Comparing UC Security Variants

Oana Ciobotaru

Saarland University and Max Planck Institute for Computer Science
Saarbrücken, Germany
ociobota@mpi-inf.mpg.de

Abstract. In this work we investigate the relations among various security notions. More precisely, we present a separation result between two variants of UC security definition: 1-bit specialized simulator UC security and specialized simulator UC security. This solves an open question from [15] and comes in contrast with the well known equivalence result between 1-bit UC security and UC security. We also give a notion of weak security and we show that the induced weak security under 1-bounded concurrent general composition is equivalent to 1-bit specialized simulator UC security. As a consequence, we obtain that our notion of weak security and the notion of stand-alone security are not equivalent.

Keywords: weak security; composability; 1-bit specialized simulator UC security; time-lock puzzles

1 Introduction

Nowadays, more and more cryptographic protocols ran by honest users are part of complex systems which could potentially harbor just as complex and powerful security attacks. In order to ensure that none of these attacks become reality, protocol designers need cryptographic constructions tailored for different security guaranties. In the last decades, for this purpose many different security notions have been proposed, varying from stand-alone security [8] to general concurrent composability [15], universal composability [4] and even the game theoretic notion of strong universal implementation [12, 11]. One important aspect that helps better understanding these security notions is studying the relation among them. In this paper we investigate the relation among various security notions, from specialized simulator UC and 1-bit specialized simulator UC to stand-alone security and weak stand-alone security.

1.1 Related Work

The initial work [22] on general security definitions highlighted the need for a framework expressing security requirements in a formal way. The first formal definition of *secure computation* was introduced by [8]. The first approaches for formally defining security notions (e.g., zero-knowledge [10, 9]) have taken into account only the stand-alone model (i.e., when security of the protocol is considered with respect to its adversary, in isolation from any other copy of itself or from a different protocol). However, there are simple protocols, (e.g., the example from[7]) that fulfill stand-alone security, but are no longer secure under parallel or concurrent composition.

The work that introduces the study of protocol composition (which the authors call reducibility) is [16]. In [2] a security definition (expressed for the first time as a comparison with an ideal process) in the stand-alone model is provided and the corresponding sequential composition theorem. In [3] a general definition of security for evaluating a probabilistic function on the parties' inputs is given. It is shown that security is preserved under a subroutine substitution composition operation, which is a non-concurent version of universal composition (i.e., only a single instance of the protocol is active at any point in time). The framework of *universally composable security* or UC security [4] allows for specifying the requirements for any cryptographic task and within this framework protocols are guaranteed to maintain their security even in the presence of an unbounded number of arbitrary protocol instances that run concurrently in an adversarially controlled manner.

The notion of *specialized simulator UC* security (which differs from the notion of universal composability in that the order of quantifiers is changed) has been introduced in [15] and it was shown that this is equivalent to *general concurrent composability* when the protocol under consideration is composed with one instance of any possible protocol. Changing the order of quantifiers in the context of security definitions has been previously used in [6, 12, 11]. A more detailed review about the existing implication relations among different security notions can be found in section 4.

The topic of reactive functionalities is first addressed by [13]. The incipient work from [18] and later from [19, 17, 20] is centered around the notion of *reactive simulatability* which is similar to the notion of universal composability and has been developed in parallel with it. The framework addresses for the first time *concurrent* composition in a computational setting: it is shown that security is preserved when a single instance of a subroutine protocol is composed concurrently with the calling protocol. The framework has been extended in [1] to deal with the case where the number of parties and protocol instances depends on the security parameter. More about the differences between reactive simulatability and universal composability notions can be read in the related work section from [4].

More recently, [12, 11] define the game theoretic notion of universal protocol implementation and they show it is equivalent to a weak variant of stand-alone security, called precise secure computation. Their work started exploring the relations between game theoretic notions and cryptographic security definitions.

1.2 Contribution

We have a twofold contribution.

First, we present a separation result between two variants of UC security: 1-bit specialized simulator UC security and specialized simulator UC security. This solves an open question from [15] and comes in contrast with the well known equivalence result between 1-bit UC security and UC security [4].

Both variants of the UC security notion are obtained from the UC security definition by changing the order of quantifiers[1]. Thus, we continue the line of study started by [6, 15]. In order to obtain the separation, we first show that the 1-bit specialized simulator UC security is equivalent to a seemingly weaker version of security, namely weak specialized simulator UC security[2].

The proof technique used in our separation result is to employ a cryptographic tool called time-lock puzzles. Intuitively, a time-lock puzzle can be used for comparing the computational power of two different polynomially bounded Turing machines. In our case, we construct a protocol that uses time-lock puzzles and pseudo-random permutations to compare the computational power of the simulator and the distinguisher.

Second, our notion of weak security[3] induces a notion of weak security composed under concurrent general composition which we show is equivalent to 1-bit specialized simulator UC security. For this we build upon the proof technique from [15]. Together with our first result, this implies that weak security and stand-alone security are not equivalent.

1.3 Organization

This work is structured as follows: In section 2 we give a review of security notions, from stand-alone security to universal composability and concurrent general composition. In section 3 we prove our separation result between specialized simulator UC security and 1-bit specialized simulator UC security. In section 4 we present our equivalence relation between weak security under 1-bounded concurrent general composition and 1-bit specialized simulator UC security. In section 5 we conclude. In appendix A we give the full detailed models for UC security and security under general concurrent composition. In appendix B we present the postponed proofs from section 4.

2 Review of Security Notions

In this work we consider that all parties and adversaries run in polynomial time in the security parameter denoted by k and not in the length of their input. In this section we review two models of security under composition: concurrent general composition and universal composability. Both frameworks require the notion of (computational) indistinguishability given below.

[1] This means that in contrast to the UC security definition, the simulator may depend on the environment.

[2] This notion, additionally to having the simulator depend on the environment, also has the simulator depend on the distinguisher that compares the views of the environment from the real and the ideal world.

[3] The difference between stand-alone security and weak security is in the order of quantifiers. For stand-alone security, the simulator is universally quantified over all distinguishers and input distributions. As detailed in section 2, for our notion of weak security the simulator depends only on the distinguisher and not on the input distribution. This comes in contrast with [12], where the simulator for weak security depends on both distinguisher and input distribution.

Definition 1 (Computational Indistinguishability). *We say that two distribution ensembles* $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *are computationally indistinguishable and we write* $X \equiv Y$*, if for every probabilistic distinguisher* $\mathcal{D}$*, polynomial in* k *there exists a function* ϵ*, negligible in* k*, such that for every* $z \in \{0,1\}^*$

$$|(Pr(\mathcal{D}(X(k,z)) = 1) - (Pr(\mathcal{D}(Y(k,z)) = 1)| < \epsilon(k)$$

A variant of this definition, which we call *indistinguishability with respect to a given adversary* $\mathcal{D}$ and we denote by $\overset{\mathcal{D}}{\equiv}$, is analogous to the definition above, where "for every probabilistic distinguisher $\mathcal{D}$" is replaced with "for distinguisher $\mathcal{D}$". Such a definition will be used in relation with our notion of weak security.

2.1 Universal Composability

The standard method for defining security notions is by comparing a real world protocol execution to an ideal world process execution. In the real world execution, a protocol interacts with its adversary and possibly with other parties. In the ideal world execution, an idealized version of the protocol (called ideal functionality) interacts with an ideal world adversary (usually called simulator) and possibly with other parties. The ideal functionality is defined by the security requirements that we want our protocol to fulfill.

On an intuitive level, given an adversary, the purpose of the simulator is to mount an attack on the ideal functionality such that no (probabilistic polynomial time or PPT) distinguisher can tell apart the output of the interaction between the ideal functionality and the simulator and the output of the interaction between the protocol and its adversary. If for every adversary, a simulator exists such that the two outputs cannot be told apart by any distinguisher, then our initial protocol is as secure as the ideal functionality, with respect to what is called *t*he stand-alone model.

Definition 2 (Stand-alone Security). *Let* ρ *be a protocol and* $\mathcal{F}$ *an ideal functionality. We say* ρ securely implements $\mathcal{F}$ *if for every probabilistic polynomial-time real-model adversary* $\mathcal{A}$ *there exists a probabilistic polynomial-time ideal-model adversary* $\mathcal{S}$ *such that for every protocol input* x *and every adversary auxiliary input* z *with* $x, z \in \{0,1\}^{poly(n)}$*, where* k *is the security parameter:*

$$\{IDEAL^{\mathcal{F}}_{\mathcal{S}}(k,x,z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\rho,\mathcal{A}}(k,x,z)\}_{k\in\mathbb{N}}.$$

By $IDEAL^{\mathcal{F}}_{\mathcal{S}}(k,x,z)$ we denote the output of $\mathcal{F}$ and $\mathcal{S}$ after their interaction and $REAL_{\rho,\mathcal{A}}(k,x,z)$ denotes the output of the parties of ρ and adversary $\mathcal{A}$ after their interaction. If we allow the simulator to depend on the distinguisher, we obtain the weak stand-alone security notion.

There are examples [7] of protocols secure in the stand-alone model that do not remain secure even when two of its instances run concurrently. More stringent security definitions take into account that a protocol interacts not only with its adversary, but also with other (possibly polynomially many) protocols or with (polynomially many) copies of itself. This is intuitively captured by the universal composability (UC) security framework [4]. (Due to lack of space, we give below only high level intuition about the model and the relevant definitions. A detailed review is included in the appendix.)

The definition of universal composability follows the paradigm described above, however it introduces an additional adversarial entity which is called environment. The environment, usually denoted by $\mathcal{Z}$, is present in both the UC real world and UC ideal world. The environment represents everything that is external to the current execution of the real-world protocol or to the ideal functionality. Throughout the execution, both in the real and in the ideal world, the environment can provide inputs to parties running ρ or the ideal functionality $\mathcal{F}$ respectively, and to the adversary. These inputs can be a part of the auxiliary input of $\mathcal{Z}$ or can be adaptively chosen. Also $\mathcal{Z}$ receives all the output messages of the parties it interacts with and of the adversary. Moreover, the only interaction between the environment $\mathcal{Z}$ and the parties of ρ or $\mathcal{F}$ is when the environment sends the inputs and receives the outputs. Finally, at the end of the execution, the environment outputs all the messages it received. The environment is modeled as a PPT machine with auxiliary input. This auxiliary input captures the intuition that $\mathcal{Z}$ may learn some information from previous executions and it may also use it at any point later.

The main difference between the execution of UC real and UC ideal world, is that in the latter the ideal functionality cannot be directly accessed by the environment. Parties involved in the ideal execution give their inputs to the ideal functionality which computes some outputs and sends back these values. Since the ideal world parties perform no computation they are called the dummy parties for the ideal functionality. The ideal $\mathcal{F}$ together with its corresponding dummy parties represent an ideal process.

