REAL}_{\pi^\rho,\mathcal{A}_\pi}(n, \overline{x}, z)\right\} \tag{4}$$

[15]This perfectly emulates the honest P_i in π^ρ. Recall that in the composed protocol π^ρ, party P_i invokes a *separate* ITM for ρ with input y_i. This is therefore the same as when $\mathcal{Z}$ runs the instructions of P_i for π, and a separate party (i.e., the real external P_i) runs P_i's instructions for ρ.

where $z' = (\overline{x}, z)$. Now, by the assumed security of ρ under specialized-simulator UC, we have that there exists an ideal-model simulator $\mathcal{S}$ such that

$$\left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n, z')\right\} \overset{\text{c}}{\equiv} \left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n, z')\right\} \tag{5}$$

We now use the ideal-model simulator $\mathcal{S}$ to construct an $\mathcal{F}$-hybrid-model simulator $\mathcal{S}_\pi$ for π. Intuitively, $\mathcal{S}_\pi$ will simulate the specialized-simulator UC setting for $\mathcal{S}$ (this includes simulating $\mathcal{Z}$), while "attacking" π^ρ.

The $\mathcal{F}$-hybrid-model simulator $\mathcal{S}_\pi$: Notice that the simulator $\mathcal{S}_\pi$ interacts with external parties running π and an ideal functionality for $\mathcal{F}$, whereas $\mathcal{S}$ interacts with an environment $\mathcal{Z}$ and an ideal functionality for $\mathcal{F}$. The communication between $\mathcal{S}$ and $\mathcal{F}$ is forwarded unmodified by $\mathcal{S}_\pi$; e.g., when $\mathcal{S}$ wishes to send a value to $\mathcal{F}$ from a corrupted party, then $\mathcal{S}$ sends this same message to $\mathcal{F}$. Apart from this, $\mathcal{S}_\pi$ must simulate $\mathcal{S}$'s communication with $\mathcal{Z}$. This communication consists of all the π-messages sent and of information exchanged upon corruption of a party. The key point to notice is that $\mathcal{Z}$'s treatment of π-messages is identical to that of the external honest parties running π in the setting of $\mathcal{S}_\pi$. Therefore, $\mathcal{S}_\pi$ can simulate this interaction with $\mathcal{Z}$, based solely on the messages sent by the honest parties in the real external interaction. (It is important here that $\mathcal{S}$ has only "one-pass black-box" access to $\mathcal{Z}$. This is because $\mathcal{S}_\pi$ cannot rewind the honest parties in its setting and it uses the messages generated by these parties to simulate $\mathcal{Z}$.) We conclude that $\mathcal{S}_\pi$'s simulation only involves re-addressing messages that are sent. For example, when an uncorrupted party P_i writes a π-message on its outgoing tape for a party P_j in π^ρ, then $\mathcal{S}_\pi$ simulates $\mathcal{S}$ being informed by $\mathcal{Z}$ that this message was written by an honest party.

At the conclusion of the simulation, $\mathcal{S}_\pi$ outputs whatever output $\mathcal{S}$ sends to $\mathcal{Z}$ (recall that $\mathcal{A}$ sends $\mathcal{A}_\pi$'s output to $\mathcal{Z}$).

From the above description, we have that the simulation by $\mathcal{S}_\pi$ for $\mathcal{S}$ is perfect. Furthermore, the output of the honest parties in the execution of π^ρ with $\mathcal{S}_\pi$ is identical to the output generated by $\mathcal{Z}$ in the UC execution of ρ. We therefore conclude that

$$\left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(n, \overline{x}, z)\right\} \overset{\text{c}}{\equiv} \left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n, z')\right\} \tag{6}$$

where $z' = (\overline{x}, z)$. The proof for the case of concurrent 1-bounded general composition follows by combining Equations (4) to (6).

Extending the proof to $O(1)$-bounded composition. The above proof deals with the case that π contains a single ideal call to functionality $\mathcal{F}$. We now show how this can be extended to the case that a *constant* c number of ideal calls are made. Let $\text{HYBRID}^{\mathcal{F}_1,\ldots,\mathcal{F}_c}_{\pi,\mathcal{S}}(n, \overline{x}, z)$ denote the hybrid-model execution with adversary $\mathcal{S}$ where all c calls are made using access to the ideal functionality $\mathcal{F}$ (we denote the different instances of $\mathcal{F}$ with different names in order to distinguish them). Furthermore, let $\text{REAL}_{\pi^{\rho_1,\ldots,\rho_c},\mathcal{A}}(n, \overline{x}, z)$ denote the real-model execution with adversary $\mathcal{A}$ where all c calls are made using real executions of protocol ρ. The extension of the proof to this case is presented somewhat informally.