When the protocol execution ends, $\mathcal{Z}$ outputs its view of that execution. In the real world, his view contains messages that $\mathcal{Z}$ has received from the adversary $\mathcal{A}$ and outputs of all parties of ρ. This is denoted by $EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)$, where k is the security parameter and z is the auxiliary input to $\mathcal{Z}$. Similarly, in the ideal world execution, the environment $\mathcal{Z}$ outputs its view which contains all the messages received from $\mathcal{S}$ as well as all messages that the dummy parties of $\mathcal{F}$ output to $\mathcal{Z}$. This is denoted by $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$. We are now ready to define UC security:

Definition 3 (UC Security). *Let ρ be a PPT protocol and let $\mathcal{F}$ be an ideal functionality. We say that ρ UC emulates $\mathcal{F}$ (or ρ is as secure as $\mathcal{F}$ with respect to UC security) if for every PPT adversary $\mathcal{A}$ there is a PPT simulator $\mathcal{S}$ such that for every PPT distinguisher $\mathcal{Z}$ and for every distribution of auxiliary input $z \in \{0,1\}^*$, the two families of random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ and $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ are computationally indistinguishable.*

In the following we also use a relaxed version of this definition, where the order of quantifiers between the environment and the ideal-world simulator is reversed [15].

Definition 4 (Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for every input $z \in \{0,1\}^*$, we have:*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

It had been shown [14] that the two notions defined above are not equivalent. In the above definition, the output of the environment is considered to be a string of arbitrary length. If the only change we make to the above definition is to consider environments that have a 1-bit output, we obtain the notion of *1*-bit specialized simulator UC security. It has been an open problem [15] whether considering only environments with one bit output would produce an equivalent definition. In this work we show this is not the case. If in the specialized simulator UC definition we let the simulator also depend on the distinguisher (i.e., the only machine to establish whether the output of the executions in the real UC world and in the ideal UC world cannot be told apart), then we obtain the notion of *w*eak specialized simulator UC security. Both specialized simulator UC variants are defined in full detail in the appendix.

In the revised version of [4] there is an extension of the UC model we reviewed above. This extension mainly considers that PPT machines run in time polynomial in both the security parameter and the length of the input. While the extended model is seemingly more expressive in terms of adversarial attacks, it does not allow for fine grained separation between security notions (e.g., the separation result from [14] does not hold in the extended UC model). Another reason for choosing the original model is that, as it will be detailed in section 4, most of the UC results have been obtained in this model.

2.2 Weak Security under 1- bounded Concurrent General Composition

Similarly to the above security concepts, the notion of security under concurrent general composition [15] is defined using the real-ideal world paradigm. (Full details about this security model are postponed to the appendix.)

In this model, an external and arbitrary protocol π gives inputs to and collects outputs from an "internal protocol" that can be a real-world protocol or an ideal functionality. We denote by ρ the real-world protocol interacting with π and by $\mathcal{F}$ the ideal functionality. Protocol π may call multiple instances of the protocol it interacts with as long as all of them run independently and all its messages may be sent in a concurrent manner.

The computation in the ideal world is performed among the parties of π and an ideal functionality $\mathcal{F}$. Protocol π is providing $\mathcal{F}$ with inputs and after performing necessary computations, $\mathcal{F}$ sends the results to parties of π. The messages between π and $\mathcal{F}$ are ideally secure, so the ideal adversary (or simulator) can neither read nor change them.[4]

The ideal-world honest parties follow the instructions of π and output the value prescribed by π. The corrupted parties output a special corrupted symbol and additionally the adversary may output an arbitrary image of its view. Let z be the auxiliary input for the ideal-world adversary $\mathcal{S}$ and let $\bar{x} = (x_1, ..., x_m)$ be the inputs vector for parties of π. The outcome of the computation of π with $\mathcal{F}$ in the ideal world is defined by the output of all parties of π and $\mathcal{S}$ and is denoted by $\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k, \bar{x}, z)\}_{k \in \mathbb{N}}$. We choose this notation in order to make it easier to differentiate between the ideal world in the UC definition and the ideal world in the general concurrent composition definition. Moreover, this is not unjustified, as in the latter case the messages that occur

[4] This comes in contrast with the standard definition of UC ideal protocol execution, where it is not enforced that the channels between the trusted parties and the rest of the participants are ideally secure.

in the ideal world correspond both to communication among real world parties of π and also between parties of π and the ideal functionality.

The computation in the real world follows the same rules as the computation in the ideal world, only that this time there is no trusted party. Instead, each party of π has an ITM that works as the specification of ρ for that party. Thus, all messages that a party of π sends to the ideal functionality in the ideal world are now written on the input tape of its designated ITM. These ITMs communicate with each other in the same manner as specified for the parties of ρ. After the computation is performed, the results are output by these ITMs to their corresponding parties of π.

The honest real-world parties follow the instructions of π and their corresponding ITM and in the end they output on their output tape the value prescribed by π. The corrupted parties output a special corrupted symbol and additionally the real-world adversary $\mathcal{A}$ may output an arbitrary image of its view. The outcome of the computation of π with ρ in the real world is defined by the output of all parties and $\mathcal{A}$ and is denoted by $\{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}$.

We are now ready to state the definition of security under concurrent general composition [15], with the additional flavor of weak security. This means that we allow the simulator to depend on the distinguisher and additionally, this distinguisher is the only entity supposed to tell apart the real world execution from the ideal world execution.

Definition 5 (Weak Security under Concurrent General Composition). *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* computes $\mathcal{F}$ under concurrent general composition with weak security *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideals calls to $\mathcal{F}$, for every probabilistic polynomial-time real-model adversary $\mathcal{A}$ for π^ρ and for every probabilistic polynomial-time distinguisher $\mathcal{D}$, there exists a probabilistic polynomial-time ideal-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$:*

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

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

3 Specialized Simulator UC Variants

Our main result in this section shows the separation between the notions of specialized simulator UC and 1-bit specialized simulator UC. This answers an existing open problem from [15] and furthermore clarifies the relations among different (weak) security notions.

3.1 On 1-bit Specialized Simulator UC

We start by showing that 1-bit specialized simulator UC (1-bit SSUC) is equivalent to weak specialized simulator UC (weak SSUC). This will give us a simpler alternative security notion that we can further work with.

Lemma 1 (Equivalence between 1-bit SSUC and weak SSUC). *A protocol fulfills the 1-bit specialized simulator UC security if and only if it fulfills the weak specialized simulator UC security.*

Proof. Let protocol ρ and ideal functionality $\mathcal{F}$ be such that ρ is as secure as $\mathcal{F}$ with respect to 1-bit specialized simulator UC. We show this implies ρ as secure as $\mathcal{F}$ with respect to weak specialized simulator UC security. Given a triple $(\mathcal{A}, \mathcal{Z}, \mathcal{D}^*)$ consisting of adversary, environment and distinguisher we have to provide a simulator $\mathcal{S}$ such that for every auxiliary input[5] z the following holds:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}^*}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}. \tag{1}$$

Given $\mathcal{Z}$ and $\mathcal{D}^*$, we can construct a 1-bit output environment $\mathcal{Z}^{\mathcal{D}^*}$ in the following way: $\mathcal{Z}^{\mathcal{D}^*}$ internally runs a copy of $\mathcal{Z}$. When internal $\mathcal{Z}$ writes on its output tape, this is forwarded by $\mathcal{Z}^{\mathcal{D}^*}$ to an internal copy of $\mathcal{D}^*$. The output of $\mathcal{D}^*$ becomes the output of $\mathcal{Z}^{\mathcal{D}^*}$. Due to the hypothesis, there exist $\mathcal{S}$ such that for every auxiliary input z and for every distinguisher $\mathcal{D}$ we have:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}}.$$

In particular:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}_{ind}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}^{\mathcal{D}^*}}(k,z)\}_{k\in\mathbb{N}},$$

where $\mathcal{D}_{ind}$ is the distinguisher that outputs whatever $\mathcal{D}^*$ outputs. As the simulator $\mathcal{S}$ can be used without modification in an interaction with $\mathcal{F}$ and the environment[6] $\mathcal{Z}$, the last relation is equivalent to (1). We conclude that ρ is as secure as $\mathcal{F}$ with respect to weak specialized simulator UC security.

The implication in the opposite direction is proven as follows. Given a pair $(\mathcal{A}, \mathcal{Z}_{1-bit})$ consisting of adversary and 1-bit output environment, we need to construct a simulator $\mathcal{S}$ such that for every auxiliary input z and for every distinguisher $\mathcal{D}$, we have:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}}.$$

Given a 1-bit output environment $\mathcal{Z}_{1-bit}$, we can uniquely decompose it into an environment $\mathcal{Z}$ (that outputs its entire view) and a distinguisher $\mathcal{D}^*$ (that given the view of $\mathcal{Z}$ outputs what $\mathcal{Z}_{1-bit}$ outputs). According to the definition of weak specialized simulator UC security, for $\mathcal{A}$, $\mathcal{Z}$, $\mathcal{D}^*$ there exists a simulator $\mathcal{S}$ such that for every auxiliary input z we have:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}^*}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

As $\mathcal{D}^*$ has binary output (i.e., thus finite output), the above equation implies the two random variables $\{\mathcal{D}^*(EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z))\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ and

[5] Here and in the following "for every auxiliary input z" should be read as "for every distribution of auxiliary input z for $\mathcal{Z}$".

[6] Indeed, by construction $\mathcal{Z}^{\mathcal{D}^*}$ does not interact with an adversarial party (i.e., $\mathcal{S}$ or $\mathcal{A}$) after the simulation of internal $\mathcal{Z}$ is over.

$\{\mathcal{D}^*(EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z))\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ are statistically close. Hence, for any computationally bounded distinguisher $\mathcal{D}$ and for any auxiliary input z the two random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}}$ and $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}_{1-bit}}(k,z)\}_{k\in\mathbb{N}}$ are indistinguishable and this concludes the proof.

3.2 Separation Result

Next we separate the notions of weak specialized simulator UC and specialized simulator UC. For this we use a cryptographic tool called time-lock puzzles, originally introduced in [21].

Definition 6 (Time-lock puzzles). *A PPT algorithm $\mathcal{G}$ (problem generator) together with a PPT algorithm $\mathcal{V}$ (solution verifier) represent a* time-lock puzzle *if the following holds:*
-sufficiently hard puzzles: for every PPT algorithm B and for every $e \in \mathbb{N}$, there is some $f \in \mathbb{N}$ such that

$$\sup_{t\geq k^f,|h|\leq k^e} Pr[(q,a) \leftarrow \mathcal{G}(1^k,t) : \mathcal{V}(1^k,a,B(1^k,q,h)) = 1] \tag{2}$$

is negligible in k.
-sufficiently good solvers: there is some $b \in \mathbb{N}$ such that for every $d \in \mathbb{N}$ there is a PPT algorithm C such that

$$\min_{t\leq k^d} Pr[(q,a) \leftarrow \mathcal{G}(1^k,t); c \leftarrow C(1^k,q) : \mathcal{V}(1^k,a,c) = 1 \wedge |c| \leq k^b] \tag{3}$$

is overwhelming in k.

Intuitively, a time-lock puzzle is a cryptographic tool used for proving the computational power of a PPT machine. $\mathcal{G}(1^k,t)$ generates puzzles of hardness t and $\mathcal{V}(1^k,a,c)$ verifies that c is a valid solution as specified by a. The first requirement is that B cannot solve any puzzle of hardness t, with $t \geq k^f$, for some f depending on B, with more than negligible probability. The algorithm B may have an auxiliary input. This ensures that even puzzles generated using hardness t chosen by B together with a trap-door like auxiliary information (of polynomial length), do not provide B with more help in solving the puzzle.

The second requirement is that for any polynomial hardness value there exist an algorithm that can solve any puzzle of that hardness. It is important that the solution for any puzzle can be expressed as a string of length bounded above by a fixed polynomial.

As promoted by [21] and later by [14], a candidate family for time-lock puzzles is presented next. A puzzle of hardness t consists of the task to compute $2^{2^{t'}} \mod n$ where $t' := \min(t, 2^k)$ and $n = p \cdot q$ is a randomly chosen Blum integer. Thus, $\mathcal{G}(1^k,t) = ((n, \min\{t,2^k\}), (p,q,\min\{t,2^k\}))$, where n is a k-bit Blum integer with factorization $n = p \cdot q$, and $\mathcal{V}(1^k,(p,q,t'),c) = 1$ if and only if $c \equiv 2^{2^{t'}} \mod p \cdot q$. Both solving the puzzle and verifying the solution can be efficiently done if p and q are known. An important property that we use in the following is that any such puzzle proposed above has a unique solution.

We employ time-lock puzzles to obtain the following result:

Lemma 2 (Weak SSUC Does Not Imply SSUC).

Assume that families of pseudo-random permutations and time-lock puzzles exist. Then there are protocols that fulfill weak specialized simulator UC security but do not fulfill specialized simulator UC security.

Proof. Let $(\pi, \mathcal{F})$ be a pair of protocol and ideal functionality as defined below. The only input the ideal functionality $\mathcal{F}$ requires is the security parameter 1^k. Then $\mathcal{F}$ sends a message to the adversary (i.e. ideal simulator $\mathcal{S}$) asking for its computational hardness. Using the reply value t' from $\mathcal{S}$ (which is truncated by $\mathcal{F}$ to maximum k bits), the ideal functionality invokes $Gen(1^k, t') \rightarrow (q', a')$ to generate a time-lock puzzle q' of hardness t', whose solution should verify the property a'. The puzzle q' is sent to $\mathcal{S}$ which replies with v'. Finally, $\mathcal{F}$ checks whether v' verifies the property a'. In case a' does not hold, $\mathcal{F}$ stops without outputting any message to the environment. Otherwise, for every value $i \in \{1, \ldots, k\}$, $\mathcal{F}$ generates a puzzle q_i of hardness $t_i = 2^i$. Let j be such that $2^j \leq t' < 2^{j+1}$, so $j \in \{1, \ldots, k\}$. For the puzzle q_j, $\mathcal{F}$ computes the solution v_j. $\mathcal{F}$ can efficiently compute this solution as it knows the additional information a_j. Additionally, $\mathcal{F}$ chooses r uniformly at random from $\{0,1\}^k$. The output of $\mathcal{F}$ to the environment is the tuple $(q_1, \ldots, q_k, g_{v_j+r}(0), r)$, where $\{g_i\}_{i \in \{0,1\}^k}$ with $g_i : \{0,1\}^k \rightarrow \{0,1\}^k$ is a family of pseudo-random permutations. Note that the addition between v_j and r considers both v_j and r as values in $\{0,1\}^k$ and the addition operation is performed mod 2^k. However, for ease of notation from now on we do not add this explicitly in the equtions.

For each hardness t', we call $P(t')$ the distribution of the view of $\mathcal{Z}$ when interacting in the ideal world.

The real world protocol π, is defined similarly to $\mathcal{F}$, the only difference is the final output: π outputs to $\mathcal{Z}$ a tuple $(q_1, \ldots, q_k, r_1, r_2)$, with r_1, r_2 randomly chosen from $\{0,1\}^k$. For each hardness t used by the adversary $\mathcal{A}$ when interacting with $\mathcal{Z}$, we call $R(t)$ the distribution of the view of $\mathcal{Z}$ when interacting in the real world.

The proof has two steps. First, we show that π is as secure as $\mathcal{F}$ with respect to weak specialized simulator UC security. Let $\mathcal{D}$ be a distinguisher of hardness $t_{\mathcal{D}}$ (i.e., it can solve puzzles of hardness less or equal to $t_{\mathcal{D}}$ with overwhelming probability but it cannot solve puzzles of hardness greater than $t_{\mathcal{D}}$ with more than negligible probability) and an adversary $\mathcal{A}$ of hardness $t_{\mathcal{A}}$. Let j be the minimum value such that $2^j > \max(t_{\mathcal{D}}, t_{\mathcal{A}})$. We only need to require that the simulator $\mathcal{S}$ has hardness t' such that $t' \geq 2^j$. This will make the two distributions $R(t')$ and $P(t)$ indistinguishable to $\mathcal{D}$.

The intuition is that in the ideal world $\mathcal{D}$ would have to solve a puzzle with hardness larger than $t_{\mathcal{D}}$. So, to $\mathcal{D}$, the solution for such a puzzle looks just like a random value, which is actually what the protocol π outputs to the environment.

More formally, let $(\mathcal{A}, \mathcal{Z}, \mathcal{D})$ be a triple of real world adversary, environment and distinguisher and let 1^k be the security parameter. Then, let e be such that the length of the messages sent by $\mathcal{Z}$ to $\mathcal{D}$ is bounded above by k^e. From (2), there exists $f_e^{\mathcal{D}}$ such that:

$$\sup_{t \geq k^{f_e^{\mathcal{D}}}, |h| \leq k^e} Pr[(q', a') \leftarrow \mathcal{G}(1^k, t') : \mathcal{V}(1^k, a', D(1^k, q', h)) = 1]$$

is negligible in k. This intuitively means that $\mathcal{D}$ can solve puzzles of hardness larger than $k^{f_e^{\mathcal{D}}}$ only with negligible probability. Given $\mathcal{A}$, in an analogue way we define $k^{f_e^{\mathcal{A}}}$.

With the notation used in the description of π and $\mathcal{F}$, it means that $t_{\mathcal{D}} = k^{f_e^{\mathcal{D}}}$ and $t_{\mathcal{A}} = k^{f_e^{\mathcal{A}}}$.

We construct $\mathcal{S}$ such that there exists a negligible function ϵ and k_0 such that for every $k \geq k_0$ and for every distribution of auxiliary input z we have:

$$|(Pr(\mathcal{D}(EXEC_{\mathcal{A},\pi,\mathcal{Z}}(k,z)) = 1) - (Pr(\mathcal{D}(EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)) = 1)| < \epsilon(k). \qquad (4)$$

We take k_0 such that for every $k \geq k_0$, it holds that $\max(t_{\mathcal{A}}, t_{\mathcal{D}}) < 2^k$.

For a given $t_{\mathcal{A}}$ and $t_{\mathcal{D}}$ and for j defined as above, let f' be such that $2^j \leq k^{f'} \leq 2^k$. Let $\mathcal{S}$ be the simulator of hardness $k^{f'}$ that as first reply to $\mathcal{F}$ sends $t' := k^{f'}$. According to (3), for $d := f'$ there exists $C_{f'}$ such that

$$Pr[(q', a') \leftarrow \mathcal{G}(1^k, k^{f'}); v' \leftarrow C_{f'}(1^k, q') : \mathcal{V}(1^k, a', v') = 1 \wedge |v'| \leq k^b]$$

is overwhelming in k. When $\mathcal{F}$ sends a puzzle q' to $\mathcal{S}$, the simulator invokes $C_{f'}$ for $(1^k, q')$ and sends to $\mathcal{F}$ the output v' of $C_{f'}$. Internally, $\mathcal{S}$ simulates the adversary $\mathcal{A}$ and emulates the messages that the adversary would receive from $\mathcal{Z}$ and π as follows: When $\mathcal{F}$ requires the value of the computational hardness from $\mathcal{S}$, then $\mathcal{S}$ acts as π and requires the computational hardness from simulated $\mathcal{A}$. When $\mathcal{S}$ receives t from $\mathcal{A}$, then it invokes $Gen(1^k, t)$, obtaining output (q, a) and forwards to simulated $\mathcal{A}$ the puzzle q. Moreover, any message that internal $\mathcal{A}$ wants to send to the environment, $\mathcal{S}$ forwards it to $\mathcal{Z}$. Any message for $\mathcal{A}$ coming from $\mathcal{Z}$ is immediately forwarded by $\mathcal{S}$ to the internally simulated adversary. This completes the construction of $\mathcal{S}$.

By construction, $\mathcal{S}$ solves the puzzle sent by $\mathcal{F}$ with overwhelming probability and hence the output of $\mathcal{F}$ to $\mathcal{Z}$ is $(q_1, \ldots, q_k, g_{v_j+r}(0), r)$ with the same probability. The view of $\mathcal{Z}$ in the real world is $(1^k, t, q, v, (q_1, \ldots, q_k, r_1, r_2))$ and the view of $\mathcal{Z}$ in the ideal world[7] is $(1^k, t, q, v, (q_1, \ldots, q_k, g_{v_j+r}(0), r))$.

It remains to be shown that for the family of pseudo-random permutations as defined above, the distributions $R(t)$ and $P(t')$ are indistinguishable for $\mathcal{D}$. The first observation is that for r chosen uniformly at random from $\{0,1\}^k$, the sum $v_j + r$ is also uniformly distributed in $\{0,1\}^k$. The second observation is that the simulator $\mathcal{S}$ is constructed such that the distinguisher $\mathcal{D}$ can solve puzzle q_j only with negligible probability and hence obtain the unique solution v_j with the same probability. Putting these two observations together, we obtain that if $\mathcal{D}$ cannot compute the solution v_j, then due to the pseudo-random permutation assumption, the distribution of $g_{v_j+r}(0)$ is indistinguishable from the distribution of a randomly chosen permutation g_i evaluated in 0. But by definition, this is the uniform distribution on $\{0,1\}^k$, so it equals the distribution of r_1. It is now trivial to see that the views of $\mathcal{Z}$ in the two worlds are indistinguishable for $\mathcal{D}$, and this concludes the first part of the proof.

Second, we prove that π is not as secure as $\mathcal{F}$ with respect to specialized simulator UC security. Intuitively, for every hardness $t_{\mathcal{S}}$ (polynomial in the security parameter k) of a simulator machine $\mathcal{S}$, there exists a distinguisher $\mathcal{D}_{\mathcal{S}}$ such that for every $t \leq t_{\mathcal{S}}$, $\mathcal{D}_{\mathcal{S}}$

[7] One may argue of course that the view of $\mathcal{Z}$ may or may not contain the values t, q, v, depending on the adversary $\mathcal{A}$. Also, additionally to the view(s) stated above, the environment could output the interaction that it has with $\mathcal{A}$ besides messages t, q, v. However, for the analysis of this proof, the views considered above are the worst case scenario that would allow a distinguisher to tell apart the two worlds.

can solve puzzles of hardness t. As we will see next, $\mathcal{D}$ uses this property to distinguish with non-negligible probability between the environment's output distribution in the real and in the ideal world. The intuition is that $\mathcal{D}_{\mathcal{S}}$ solves one by one the puzzles in the output of $\mathcal{F}$ and for each solution evaluates the corresponding permutation in the value 0; as each puzzle has a unique solution, the result of this computation with respect to puzzle q_j equals the one before last value in the output of $\mathcal{F}$ to $\mathcal{Z}$ with overwhelming probability but equals the one before last value in the output of π only with negligible probability.