The main idea in extending the proof is to construct a hybrid execution H_i, where the first i calls are made using $\mathcal{F}$ and the last c–i calls are made using ρ; this execution with adversary $\mathcal{S}_i$ is denoted $H_i = \text{HYBRID}^{\mathcal{F}_1,\ldots,\mathcal{F}_i}_{\pi^{\rho_{i+1},\ldots,\rho_c},\mathcal{S}_i}(n, \overline{x}, z)$. A technical difficulty that arises here in the case of specialized-simulator UC is that the simulator may depend on the environment. Now, since the environment that emulates H_i will be different to the environment that emulates H_{i+1}, a different simulator is needed for every hybrid. We therefore show how to construct a series of simulators $\mathcal{S}_1, \ldots, \mathcal{S}_c$ inductively, so that for every i, an execution of H_i with $\mathcal{S}_i$ is computationally indistinguishable from an execution of REAL (i.e., H_0) with $\mathcal{A}$.

The assumption for the induction is that for every i and every real-model adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}_i$ for H_i such that

$$\left\{\text{HYBRID}^{\mathcal{F}_1,\ldots,F_i}_{\pi^{\rho_{i+1},\ldots,\rho_c},\mathcal{S}_i}(n,\overline{x},z)\right\} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi^{\rho_1,\ldots,\rho_c},\mathcal{A}}(n,\overline{x},z)\right\} \tag{7}$$

In the base case of the induction, we can take $\mathcal{S}_0 = \mathcal{A}$ and $\mathcal{S}_1$ to be the simulator constructed in the above proof for the case of a single call to $\mathcal{F}$. Notice that in H_1, there is indeed only a single call to $\mathcal{F}$. Therefore, the proof for the case of a single call works here. (Technically one needs to define a real protocol π' such that $\pi' = \pi^{\rho_2,\ldots,\rho_c}$ and the proof goes through.)

We now show how to construct a simulator $\mathcal{S}_{i+1}$ for the i+1th hybrid H_{i+1} from the simulator $\mathcal{S}_i$. The idea is very similar to the case of a single ideal call, as proved above. The only difference here is that there are ideal calls to $\mathcal{F}_1, \ldots, \mathcal{F}_i$ that did not exist before. Nevertheless, these are easily dealt with as follows. As in the proof for a single ideal call, an environment $\mathcal{Z}$ is constructed who internally emulates the roles of the honest parties in π, *along* with their ideal calls to $\mathcal{F}_1, \ldots, F_i$ and their executions of $\rho_{i+2}, \ldots, \rho_c$. The messages from the executions of $\rho_{i+2}, \ldots, \rho_c$ are dealt with in the same way as π-messages are dealt with. However, the ideal calls to $\mathcal{F}_1, \ldots, F_i$ are dealt with differently, in that the environment $\mathcal{Z}$ *internally* plays the ideal functionality for these calls (for this reason, the functionality must be computable in probabilistic polynomial-time). Thus $\mathcal{Z}$ internally emulates the honest parties for all the parts of the protocol except for ρ_{i+1}. In contrast, as above, the execution of ρ_{i+1} is run by the external parties ($\mathcal{Z}$ writes their inputs and reads their outputs for this execution in the same way). It remains to describe the UC-setting adversary $\mathcal{A}$. Analogously to the adversary for the case of a single ideal call, adversary $\mathcal{A}$ controls all of the corrupted parties and interacts with $\mathcal{Z}$ as needed for communicating with the honest parties for all parts of the execution except for ρ_{i+1}. One important difference, however, is that $\mathcal{A}$ internally runs the simulator $\mathcal{S}_i$ for the corrupted parties' ideal calls to $\mathcal{F}_1, \ldots, \mathcal{F}_i$. Furthermore, the ideal messages sent by $\mathcal{S}_i$ to $\mathcal{F}_1, \ldots, \mathcal{F}_i$ are sent by $\mathcal{A}$ to $\mathcal{Z}$, who correctly computes the response of the functionality and returns it to $\mathcal{A}$. Using very similar arguments to the single-call case, we obtain that

$$\left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n,z')\right\} \equiv \left\{\text{HYBRID}^{\mathcal{F}_1,\ldots,F_i}_{\pi^{\rho_{i+1},\ldots,\rho_c},\mathcal{S}_i}(n,\overline{x},z)\right\} \tag{8}$$