Formally, let $\mathcal{A}$ be the real world adversary that can solve puzzles of hardness $t_{\mathcal{A}}$ such that when receiving its input from the environment, it replies to π with $t_{\mathcal{A}}$ and the corresponding correct solution for the puzzle received. Let $\mathcal{Z}$ be the environment that just sends the security parameter to all parties (i.e., including the adversarial parties), receives their outputs and then outputs as view the messages received from the honest parties (i.e., protocol π in the real world or $\mathcal{F}$ in the ideal world). For every simulator $\mathcal{S}$, we show that there exists a distinguisher $\mathcal{D}_{\mathcal{S}}$ and a distribution for the auxiliary input z such that:

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}_{\mathcal{S}}}{\not\equiv} \{EXEC_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

Given $\mathcal{S}$ of hardness $t_{\mathcal{S}}$, we choose $\mathcal{D}_{\mathcal{S}}$ such that it can solve puzzles of hardness at leas $t_{\mathcal{D}} = max(t_{\mathcal{S}}, t_{\mathcal{A}})$ with overwhelming probability in k. Such a $\mathcal{D}_{\mathcal{S}}$ exists according to (3). Additionally, after receiving the view of $\mathcal{Z}$, $\mathcal{D}_{\mathcal{S}}$ solves one by one each puzzle q_i included in that view that has associated hardness $t_i \leq t_{\mathcal{D}}$ and it obtains each time the corresponding correct and unique solution v_i with overwhelming probability. Then $\mathcal{D}_{\mathcal{S}}$ evaluates g_{v_i+r} in 0. Lets call m the one before last string in the output of the honest party (i.e., $\mathcal{F}$ or π) to $\mathcal{Z}$[8]. Next, $\mathcal{D}_{\mathcal{S}}$ checks if $m = g_{v_i+r}(0)$ for any i as defined above. If this check succeeds once, $\mathcal{D}$ outputs 1, otherwise it outputs 0. The output 1 implies with overwhelming probability that m is a part of the view of $\mathcal{Z}$ from the ideal world and the output 0 implies with overwhelming probability that m is a part of the view of $\mathcal{Z}$ from the real world.

Indeed, if m is part of the view of the real world, then according to the definition of π, m is a random string in $\{0,1\}^k$. For uniformly random r, due to the definition of pseudo-random permutations, for every i, $g_{v_i+r}(0)$ is uniformly distributed in $\{0,1\}^k$. In this case, the probability for $m = g_{v_i+r}(0)$ to hold is negligible, which is equivalent to $\mathcal{D}_{\mathcal{S}}$ outputting 1 with negligible probability when the view of $\mathcal{Z}$ is from the real world.

Similarly, if m is part of the view of $\mathcal{Z}$ in the ideal world, then there exists j such that $g_{v_j+r}(0)$ and m are different only with negligible probability. This implies $\mathcal{D}_{\mathcal{S}}$ outputs 0 with negligible probability when the view of $\mathcal{Z}$ is from the ideal world and this concludes the proof.

We are now ready to conclude that 1-bit specialized simulator UC security and specialized simulator UC security are not equivalent notions. By putting together the results from lemma 1 and from lemma 2 we obtain:

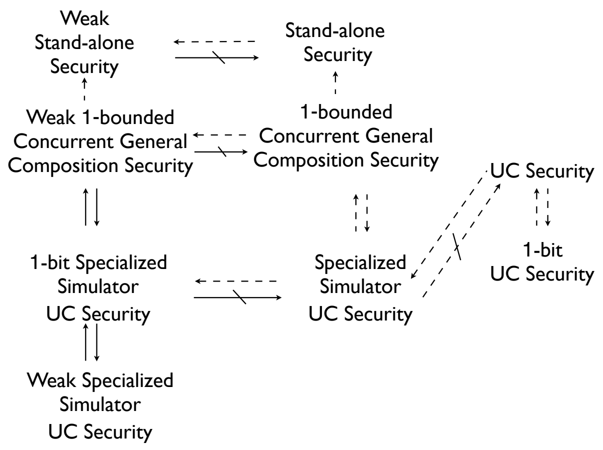
Theorem 1 (1-bit SSUC and SSUC Not Equivalent). *Assume time-lock puzzles and pseudo-random permutations exist. Then there are protocols secure with respect to*

[8] Due to the definition of $\mathcal{Z}$, the string m is also a part of the output of the environment.

1-bit specialized simulator UC security which are not secure with respect to specialized simulator UC security.

4 Equivalence of Security Notions

Fig. 1. Implication Relations among Computational Security Concepts


Weak Stand-alone Security
Stand-alone Security
Weak 1-bounded Concurrent General Composition Security
1-bounded Concurrent General Composition Security
UC Security
1-bit Specialized Simulator UC Security
Specialized Simulator UC Security
1-bit UC Security
Weak Specialized Simulator UC Security

Implication relations among various security notions with respect to computational security are depicted in figure 1. The continuous arrows depict the relations that we prove in this paper, the other relations have been previously known.

In [5] it has been shown that stand-alone security does not imply specialized simulator UC security. The implication in the opposite direction holds trivially. Similarly, it is trivial to see that universal composability implies specialized simulator UC security. In [14] it has been shown that specialized simulator UC security does not imply UC security. It is also a known fact that specialized simulator UC security is equivalent to security under 1-bounded concurrent general composition [15].

Our goal in this section is to prove that weak security under 1-bounded concurrent general composition implies 1-bit specialized simulator UC security. A similar proof technique has been used in [15], however, as it will be detailed below, our proof requires more technicalities.

More formally, the result we show is the following:

Theorem 2 (Equivalence between Weak Security under 1-bounded Concurrent General Composition and 1-bit Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We have that ρ implements $\mathcal{F}$ under 1-bounded concurrent general composition with weak security, if and only if ρ securely computes $\mathcal{F}$ under 1-bit specialized simulator UC security.*

The corresponding proof can be found in the appendix.

4.1 On Other Weak Security Notions

We show that the approach taken in theorem 2 is not an overkill. Indeed, there are protocols that are secure with respect to the weak security definition but they are not secure anymore in the standard stand-alone model.

Lemma 3 (Weak Security Does Not Imply Stand-alone Security). *If time-lock puzzles and pseudo-random permutations exist, then there are protocols that fulfill the weak security notion, but do not fulfill the stand-alone security notion.*

Proof. From theorem 2, weak security under 1-bounded concurrent general composition is equivalent to 1-bit specialized simulator UC. As shown in [15], stand-alone security under 1-bounded concurrent general composition is equivalent to specialized simulator UC. According to theorem 1, the two UC variants are not equivalent. This implies weak security and stand-alone security are also not equivalent[9].

5 Conclusions

In this work we have shown that two variants of the UC security definition where the order of quantifiers is reversed, namely 1-bit specialized simulator UC security and specialized simulator UC security are not equivalent. This comes in contrast to the well known result that UC security and 1-bit UC security are equivalent. We also show that weak security under concurrent general composition is equivalent to 1-bit specialized simulator UC. These results combined imply that weak security and stand-alone security are not equivalent.

Acknowledgements

I would like to thank Dominique Unruh for pointing out to me the open problem from [15] and for insightful discussions.

[9] One may wonder if the equivalence result between UC security and specialized simulator UC security that is known to hold in the extended UC model does not hinder the correctness of this result. However, this is not the case. On one hand, in the extended UC model, specialized simulator UC security and UC security are equivalent. Combining this with the well known result of equivalence between UC security and 1-bit UC security, we obtain that in the extended UC model, specialized simulator UC security and 1-bit specialized UC security are equivalent. This equivalence should not look surprising, as it is obtained in a more "permissive" adversarial UC model. On the other hand, the results obtained in this work show that there is at least one composition operation under which weak security and stand-alone security are not equivalent.

References

1. Backes, M., Pfitzmann, B., Waidner, M.: A general composition theorem for secure reactive systems. In: TCC. pp. 336–354 (2004)
2. Beaver, D.: Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. J. Cryptology 4(2) (1991)
3. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology 13(1), 143–202 (2000)
4. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science. pp. 136–145 (2001), full version on Cryptology ePrint Archive, Report 2000/067
5. Canetti, R., Fischlin, M.: Universally composable commitments. In: Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference. pp. 19–40 (2001)
6. Dwork, C., Naor, M., Reingold, O., Stockmeyer, L.J.: Magic functions. J. ACM 50(6), 852–921 (2003)
7. Feige, U.: Alternative Models For Zero Knowledge Interactive Proofs. Ph.D. thesis, Weizmann Institute of Science (1992)
8. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: FOCS. pp. 174–187 (1986)
9. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology 7(1), 1–32 (1994)
10. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1), 186–208 (1989)
11. Halpern, J.Y., Pass, R.: A computational game-theoretic framework for cryptography. unpublished manuscript (2010)
12. Halpern, J.Y., Pass, R.: Game theory with costly computation: Formulation and application to protocol security. In: Innovations in Computer Science (ICS 2010). pp. 120–142 (2010)
13. Hirt, M., Maurer, U.: Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing. pp. 25–34. PODC '97 (1997)
14. Hofheinz, D., Unruh, D.: Comparing two notions of simulatability. In: Theory of Cryptography, Proceedings of TCC 2005. pp. 89–103 (2005)
15. Lindell, Y.: General composition and universal composability in secure multi-party computation. In: FOCS. pp. 394–403 (2003)
16. Micali, S., Rogaway, P.: Secure computation (abstract). In: CRYPTO. pp. 392–404 (1991)
17. Pfitzmann, B., Schunter, M., Waidner, M.: Cryptographic security of reactive systems. Electr. Notes Theor. Comput. Sci. 32 (2000)
18. Pfitzmann, B., Waidner, M.: A general framework for formal notions of secure systems. http://www.semper.org/sirene/lit. (1994)
19. Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: CCS '00: Proceedings of the 7th ACM conference on Computer and communications security. pp. 245–254. ACM (2000)
20. Pfitzmann, B., Waidner, M.: A model for asynchronous reactive systems and its application to secure message transmission. In: IEEE Symposium on Security and Privacy. pp. 184– (2001)
21. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep., Massachusetts Institute of Technology (1996)
22. Yao, A.C.: Theory and application of trapdoor functions. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS '82). pp. 80–91 (1982)

A Related Security Definitions

We give below a variant for the computational indistinguishability definition.

Definition 7 (Indistinguishability with respect to a Given Distinguisher). *We say that two distribution ensembles* $\{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *are computationally indistinguishable with respect to a given probabilistic polynomial time distinguisher* $\mathcal{D}$ *and we write* $X \overset{\mathcal{D}}{\equiv} Y$*, if there exists a function* ϵ*, negligible in* k *and* k_0 *such that*

$$|(Pr(\mathcal{D}(X(k,z)) = 1) - (Pr(\mathcal{D}(Y(k,z)) = 1)| < \epsilon(k),$$

for every $k \geq k_0$*.*

A.1 Review of UC Model

There are examples [7] of protocols secure in the stand-alone model do not remain secure even when two of its instances run concurrently. More stringent security definitions take into account that a protocol interacts not only with its adversary, but also with other (possibly polynomially many) protocols or even (polynomially many) copies of itself. This is intuitively captured by the universal composability (UC) security framework [4].

In this case, the exterior world with respect to a given protocol is formalized by the notion of environment. Intuitively, the environment for a protocol contains all the other protocols, systems or users, together with their own adversaries, that may or may not interact with the considered protocol. It is important to note that the adversary for the protocol is not considered to be a part of the environment, but it could be controlled by the environment.

In order to determine whether a protocol securely implements a given task, first we define the ideal process for carrying out that task. Intuitively, in an ideal process for a given task, all parties give their inputs directly to the *ideal functionality for that task* which can be regarded as a formal specification of the security requirement of the task. According the universal composability security definition, a protocol securely implements a task if any damage that can be caused by an adversary while interacting with the protocol and the environment, can also be caused by an adversary interacting with the ideal process for that task and the environment. Intuitively, the entity assessing the amount of damage is the environment. Since there is no damage we can cause to the ideal functionality, the protocol considered must also be secure. We say that the protocol runs in a real-world model and the ideal functionality runs in the ideal-world model.

Real-world Protocols More formally, let ρ be a cryptographic protocol. The real-world model for the execution of protocol ρ contains the following interactive Turing machines (ITMs): an ITM $\mathcal{Z}$ called the environment, a set of ITMs representing the parties running the protocol ρ and an adversary ITM $\mathcal{A}$. We now have a more detailed look at each of these ITMs.

The environment $\mathcal{Z}$ represents everything that is external to the current execution of ρ and it is modeled as an ITM with auxiliary input. Throughout the course of the protocol execution, the environment can provide inputs to parties running ρ and to the adversary. These inputs can be a part of the auxiliary input of $\mathcal{Z}$ or can be adaptively chosen by the environment. Also $\mathcal{Z}$ receives all the outputs that are generated by the parties and the adversary. The only interaction between the environment $\mathcal{Z}$ and the parties is when the environment sends the inputs and receives the outputs. Finally, at the end of the execution of ρ, the environment outputs all the messages received.

The adversary can receive inputs from $\mathcal{Z}$ at any moment during the protocol execution and it can send replies to $\mathcal{Z}$ at any time. In order to capture any possible adversarial behaviour, $\mathcal{A}$ and $\mathcal{Z}$ can communicate freely throughout the course of the protocol and they can exchange information after any message sent between the parties and after any output made by a party.

Next, we look at the notion of corruption. By considering a party P corrupted we mean that from that point on that adversary has access to all the inputs and communication messages send or received by that party, and for any communication model, $\mathcal{A}$ can decide to alter such messages in any way it wants. Moreover, all the past incoming or outgoing messages of P are known to $\mathcal{A}$.

In order for $\mathcal{A}$ to corrupt a party P, it first informs $\mathcal{Z}$ by sending it a corruption message $(\mathit{corrupt}, P)$. Thus $\mathcal{Z}$ is aware at any given moment about the corruption state of all parties. Depending on the moment when the adversary $\mathcal{A}$ can corrupt a party, there are two corruption models: static and adaptive. In the static corruption model, the adversary $\mathcal{A}$ is allowed to corrupt parties only in the beginning of the protocol, before the respective parties receive their inputs from $\mathcal{Z}$. In contrast, if $\mathcal{A}$ is allowed to corrupt a party at any given moment during the protocol execution, then the adversary is called adaptive. Another way to look at the corruption model is by inspecting whether the adversary is passive, (i.e., only learns all inputs and communication messages a corrupted party sends and receives), or if $\mathcal{A}$ is active. The latter case implies $\mathcal{A}$ is allowed to modify any input a corrupted party gets and also any communication message sent.

In order to simplify the presentation, we use an equivalent definition for the static corruption model. As in the standard static case, the moment of corruption is fixed in the beginning, we can skip sending and receiving the corruption messages. Instead, we assume the corrupted parties are fixed from the start and the adversary does not have to choose them. Then, the previous static adversary definition is equivalent to the latter formulation, which we use in this work.

Besides corrupting parties, the adversary may interfere with the communication between honest parties. The most basic UC model ensures that all messages are handed to the adversary and the adversary delivers messages of its choice to all parties. This model makes no assumption on the communication properties: authenticity, secrecy or synchrony of the messages delivered. For the more specialised models of authenticated, secure or synchronous communication, an ideal functionality is added to the basic model to capture the respective properties.

Authenticated communication assumes the adversary cannot alter content of messages without being detected. The synchronous communication model captures the property that messages are all delivered and without delay from the moment they were generated. The ideally secure communication model assumes the adversary receives all messages, but it has neither access to the content of communication, nor possibility to modify any message without breaking authenticity. In this model, the adversarial capabilities are limited to either delaying or not delivering some or all messages between the uncorrupted parties.

When the protocol execution ends, $\mathcal{Z}$ outputs its view of that execution. This view contains messages that $\mathcal{Z}$ has received from the adversary $\mathcal{A}$ and outputs of all parties. Formally, $EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)$ denote the output of $\mathcal{Z}$ in an execution of the protocol ρ with adversary $\mathcal{A}$ and environment $\mathcal{Z}$, where k is the security parameter and z is the auxiliary input to the environment $\mathcal{Z}$. We denote by $EXEC_{\rho,\mathcal{A},\mathcal{Z}}$ the family of random variables $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$.

Ideal Process and Ideal Functionalities In order to formalize the ideal process, we do not want to define a different model, but we rather need to adapt to the one above. In the same way as in the real-world, the environment $\mathcal{Z}$ is the only ITM that can send inputs at any moment to the ideal process parties and to the ideal adversary. In the case of the ideal process, the adversary is called the ideal simulator and is commonly denoted by $\mathcal{S}$. Moreover, $\mathcal{Z}$ receives all the outputs generated by the parties, as well the possible outputs of $\mathcal{S}$.

The first difference is that in the ideal model there exists a trusted party, the ideal functionality, that cannot be directly accessed by the environment. This works as follows: Parties involved in the ideal process give their inputs to the ideal functionality which computes outputs for each party and sends these values to them. Hence, the role of the ideal functionality is to receive inputs, perform computations and send results to the ideal parties. As these parties do not take an active role in the computation and just send inputs to and receive outputs from the ideal functionality, they are called dummy parties of the ideal functionality.

The second difference with the real-world model is that messages delivered by the adversary to dummy parties are ignored. In the ideal protocol the adversary sends corruption messages directly to the ideal functionality. The ideal functionality then determines the effect of corrupting a party. A typical response is to let the adversary know all the inputs received and outputs sent by the party so far.

The environment $\mathcal{Z}$ and the simulator $\mathcal{S}$ can communicate freely during the execution of the ideal process. Additionally, the ideal functionality informs the simulator every time it wants to output a message. If the simulator agrees, then the respective output is made. This is required by the UC ideal model in order to allow $\mathcal{S}$ to simulate the behavior of a UC real world adversary delaying messages or not sending some or all of the communication among real-world protocol parties.

Similar to the real-world model, the environment $\mathcal{Z}$ outputs its view in the end of the ideal process execution. The view contains all the messages received from the simulator as well as all the messages that the dummy parties output to $\mathcal{Z}$. More formally, by $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)$ we denote the output of $\mathcal{Z}$ in an execution of the ideal process with the trusted party $\mathcal{F}$, simulator $\mathcal{S}$ and environment $\mathcal{Z}$, where k is the security parameter and z is the auxiliary input to the environment $\mathcal{Z}$. We denote by $EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the family of random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$.

Protocol Emulation We now define what it means that a real-world protocol ρ *emulates* with respect to UC security another real-world protocol θ. The environment

$\mathcal{Z}$ is the ITM deciding whether the interaction with the protocols and their respective adversaries can be distinguished.

All the ITMs used in either of the protocol executions for ρ or θ, including the environment $\mathcal{Z}$, are computationally bounded. Thus, it is sufficient if we formalize the notion of emulation in terms of computational indistinguishability. The environment $\mathcal{Z}$ will act as a distinguisher for the two protocol executions. Since all the information $\mathcal{Z}$ gains throughout its interaction is contained within the view $\mathcal{Z}$ outputs in the end, it is sufficient to compare the two views. Essentially, protocol ρ emulates protocol θ if for every adversary $\mathcal{A}$ there is an ideal simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$ the views of the two interactions are computationally indistinguishable.

Definition 8 (UC Security). *Let ρ and θ be PPT protocols. We say that ρ UC securely emulates θ if for every PPT adversary $\mathcal{A}$ there is a PPT simulator $\mathcal{S}$ such that for every PPT distinguisher $\mathcal{Z}$ and for every input distribution of $z \in \{0,1\}^*$, the two families of random variables $\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ and $\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$ are computationally indistinguishable.*

This general notion of emulation can be adapted to the special case of the ideal process. We say that a protocol realizes an ideal functionality if it emulates the ideal process for that functionality.

In the following we also use a relaxed version of this definition, where the order of quantifiers between the environment and the ideal-world simulator is reversed [15].

Definition 9 (Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$ and for every environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for every distribution of auxiliary input $z \in \{0,1\}^*$, we have:*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}$$

In the above definition, the output of the environment is considered to be a string of arbitrary length. If the only change we make to the above definition is to consider environments that have a 1-bit output, we obtain the notion of *1*-bit specialized simulator UC security. It has been an open problem [15] whether considering only environments with one bit output would produce an equivalent definition. In this work we show this is not the case.

Definition 10 (1-bit Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under 1-bit specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$ and for every 1-bit output environment $\mathcal{Z}$, there exists a simulator $\mathcal{S}$ such that for every input $z \in \{0,1\}^*$, we have:*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

If in the specialized simulator UC definition we let the simulator also depend on the distinguisher who is the only PPT machine to establish whether the output of the executions in the real UC world and ideal UC world cannot be told apart, then we obtain the notion of *w*eak specialized simulator UC security.

Definition 11 (Weak Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. We say that ρ emulates $\mathcal{F}$ under weak specialized simulator UC security if for every probabilistic polynomial time adversary $\mathcal{A}$, for every environment $\mathcal{Z}$ and for every distinguisher $\mathcal{D}$, there exists a simulator $\mathcal{S}$ such that for every distribution of input $z \in \{0,1\}^*$, we have:*

$$\{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}}.$$

A.2 Review of Concurrent General Composability Model

Next we review the notions of stand-alone security and security under concurrent general composability from [15] when additionally the order of quantifiers is reversed. The idea for having the order of quantifiers reversed emerged in [6] and was further studied in [12, 11].

Definition 12 (Weak Stand-alone Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. Then, ρ* computes $\mathcal{F}$ under weak security *if for every probabilistic polynomial-time real-model adversary $\mathcal{A}$ and for every probabilistic polynomial-time distinguisher $\mathcal{D}$ there exists a probabilistic polynomial-time ideal-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$:*

$$\{IDEAL^{\mathcal{F}}_{\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

In general, given a security notion there are two approaches to ensure the security properties of a protocol under composition. One way is to prove that the security property defined for the stand-alone case is preserved under composition. The other way is to define the security notion for the protocol directly under composition. The latter approach has the benefit that it captures the security property without having the drawback of a possible very strong and thus very restrictive stand-alone definition. Due to this reason we will focus on the second approach.