Likewise, by the assumed security of ρ under specialized-simulator UC, we have that there exists an ideal-model simulator $\mathcal{S}$ such that

$$\left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z')\right\} \overset{\text{c}}{\equiv} \left\{\text{UC-REAL}_{\rho,\mathcal{A},\mathcal{Z}}(n,z')\right\} \tag{9}$$

Finally, as in the single-call case, it remains to construct a simulator $\mathcal{S}_{i+1}$ for the hybrid execution H_{i+1} from the UC-setting simulator $\mathcal{S}$. Again, the construction is almost the same as in the single-call case and it involves using the real honest parties and ideal functionalities in order to simulate $\mathcal{Z}$ for $\mathcal{S}$. We therefore obtain

$$\left\{\text{HYBRID}^{\mathcal{F}_1,\ldots,F_{i+1}}_{\pi^{\rho_{i+2},\ldots,\rho_c},\mathcal{S}_{i+1}}(n,\overline{x},z)\right\} \overset{\text{c}}{\equiv} \left\{\text{UC-IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(n,z')\right\} \tag{10}$$

Then, combining Equations (8) to (10), we obtain indistinguishability between the hybrid executions H_i and H_{i+1}. By combining this with the inductive assumption stated in Eq. (7), we have that for every $\mathcal{A}$ there exists a simulator $\mathcal{S}_i$ such that

$$\left\{\text{HYBRID}^{\mathcal{F}_1,\ldots,F_i}_{\pi^{\rho_{i+1},\ldots,\rho_c},\mathcal{S}_i}(n,\overline{x},z)\right\} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi^{\rho_1,\ldots,\rho_c},\mathcal{A}}(n,\overline{x},z)\right\} \tag{11}$$

and thus the inductive step holds.[16] We conclude that for every i, there exists a simulator $\mathcal{S}_i$ such that an execution of H_i with $\mathcal{S}_i$ cannot be distinguished from an execution of H_0 (i.e., REAL) with $\mathcal{A}$. Therefore, $\mathcal{S}_c$ is a "good" simulator for $\pi^{\rho_1,\ldots,\rho_c}$.

It remains to show that $\mathcal{S}_c$ runs in polynomial-time. In order to see this, denote the running-time of all the honest parties in $\pi^{\rho_1,\ldots,\rho_c}$ by $q(n)$ and the running-time of the real-model adversary $\mathcal{A}$ by $p(n)$. Now, by the fact that ρ is specialized-simulator UC, we have that $\mathcal{S}_1$ runs in time that is polynomial in the adversary and environment. It therefore runs in time that is polynomial in $p(n) + q(n)$ (recall that the environment for this case essentially just plays the role of the honest parties). Likewise, $\mathcal{S}_2$ runs in time that is polynomial in $p(n) + q(n) + \mathsf{time}(\mathcal{S}_1)$. We therefore have at most a polynomial increase in the running-time of the simulator each time. In general, this can yield an exponential running-time (because after i iterations, the complexity of $\mathcal{S}_i$ can be $n^{O(i)}$). However, since c is constant, we conclude that $\mathcal{S}$ runs in polynomial-time. This completes the proof. ■

Different functionalities and nested hybrid calls. Lemma 3.2 relates to a case where a protocol π utilizes a constant number of ideal calls to a single functionality $\mathcal{F}$. However, nothing is changed when a constant number of *different* functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_c$ are used. In contrast, another possible scenario is where a protocol π utilizes ideal calls to $\mathcal{F}$ only. However, the secure protocol ρ that realizes $\mathcal{F}$ utilizes ideal calls to another functionality $\mathcal{G}$. In this case, the real-model protocol is $\pi^{\rho^{\tau}}$, where τ securely realizes $\mathcal{G}$. We note that security in this case does hold for specialized-simulator UC, but it requires significant modifications to the proof of Lemma 3.2.