The concurrent general composition has been introduced in [15]. In this security model, a protocol ρ that is being investigated is run concurrently, possibly multiple times, with an arbitrary protocol π. The protocol π can be any arbitrary protocol and intuitively, it represents the network activity around ρ. There is another way to look at this: one can consider protocol π to be the external protocol that gives inputs and reads the outputs of the internal protocol ρ. As π is arbitrary, it can call multiple instances of ρ. However, we consider that different instances run independently from one another. The only correlation between them are the inputs and outputs, in the following way: the inputs for a certain run of ρ that are provided by π might depend on the previous inputs and outputs given and collected by π. Also, the messages of π may be sent concurrently to the execution of ρ. This composition of π with ρ is denoted as in the original notation by π^ρ.

As in the case of universal composability, in order to give the definition of security for ρ under concurrent general composition, we need to compare the execution of ρ with that of an ideal functionality so we have to define the real and the ideal world.

The computation in the ideal world is performed among the parties of π and a trusted party, playing the role of ideal functionality $\mathcal{F}$. Thus the messages considered in the ideal world are standard messages between parties of π and ideal messages between π and $\mathcal{F}$. The protocol π is providing $\mathcal{F}$ with inputs and after performing necessary computations, $\mathcal{F}$ sends the results to parties of π. The ideal adversary is called a simulator and as in the UC model, is denoted by $\mathcal{S}$. In addition to having full control over the parties it corrupts (see also the case of real world adversary), the simulator controls the scheduling of the messages between the parties of π and if not otherwise mentioned, it can also arbitrarily read and change messages. An exception is represented by the messages between π and $\mathcal{F}$: they are ideally secure, so the simulator can neither read nor change them [10].

During the computation, the honest parties follow the instructions given by π and in the end they output on their outgoing communication tape whatever value is prescribed by π. The corrupted parties output a special corrupted symbol and additionally the adversary may output an arbitrary image of its view. Let z be the auxiliary input for the ideal-world adversary $\mathcal{S}$ and let the inputs vector be $\bar{x} = (x_1, ..., x_m)$. Then the outcome of the computation of π with $\mathcal{F}$ in the ideal world (which we may also call $\mathcal{F}$-hybrid world) is defined by the output of all parties and $\mathcal{S}$ and is denoted by $\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k, \bar{x}, z)\}_{k\in\mathbb{N}}$.

The computation in the real world follows the same rules as the computation in the ideal world, only that this time there is no trusted party. Instead, each party of π has an ITM that works as the specification of ρ for that party. Thus, all messages that a party of π sends to the ideal functionality in the ideal world are now written on the input tape of its designated ITM. These ITMs communicate with each other in the same manner as specified for the parties of ρ. After the computation is performed, the results are output by these ITMs and the corresponding parties of π copy them on their incoming communication tapes. These messages are used by the parties of π in the same way as the messages output by $\mathcal{F}$ in the ideal-world. Similarly as above, in the real-world the adversary has full control over message delivery. There is one exception: any uncorrupted party of π can write and read directly to and from the input and respectively output tape of its designated ITM without any interference from the adversary [11]. Moreover, when we say that a real-world party is corrupted, we mean that a party of π and its corresponding ITM are corrupted [12].

Similarly to the ideal world, during the computation, the honest parties follow the instructions of π and their corresponding ITM and in the end they output on their outgoing communication tape whatever value is prescribed by π. The corrupted parties output a special corrupted symbol and additionally the real-world adversary $\mathcal{A}$ may output an arbitrary image of its view. Let z be the auxiliary input for $\mathcal{A}$ and let the inputs vector be $\bar{x} = (x_1, ..., x_m)$. Then the outcome of the computation of π with ρ in the real world is defined by the output of all parties and $\mathcal{A}$ and is denoted by $\{REAL_{\pi^{\rho},\mathcal{A}}(k, \bar{x}, z)\}_{k\in\mathbb{N}}$.

[10] This comes in contrast with the standard definition of UC ideal protocol execution, where it is not enforced that the channels between the trusted parties and the rest of the participants are ideally secure.

[11] Actually, the ideal adversary is not even aware of this taking place. This is similar to the UC communication between the environment and the real-world or ideal-world parties.

[12] This is not a restriction as an adversary that corrupts both a party of π and its ITM can just fully control only one of them and let the other one follow its prescribed protocol.

Independent of the world where the corruption takes place, the adversary could be static or adaptive. If the adversary is static, then the parties that are under the control of the adversary are fixed and do not depend on its auxiliary input or random tape. This is a restrictive definition of static corruption. However, the definition of adaptive corruption and the corresponding proof include the proof for a standard static corruption case. In the case of adaptive corruption, the adversary may decide during the protocol to arbitrarily corrupt a party, depending on the messages received so far. In both cases, once the adversary has corrupted a party then it learns all previous inputs and messages that the party received. From the moment of the corruption further, the adversary has full control over the messages that the party sends. Moreover, we consider that the adversary fully controls the message scheduling: he decides if and when to deliver the messages between output tape of one party (or, more general, machine) to the input tape of another. As mentioned above, there is one exception: the adversary does not have any control over the messages that an uncorrupted party sends to its corresponding ITM.

We are now ready to state the definition of security under concurrent general composition as in [15]. There are two notions of security under concurrent general composition: one for unbounded or polynomial calls that π may make to $\mathcal{F}$ and the second one, when π utilizes a fixed number of calls to $\mathcal{F}$.

Definition 13 (Security under Concurrent General Composition). *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* securely computes $\mathcal{F}$ under concurrent general composition *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideals calls to $\mathcal{F}$ and every probabilistic polynomial-time real-model adversary $\mathcal{A}$ for π^ρ, there exists a probabilistic polynomial-time hybrid-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$:*

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

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

We also use a weak version of the security definition from above.

Definition 14 (Weak Security under Concurrent General Composition). *Let ρ be a protocol and $\mathcal{F}$ a functionality. Then, ρ* computes $\mathcal{F}$ under concurrent general composition with weak security *if for every probabilistic polynomial-time protocol π in the $\mathcal{F}$-hybrid model that utilizes ideals calls to $\mathcal{F}$, for every probabilistic polynomial-time real-model adversary $\mathcal{A}$ for π^ρ and for every probabilistic polynomial-time distinguisher $\mathcal{D}$, there exists a probabilistic polynomial-time hybrid-model adversary $\mathcal{S}$ such that for every $\bar{x}, z \in \{0,1\}^*$:*

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}}(k,\bar{x},z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}}(k,\bar{x},z)\}_{k\in\mathbb{N}}.$$

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

B Postponed Proofs

In the following we need *one-time information-theoretic message authentication codes* so we include the definition below.

Definition 15 (One-Time Information-Theoretic Message Authentication Code). *A one-time information-theoretic message authentication code is a triple $(Gen, Mac, Verify)$ where $Gen(1^n)$ outputs a key k, $Mac(k, x)$ outputs a tag t (obtained using k) for the message x of length n and $Verify(k, m, t)$ outputs 0 or 1. The correctness property requires that $\forall n, \forall k$ in the range of $Gen(1^n)$ and $\forall x \in \{0,1\}^n$ we have $Verify(k, x, Mac(k, x)) = 1$.*

Moreover, the following security property is fulfilled. For every adversary $\mathcal{A}$ such that

$$Pr[(x', t') \leftarrow A(x, t) \wedge x' \neq x \wedge Verify(k, x', t') = 1 : x \leftarrow A(1^n), k \leftarrow Gen(1^n), t \leftarrow Mac(k, x)]$$

is negligible in n.

Lemma 4 (Equivalence between Weak Security under 1-bounded Concurrent General Composition and Weak Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. Then ρ securely computes $\mathcal{F}$ under 1-bounded concurrent general composition with weak security if and only if ρ securely implements $\mathcal{F}$ under weak specialized simulator UC security.*

Proof. As expected, the more involved part of the proof is the implication from weak security under concurrent general composition to specialized simulator 1-bit UC security. The reverse direction can be shown analogously to the proof existing in the initial version of [15].

Let $R_1, \ldots, R_m$ be the parties for ρ. Let $(\mathcal{A}, \mathcal{Z}, \mathcal{D})$ be a triple consisting of UC real world adversary (possibly adaptive), environment and distinguisher. We need to show there exists an UC ideal world simulator $\mathcal{S}$ such that the views of the environment in real world and in the ideal world cannot be distinguished by $\mathcal{D}$. The adversary $\mathcal{A}$ may not corrupt any party, in which case $\mathcal{A}$ is still capable of scheduling messages in the network. Additionally, remember that in the UC model the only messages that $\mathcal{A}$ has no control of, even by scheduling, are the input messages that the environment $\mathcal{Z}$ writes directly on the input tapes of the parties and the output messages that $\mathcal{Z}$ reads directly from the parties output tapes.

The intuition behind the proof is as follows: We use the fact that ρ composed with an instance of any protocol (i.e., even one that has more parties than ρ) is secure[13]. We construct a protocol π for $m + 2$ parties that besides the m parties of ρ has $P_{\mathcal{Z}}$ and $P_{\mathcal{A}}$ playing the role of $\mathcal{Z}$ and $\mathcal{A}$ respectively. In this way, we reduce the proof of weak specialized simulator UC security of ρ to weak security under concurrent general composition. As mentioned above, the adaptive adversary $\mathcal{A}$ could corrupt everyone or could corrupt no party and act as a network adversary. Thus, the motivation behind using the two extra parties in the protocol π is to ensure there is always an honest entity and also a corrupted entity, same as in the UC world. In order to model the ideally secure

[13] The security is of course in the sense of definition 14.

channels that the specialized simulator UC (real/ideal) setting ensures by definition between $\mathcal{Z}$ and the parties of ρ, we use one-time pads and one-time authentication MACs in the concurrent general composition world between $P_{\mathcal{Z}}$ and the parties of ρ.

However, it is important to know how long should the keys be. They should suffice for all necessary encrypted and authenticated communication. Let q be a polynomial such that for every security parameter n and for every i the value $q(n)$ bounds above the length of encryption and authentication keys needed between each pair $P_{\mathcal{Z}}$ and P_i with $i \in \{1, \ldots, m\}$. We postpone until after the description of π why such polynomial q exists and how it is computed.

Formally, protocol π is described below and it can be used for both the real and the ideal concurrent general composability worlds.

1. *Inputs:* Each party P_i with $i \in \{1, \ldots, m\}$ receives a pair (k^i_{mac}, k^i_{enc}) of keys[14]. Party $P_{\mathcal{A}}$ receives the empty string λ as input. Party P_{m+1} receives an input z and also the tuples $((k^1_{mac}, k^1_{enc}), \ldots, (k^m_{mac}, k^m_{enc}))$[15];
2. *Outputs:* The protocol outputs whatever $P_{\mathcal{Z}}$ outputs. The rest of the parties of π output an empty string λ;
3. *Instructions for P_i, with $i \in \{1, \ldots, m\}$:* When P_i receives $(input, x_i, t_i)$ from $P_{\mathcal{A}}$, it verifies the correctness of the tag. If verification succeeds, it computes $m_i = x_i \oplus k^i_{enc}$ and sends m_i either to its corresponding ITM that emulates R_i of ρ or to the functionality $\mathcal{F}$. (This depends on whether π is part of the composed protocol π^ρ or $\pi^{\mathcal{F}}$. Remember that independent of the channels model, an adversary in the concurrent general composability world cannot interfere in any way with the messages that an uncorrupted party of π wants to send to its associated ITM for ρ.) If verification fails, then P_i halts. When the ITM emulating R_i or when $\mathcal{F}$ respectively sends the output value y_i to P_i, then P_i computes $e_i = y_i \oplus k^i_{enc}$ and $v_i = MAC(k^i_{mac}, e_i)$ and sends the message $(output, e_i, v_i)$ to party $P_{\mathcal{Z}}$;
4. *Instructions for $P_{\mathcal{Z}}$:* Upon receiving an input value z, it uses it for internally invoking $\mathcal{Z}$. When internal $\mathcal{Z}$ wants to send a message $(input, m_i)$ to party i, then $P_{\mathcal{Z}}$ computes $x_i = m_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, x_i)$ and sends $(input, x_i, t_i)$ to P_i. When $P_{\mathcal{Z}}$ receives a message $(output, y_i, v_i)$ from party P_i, it first checks the correctness of the tag v_i. If verification succeeds, then $P_{\mathcal{Z}}$ computes $m_i = y_i \oplus k^i_{enc}$ and stores m_i. Otherwise, it halts. When internal $\mathcal{Z}$ wants to read the output tape of party i, then $P_{\mathcal{Z}}$ looks up if there is a message m_i stored from party P_i. If so, it writes m_i to corresponding tape of $\mathcal{Z}$, otherwise it just writes λ to $\mathcal{Z}$. Regarding the communication with its adversary, when $P_{\mathcal{Z}}$ receives a message from $\mathcal{Z}$ of the form $(\mathcal{Z}, \mathcal{A}, m)$, it forwards it to $P_{\mathcal{A}}$. Similarly when $P_{\mathcal{Z}}$ receives a message of the form $(\mathcal{A}, \mathcal{Z}, m)$ from $P_{\mathcal{A}}$, it forwards it internally to $\mathcal{Z}$.

[14] For ease of notation, we use one encryption key and one MAC key per party P_i, as they can be considered long enough to encrypt and authenticate the entire communication between P_i and $P_{\mathcal{Z}}$. However, for each different encryption (authentication) that needs to be performed, a new part of the string k^i_{enc} (and k^i_{mac}, respectively) is used.

[15] The input strings to π may have any distribution and the indistinguishability between the real and the ideal concurrent general composability worlds would still be preserved. However, for this proof we restrict the inputs to encryption keys (i.e., they are uniformly distributed in $\{0,1\}^{q(k)}$) and MAC keys (i.e., they are generated with the *Gen* key generation algorithm).

5. *Instructions for $P_{\mathcal{A}}$:* This party has no predefined instructions. $P_{\mathcal{A}}$ is needed in order to provide a means of communication for the adversary of the general concurrent composition setting which in this model can only send messages through a corrupted party[16].

We now explain how the polynomial q is chosen. Since the communication between $P_{\mathcal{Z}}$ and each of the parties P_i with $i \in \{1, \ldots, m\}$ has to be secure and authenticated, the length of the secret keys for the one-time pad and and for the one-time MAC should be long enough. The intuition is that the length of the encryption keys shared by $P_{\mathcal{Z}}$ and P_i is bounded above by the length of the longest string that machine $\mathcal{Z}$ can write plus the longest string that R_i can write. Since both machines are polynomially bounded and they are fixed before the protocol π is constructed, there exist a polynomial q_i such that $q_i(n)$ bounds from above the length of the common encryption keys for every security parameter n. Moreover, the length of the secret key needed for the authenticated messages between $P_{\mathcal{Z}}$ and P_i is at most as long as the one-time pad secret keys. Putting the above arguments together we conclude there exists a polynomial q such that $q(n) \geq max\{q_1(n), \ldots, q_m(n)\}$.

For the protocol π given above we construct an adversary $\mathcal{A}_\pi$ interacting with the composed protocol π^ρ. Intuitively, the task of $\mathcal{A}_\pi$ is to enable the communication among $\mathcal{Z}$ (invoked by $P_{\mathcal{A}}$), $\mathcal{A}$ (invoked by the adversary $\mathcal{A}_\pi$) and the ITMs implementing ρ, in the same way as it happens in the UC real world. In order to make this work and for reasons explained above, the adversary A_π corrupts $P_{\mathcal{A}}$. We construct the adversary $\mathcal{A}_\pi$ as follows: It internally runs the code of the UC real world adversary $\mathcal{A}$ and if $\mathcal{A}$ corrupts a party R_i, then $\mathcal{A}_\pi$ corrupts the party P_i together with its corresponding ITM for computing ρ. The intuition is that $\mathcal{A}_\pi$ instructs the corrupted parties of π to run the protocol as before, while their corresponding corrupted ITMs follow the instructions of $\mathcal{A}$. The handling of messages by $\mathcal{A}_\pi$ is as follows:

1. Input messages $(\mathit{input}, x_i, t_i)$ sent by $P_{\mathcal{Z}}$ are forwarded *immediately* by $\mathcal{A}_\pi$ to P_i; Output messages $(\mathit{output}, e_i, v_i)$ sent by P_i are *immediately* forwarded to $P_{\mathcal{Z}}$. Moreover, as soon as party P_i is corrupted, its current state and all its previously received messages are sent to $\mathcal{A}$. The information that $\mathcal{Z}$ expects to receive upon corruption is sent by $\mathcal{A}_\pi$ to $P_{\mathcal{Z}}$. All messages received from this point on by P_i are forwarded by A_π to $\mathcal{A}$.
2. When $P_{\mathcal{Z}}$ sends a message $(\mathcal{Z}, \mathcal{A}, m)$ to party $P_{\mathcal{A}}$, then $\mathcal{A}_\pi$ forwards it to its internal run of $\mathcal{A}$ as if coming from $\mathcal{Z}$. The messages $(\mathcal{A}, \mathcal{Z}, m)$ that $\mathcal{A}$ wants to send to $\mathcal{Z}$ are forwarded by $\mathcal{A}_\pi$ to $P_{\mathcal{Z}}$;
3. All messages that $\mathcal{A}$ instructs a corrupted party R_i to send to an uncorrupted party R_j will be forwarded by $\mathcal{A}_\pi$ to the corresponding ITM of P_j as if coming from the corresponding ITM of P_i; However $\mathcal{A}$ schedules messages among parties R_i with $i \in \{1, \ldots, m\}$, $\mathcal{A}_\pi$ does the same for the messages between the corresponding ITMs of parties P_i with $i \in \{1, \ldots, m\}$.
4. The adversary $\mathcal{A}_\pi$ has no control over the messages between an uncorrupted P_i and its corresponding ITM for computing ρ.

[16] This is in contrast to the UC model where even if none of the protocol parties is corrupted, the adversary can interact with the environment $\mathcal{Z}$.

After having defined the protocol π and the adversary $\mathcal{A}_\pi$, we prove that the output of $P_\mathcal{Z}$ in the execution of π^ρ (which we denote by $\{REAL_{\pi^\rho,\mathcal{A}_\pi}(k,\bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}}$) and the output of $\mathcal{Z}$ in the UC real world are identically distributed. For every $z \in \{0,1\}^*$, let $\bar{z} = (z, k^1_{enc}, k^1_{mac}, \ldots, k^m_{enc}, k^m_{mac}), \lambda, (k^1_{enc}, k^1_{mac}), \ldots, (k^m_{enc}, k^m_{mac})$ be the vector where the first component is the input to $P_\mathcal{Z}$, the second component is the input to $P_\mathcal{A}$, and each of the other components is the input to a party P_i, with $i \in \{1,\ldots,m\}$.

We prove that for every $z \in \{0,1\}^*$, for every k^i_{enc} randomly chosen from $\{0,1\}^{q(n)}$ and for every k^i_{mac} generated by $Gen(1^{q(n)})$ we have:

$$\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \equiv \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k,\bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}} \tag{5}$$

which as a special case, of course implies:

$$\{EXEC_{\rho,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k,\bar{z})|P_\mathcal{Z}\}_{k\in\mathbb{N}} \tag{6}$$

Our claim is based on the following facts: First, the inputs to parties are provided by $\mathcal{Z}$ in both models, as in the composed protocol π^ρ the party $P_\mathcal{Z}$ distributing the inputs is internally running $\mathcal{Z}$. Thus the input messages in both worlds are identically distributed. By construction, $\mathcal{A}_\pi$ follows the instructions of $\mathcal{A}$ (i.e., for network scheduling and for the corrupted messages among the corresponding ITMs for $P_1,\ldots,P_m$) and it also provides an internal perfect emulation for the view of $\mathcal{A}$. Once an honest party P_i receives an input, it immediately writes it on the input tape of its associated ITM for ρ. This implies that such a party with its ITM follows the same protocol as the corresponding party of ρ. We can now conclude that the view of $\mathcal{Z}$ in the UC real world for ρ and the view of $P_\mathcal{A}$ in the composed protocol π^ρ are identically distributed, so equation (5) follows.

According to the definition of weak security under 1-bounded general concurrent composition, we know that for the triple π, $\mathcal{A}_\pi$ and $\mathcal{D}$, there exists a polynomially bounded hybrid simulator $\mathcal{S}_\pi$ such that for every $\bar{z}$ defined as above we have:

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(k,\bar{z})\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{REAL_{\pi^\rho,\mathcal{A}_\pi}(k,\bar{z})\}_{k\in\mathbb{N}}. \tag{7}$$

We are now ready to construct a simulator $\mathcal{S}$ for the UC ideal world by using $\mathcal{S}_\pi$. We have to observe that in the hybrid world of concurrent general composition and in the UC real world the messages going over the network are the same. Intuitively, the new simulator $\mathcal{S}$ has to have a scheduling indistinguishable from that of $\mathcal{S}_\pi$ so the constructed simulator $\mathcal{S}$ internally invokes $\mathcal{S}_\pi$. As a short summary of the messages that have to be defined for $\mathcal{S}$: communication from $\mathcal{S}$ to $\mathcal{F}$, communication from $\mathcal{S}$ to $\mathcal{Z}$ and network scheduling (between parties of π and $\mathcal{F}$). As $\mathcal{S}$ internally runs $\mathcal{S}_\pi$, the constructed adversary has to provide an emulation for the entities that $\mathcal{S}_\pi$ is interacting with: the parties of $\pi^{\mathcal{F}}$[17]. Such an emulation of $\pi^{\mathcal{F}}$ consists of defining the input/output messages of the parties, the messages among $P_1,\ldots,P_m, P_\mathcal{A}, P_\mathcal{Z}$ and the messages from $P_1,\ldots,P_m$ to $\mathcal{F}$.

[17] Observe that it is actually sufficient to simulate the parties of π without the messages sent by $\mathcal{F}$ as they can be forwarded by $\mathcal{S}$ from its communication with the ideal functionality.