Using specialized-simulator UC. Lemma 3.2 provides a rather limited composition operation in that only a constant number of ideal calls to $\mathcal{F}$ can be made. This is in contrast to the composition theorem of [6] for the UC definition which allows any polynomial number of calls to $\mathcal{F}$. Despite this limitation, Lemma 3.2 can be used to obtain security in a real setting where functionalities may be called a polynomial number of times. This is obtained by defining *multi-session functionalities.* Such a functionality is an extension of a stand-alone functionality, in that the functionality itself explicitly allows for any polynomial number of invocations of the basic functionality. (We note that multi-session functionalities have been used to bypass problems which arise in the context of universal composition in the common reference string model [7, 10, 11].) The basic idea is that instead of designing a protocol and proving it secure in a stand-alone setting (with an environment $\mathcal{Z}$), security is proven with respect to many concurrent executions of the *same protocol* (and with an environment $\mathcal{Z}$). In other words, concurrent self composition with respect to an environment is proven. The composition operation provided by Lemma 3.2 then states that any constant number of *different* protocols are secure when run any *polynomial* number of times concurrently with each other and with an arbitrary protocol π. Such a setting models the security needs in modern networks. (Of course, designing and proving protocols is harder in this setting, because concurrent executions must explicitly be considered. Nevertheless, the bottom line is that the desired level of security can be achieved.)

[16] This is shown formally as follows. Assume that there exists a distinguisher D who can distinguish (with non-negligible advantage) between the result of an execution of H_{i+1} with adversary $\mathcal{S}_{i+1}$, and a REAL execution with adversary $\mathcal{A}$. Then, D can either distinguish between H_{i+1} and H_i or between H_i and REAL. Both of these possibilities have been ruled out; the first by combining Equations (8) to (10) and the second by the inductive hypothesis.

4 Impossibility Results for General Composition

An important ramification of Theorem 3, and Propositions 4 and 5, is that known impossibility results for specialized-simulator UC apply to concurrent (and even parallel) 1-bounded 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. Furthermore, they apply for any type of simulation (and not just "black-box" simulation).

Impossibility for specialized-simulator UC. The following impossibility results for specialized-simulator UC and static adversaries were shown in [9]:[17]

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 one-way,[18] then $\mathcal{F}_f$ cannot be securely realized under specialized-simulator UC. We note that the zero-knowledge functionality [6] over (moderately) hard-on-the-average languages is weakly one-way.

2. Let $f : X \times X \to \{0,1\}^*$ 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,[19] then $\mathcal{F}_f$ cannot be securely realized under specialized-simulator UC.

3. Let $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$ be a (deterministic) function and denote $f = (f_1, f_2)$ (where $f_i : X \times X \to \{0,1\}^*$ 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_1 or f_2 is privacy preserving, then $\mathcal{F}_f$ cannot be securely realized under specialized-simulator UC.[20] See Appendix A for a definition of privacy preserving functions.

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.

[17]The focus of [9] was to prove impossibility results for the UC definition. However, they do explicitly state which of their results also apply to the relaxed variant of UC in which the quantifiers are reversed (i.e., what we call "specialized-simulator UC").

[18]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.

[19]Formally, a function f depends on both inputs if there does not exist a function $g : X \to \{0,1\}^*$ 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.

[20]We note that in the FOCS 2003 proceedings version of this paper, it was stated that if f_1 or f_2 has an insecure minor, then $\mathcal{F}_f$ cannot be securely realized under specialized-simulator UC. This statement is incorrect (and was also not proven in [9]).

Impossibility results for 1-bounded general composition. Let Φ be the set of ideal functionalities described above, that cannot be securely realized under specialized-simulator UC. Applying Lemma 3.1 and Propositions 4 and 5 to the results of [9], we obtain the following corollaries:

Corollary 7 *Consider an adversarial model with adaptive or static corruptions. Then, the set of two-party functionalities Φ cannot be securely computed under concurrent 1-bounded general composition for arbitrary sets or a single set of parties.*

Corollary 7 follows directly from Proposition 4. We note that although Proposition 4 refers to static adversaries and a single set of parties, we obtain all four combinations (of adaptive/static adversaries and arbitrary/single sets of parties) due to the fact that security against adaptive adversaries implies security against static adversaries and security for arbitrary sets of parties implies security for a single set of parties. The next corollary relates to parallel general composition:

Corollary 8 *Consider an adversarial model with adaptive or static corruptions. Then, the set of two-party functionalities Φ cannot be securely computed under parallel 1-bounded general composition for arbitrary sets or a single set of parties.*

Corollary 8 follows from Proposition 5 and from the fact that the results of [9] also apply to specialized-simulator *synchronized* UC. Note that parallel composition is usually somewhat easier to obtain than concurrent composition. Nevertheless, Corollary 8 demonstrates that in the context of general composition, parallel composition is also "hard" to achieve.

Remark. The above impossibility results are very broad in that they rule out very large classes of *deterministic* functionalities. We note that impossibility for many probabilistic functionalities was also proven in [9]. However, these results hold only for the UC definition, and not for specialized-simulator UC. The feasibility of securely realizing probabilistic functionalities in the setting of concurrent general composition is therefore left open.

Open Questions

In this paper, we have considered two definition of security (UC security and specialized-simulator UC) and two notions of composition (concurrent general composition and concurrent $O(1)$-bounded general composition). Figure 1 shows the known relations between these notions (an arrow from one notion to another means that security under the first definition implies security under the second). The vertical arrows hold by triviality, the implication of UC security to concurrent general composition is due to [6], and the equivalence between specialized-simulator UC and concurrent $O(1)$-bounded general composition is stated here in Theorem 3.

$$\begin{tikzcd}
\text{UC security} \arrow[r] \arrow[d] & \text{concurrent general composition} \arrow[d] \\
\text{specialized−simulator UC} \arrow[r, leftrightarrow] & \text{concurrent O(1)−bounded general composition}
\end{tikzcd}$$

Figure 1: Known relations between notions

The main open questions left by this work are the implications that do not appear in Figure 1. There are actually three (non-disjoint) questions here: First, is UC security equivalent to concurrent (unbounded) general composition? Second, is UC security equivalent to specialized-simulator UC. Third, is concurrent general composition equivalent to concurrent $O(1)$-bounded general composition? Equivalence or separations can be demonstrated on the level of protocols (is every protocol that is secure by one notion also secure by the other) and on the level of feasibility (are there functionalities that can be securely realized by one notion and not by the other).

Another important question that is related to the above, is to broaden the known impossibility results for specialized-simulator UC, especially where impossibility is known for UC security. An important example of this is the case of probabilistic functionalities; the UC impossibility results of [9] for probabilistic functionalities do *not* hold for specialized-simulator UC.

Acknowledgements

We would like to thank Ran Canetti, Oded Goldreich and Hugo Krawczyk for many helpful discussions and 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] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

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

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

[6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.

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

[8] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires $\tilde{\Omega}(\log n)$ Rounds. In *33rd STOC*, pages 570–579, 2001.

[9] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universal Composition Without Set-Up Assumptions. In *EUROCRYPT'03,* Springer-Verlag (LNCS 2656), pages 68–86, 2003.

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

[11] R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO'03*, 2003.

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

[13] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In 30*th STOC*, pages 409–418, 1998.

[14] O. Goldreich. *Secure Multi-Party Computation.* Manuscript, version 1.4, 2002. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[15] O. Goldreich. Cryptography and Cryptographic Protocols (Survey). *PODC Jubilee Issue of Distributed Computing.*

[16] 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.

[17] 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.

[18] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. *Cryptology ePrint Archive*, Report #2002/040, `http://eprint.iacr.org/2002/040`. (An extended abstract appeared in the 16*th DISC,* Springer-Verlag (LNCS 2508), pages 17–32, 2002.)

[19] Y. Lindell. Bounded-Concurrent Secure Two-Party Computation Without Setup Assumptions. In 35*th STOC,* 2003.

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

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

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

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

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

A Privacy Preserving Functions

As mentioned in Section 4, privacy preserving functions cannot be securely realized under specialized-simulator UC, and therefore cannot be securely realized under concurrent, or even parallel, 1-bounded general composition. In this appendix, we define what it means for a function to be privacy preserving. We stress that this definition does *not* refer to *protocols*, where this usually means that the protocol does not reveal any more about the private inputs than the function itself. Rather, here we consider if a party's input is revealed by the *function itself*, or if it preserves the party's privacy.

Intuitively, we say that a party P_1's privacy is preserved if party P_2 cannot choose an input that will enable it to derive P_1's input. For example, let us take the maximum function for a given range, say $\{0, \ldots, n\}$. Then, party P_2 can always input 0 and then it will learn P_1's exact input. In contrast, the less-than function *is* privacy preserving 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).

Before formally defining privacy preserving functions, we define what it means for two inputs to be "equivalent": Let $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$ be a two-party function and denote $f = (f_1, f_2)$. Furthermore, let $X_1 \subseteq X$ be a set of inputs for P_1. We say that X_1 is a set of non-equivalent inputs if there exists a polynomial-size set of inputs $X_2 \subseteq X$ for P_2 such that for every $x_1, x_1' \in X_1$, there exists an $x_2 \in X_2$ such that $f_2(x_1, x_2) \neq f_2(x_1' x_2)$. If this does not hold, then x_1 can be used instead of x_1' and this makes no difference to P_2's output, as long as the inputs are chosen from X_1 and X_2. Observe that in the important special case that f has a *finite domain*, the set X_2 can be taken to be the entire set of inputs for P_2 (yielding a more "natural" requirement, where x_1 can always be used instead of x_1'). We now define privacy preserving functions:

Definition 9 (privacy preserving functions): *Let $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$ be a two-party function and denote $f = (f_1, f_2)$. We say that function f_2 is* privacy preserving for *P_1 if there exists a polynomial-size set of* non-equivalent *inputs $X_1 \subseteq X$ for P_1, so that for* every *input x_2 for P_2, there exist $x_1, x_1' \in X$ ($x_1 \neq x_1'$) such that $f_2(x_1, x_2) = f_2(x_1', x_2)$. Privacy preserving for P_2 is defined analogously.*

In order to appreciate why this definition is reasonable, consider the converse. That is, assume that there exists an input x_2 such that for every $x_1, x_1' \in X_1$, it holds that $f_2(x_1, x_2) \neq f_2(x_1', x_2)$. In this case, P_2 can set its own input to this x_2, and then it will always obtain the exact input used by P_1. In this sense, P_1's privacy is not at all preserved. (Of course, this is strictly true only if P_1's input is chosen from the set X_1. However, in the case that f has a finite domain, this is not a limitation; see below.) We require X_1 to be a set of non-equivalent inputs in order to rule out the case that privacy holds only because there is a pair of inputs x_1 and x_1' for which $f_2(x_1, x_2) = f_2(x_1', x_2)$ for all x_2. In such a case, x_1 and x_1' are essentially the same input and thus this does not express any "non-trivial" privacy preserving.

Note that the less-than function (also known as Yao's famous millionaires problem) preserves the privacy of P_1. Specifically, any set of three distinct values constitutes a set of non-equivalent inputs. Furthermore, for any input that P_2 chooses, at least two of P_1's potential inputs will be both greater than or both less than P_2's input. Thus, P_2 is unable to fully determine P_1's input, no matter what input it uses.

Functions over a finite domain. As mentioned above, the special case where f has a finite domain yields a simpler, and more natural, definition of privacy preserving functions. To further simplify the definition, let us assume that there are no equivalent inputs whatsoever. This is fine because the function can always be redefined over a smaller domain, where the lexicographically smallest input is always used in the case of equivalent inputs. (This can also be easily computed because the function has a finite domain.) In such a case, the definition is as follows:

Definition 10 (privacy preserving functions over a finite domain): *Let $f : X \times X \to \{0,1\}^* \times \{0,1\}^*$ be a two-party function with a finite domain and no equivalent inputs, and denote $f = (f_1, f_2)$. We say that function f_2 is* privacy preserving for *P_1 if for* every *input x_2 for P_2, there exist distinct inputs x_1 and x_1' for P_1 such that $f_2(x_1, x_2) = f_2(x_1', x_2)$. Privacy preserving for P_2 is defined analogously.*

As described above, if a function f is not privacy preserving for P_1, then P_2 can always learn P_1's exact input (by inputting the value x_2 for which $f_2(x_1, x_2) \neq f_2(x_1', x_2)$ for all x_1, x_1'). Thus, such a function preserves no privacy for P_1.