1. Messages sent by $\mathcal{F}$ to $\mathcal{S}$ are forwarded to the internally emulated $\mathcal{S}_\pi$. The messages that internally emulated $\mathcal{S}_\pi$ wants to send to $\mathcal{F}$ are forwarded by $\mathcal{S}$ to $\mathcal{F}$. Similarly, the messages that internally emulated $\mathcal{S}_\pi$ sends to the internally simulated $P_\mathcal{Z}$ are forwarded by $\mathcal{S}$ to $\mathcal{Z}$. The messages that $\mathcal{Z}$ sends to $\mathcal{S}$ are forwarded internally to $\mathcal{S}_\pi$ as coming from $\mathcal{P}_\mathcal{Z}$.
2. Simulation of $P_\mathcal{Z}$: When $\mathcal{S}$ receives a message $(\mathcal{Z}, \mathcal{A}, m)$ from $\mathcal{Z}$, it sends it to the internally emulated $P_\mathcal{A}$ as if coming from the emulated $P_\mathcal{Z}$. When $\mathcal{S}_\pi$ instructs emulated $\mathcal{P}_\mathcal{A}$ (which is a corrupted party) to send a message $(\mathcal{A}, \mathcal{Z}, m)$ to $P_\mathcal{Z}$, the simulator $\mathcal{S}$ forwards the same message to $\mathcal{Z}$.
3. Simulation of $P_\mathcal{A}$: As an uncorrupted party, $\mathcal{P}_\mathcal{A}$ does not do anything, just receives messages from $\mathcal{P}_\mathcal{Z}$. These messages were actually sent by $\mathcal{Z}$ to $\mathcal{S}$. When internal $\mathcal{S}_\pi$ wants to corrupt emulated $P_\mathcal{A}$ (and this is actually the first party of π that $\mathcal{S}_\pi$ corrupts), then all that $\mathcal{S}$ needs to do is to send $\mathcal{S}_\pi$ all the messages it received from $\mathcal{Z}$.
4. In the UC ideal world, when an uncorrupted dummy party $\mathcal{D}_i$ receives an $(input, m_i)$ from the environment $\mathcal{Z}$, it immediately forwards the input value to $\mathcal{F}$. When $\mathcal{S}$ receives over the network such a message[18], it generates x_i randomly in the length of the received input and a MAC key k^i_{mac} with the corresponding generation algorithm, computes $t_i = MAC(k^i_{mac}, x_i)$ and internally sends the message $(input, x_i, t_i)$ to P_i as if coming from $P_\mathcal{Z}$. When $\mathcal{F}$ wants to send an output message (i.e., same discussion as above) to $\mathcal{D}_i$, the simulator $\mathcal{S}$ internally randomly generates y_i in the length of the output received over the network, then computes $v_i = MAC(k^i_{mac}, y_i)$ and sends message $(output, y_i, v_i)$ to $\mathcal{S}_\pi$ as if coming from the ideal functionality in $\pi^\mathcal{F}$.
   Whenever $\mathcal{S}_\pi$ corrupts a party P_i, we have one of the following 3 cases:
   -For a corrupted party P_i, that $\mathcal{S}_\pi$ wants to corrupt before a certain input is sent to it by $P_\mathcal{Z}$, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$, informs $\mathcal{Z}$ about it and generates a correct key pair (k^i_{enc}, k^i_{mac}) for encryption and authentication and gives them to $\mathcal{S}_\pi$[19]. When input value(s) x_i for $\mathcal{D}_i$ are received by $\mathcal{S}$ over the network[20], then $\mathcal{S}$ computes $y_i = x_i \oplus k^i_{enc}$ and $v_i = MAC(k^i_{mac}, y_i)$. Next, $\mathcal{S}$ sends y_i, v_i to $\mathcal{S}_\pi$ as coming from $P_\mathcal{Z}$. When an output o_i is sent by $\mathcal{F}$ to $\mathcal{D}_i$, then $\mathcal{S}$ computes $c_i = x_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, y_i)$ and sends c_i, t_i to simulated P_i as if coming from $P_\mathcal{Z}$.
   -For a corrupted party P_i, that $\mathcal{S}_\pi$ corrupts after a certain input is sent to P_i, but before the corresponding output is received, first the emulation from the case of uncorrupted input takes place. Thus, a message (y_i, v_i) has been already sent from $\mathcal{P}_\mathcal{Z}$ to P_i. When the corruption takes place, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$, informs $\mathcal{Z}$ about it and generates a correct key pair (k^i_{enc}, k^i_{mac}) for encryption and authentication. Then it sends the pair to $\mathcal{S}_\pi$, together with the correct input x_i in plain. When an output o_i is sent by $\mathcal{F}$ to

[18] If the channels between the dummy parties and the ideal functionality are ideally secure, then the value received could also be encrypted, so what is forwarded should not depend on what is received.
[19] This simulates the information that $\mathcal{S}_\pi$ should learn from the newly corrupted (simulated) party.
[20] As $\mathcal{D}_i$ is corrupted, they are received from $\mathcal{Z}$ unencrypted.

$\mathcal{D}_i$, then $\mathcal{S}$ computes $c_i = x_i \oplus k^i_{enc}$ and $t_i = MAC(k^i_{mac}, y_i)$ and sends c_i, t_i to simulated P_i as if coming from $P_{\mathcal{Z}}$.

-For a corrupted party P_i that $\mathcal{S}_\pi$ corrupts after a certain input is sent to it and after the corresponding output is received, the simulator $\mathcal{S}$ corrupts the corresponding dummy party $\mathcal{D}_i$ and informs $\mathcal{Z}$ about the corruption[21]. Then $\mathcal{S}$ reads in plain the input and output values received by $\mathcal{D}_i$ and, using the simulated encrypted messages, computes the corresponding encryption keys which are sent to S_π as P_i input.

5. The following is valid only for honest parties: When $\mathcal{S}_\pi$ delivers a message from P_i to the ideal functionality in $\pi^{\mathcal{F}}$, then $\mathcal{S}$ delivers the same message from $\mathcal{D}_i$ to $\mathcal{F}$[22]. When $\mathcal{S}_\pi$ delivers an output from P_i to $P_{\mathcal{Z}}$, then $\mathcal{S}$ delivers the output from $\mathcal{F}$ to $\mathcal{D}_i$[23].

In order to conclude the proof we have to show that the output of the executions in both hybrid composition world and UC ideal world can be distinguished only with negligible probability. For this we detail the following three steps: a proof that the view of internally emulated $\mathcal{S}_\pi$ is identical with $\pi^{\mathcal{F}}$, a proof that the messages in the two worlds (hybrid composition and the UC ideal world) are identically distributed and finally, a proof that the delivery of output messages happens in the same time in both worlds.

We start by analyzing $\mathcal{S}$ internal emulation for $\mathcal{S}_\pi$. It is easy to see that by construction $\mathcal{S}_\pi$, internally invoked by $\mathcal{S}$, gets and delivers the same messages as $\mathcal{S}_\pi$ does in the concurrent general composition world.

Next, we look at the messages sent between entities in both worlds. In the ideal UC world, the inputs are sent by $\mathcal{Z}$ and in the hybrid world with $\pi^{\mathcal{F}}$, the inputs are sent by $P_{\mathcal{Z}}$ who runs $\mathcal{Z}$. The messages that are sent between $P_{\mathcal{Z}}$ (running $\mathcal{Z}$) and $P_{\mathcal{A}}$ (corrupted and controlled by $\mathcal{S}_\pi$), are the same as the messages sent in the UC ideal world between $\mathcal{Z}$ and $\mathcal{S}$ who runs $\mathcal{S}_\pi$. In both worlds, the messages sent by parties to the ideal functionality are the same: the honest parties just forward their inputs and the corrupted parties are instructed by $\mathcal{S}_\pi$ and respectively by $\mathcal{S}$ running $\mathcal{S}_\pi$. We only need to show that the delivery of messages is the same in both worlds. Combining this claim with the proof above, we obtain that the outputs of both worlds are computationally indistinguishable.

Finally, we compare message delivery in both worlds. It is clear that the messages between adversary and the environment $\mathcal{Z}$ or party $P_{\mathcal{Z}}$ running $\mathcal{Z}$ are identically delivered. The same hods for messages between the parties and the ideal functionality. We treat in more detail the case of inputs and outputs delivery. By definition, in the UC world, the input messages are written by $\mathcal{Z}$ directly on the input tapes of the protocol parties and for the honest parties, the adversary has no control over this step[24]. In

[21] Note that the simulation done by $\mathcal{S}$ for uncorrupted P_i receiving encrypted and authenticated input and output from $\mathcal{P}_{\mathcal{Z}}$ already took place.

[22] Actually, the simulator $\mathcal{S}_\pi$ has to make two deliveries (from $P_{\mathcal{Z}}$ to P_i and from P_i to the ideal functionality in $\pi^{\mathcal{F}}$), before $\mathcal{S}$ does the delivery of message from $\mathcal{D}_i$ to $\mathcal{F}$.

[23] Similarly as above, the simulator $\mathcal{S}_\pi$ has to make two deliveries (the output of $\mathcal{F}$ to P_i and from P_i to $P_{\mathcal{Z}}$), before $\mathcal{S}$ does its delivery from $\mathcal{F}$ to $\mathcal{D}_i$.

[24] However, in the UC ideal world, immediately after receiving inputs, the honest dummy parties are activated and they write their inputs on the communication tape for the ideal

the execution of $\pi^{\mathcal{F}}$, $P_{\mathcal{Z}}$ is distributing the inputs to the rest of the parties, but they are scheduled by $\mathcal{S}_\pi$, so we cannot know when they are delivered. However, we ensure that in both worlds an input of an honest party reaches the ideal functionality in the same time. Indeed, this holds as an honest dummy party $\mathcal{D}_i$ once it receives its input, it immediately sends it to the ideal functionality. As simulator $\mathcal{S}$ delivers this message only after $\mathcal{S}_\pi$ has delivered the same message to $\mathcal{F}$, we have shown the claim.

Similarly, we show that an output message is delivered to $\mathcal{Z}$ and to the party $P_{\mathcal{Z}}$ in the same time. Both entities have basically the same instructions. We assume the machine environment $\mathcal{Z}$ reads all output tapes whenever it is activated. This gives the most power to the environment to distinguish between the delivery of messages. By construction, $\mathcal{S}$ sends an output of $\mathcal{F}$ to an honest dummy party $\mathcal{D}_i$ only when $\mathcal{S}_\pi$ sends the same output to $P_{\mathcal{Z}}$. Once it receives its output, the honest $\mathcal{D}_i$ immediately writes this value on its output tape (and this can be read by $\mathcal{Z}$ at any time). Analogously, $\mathcal{Z}$, (which is internally run by $P_{\mathcal{Z}}$), can read at any time the tape with output messages sent for it. So we have that also the outputs from the ideal functionality are delivered simultaneously in both worlds. This implies that for every $\bar{z}$ defined as before we have:

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(k,\bar{z})|P_{\mathcal{Z}}\}_{k\in\mathbb{N}} \equiv \{EXEC_{\mathcal{F},\mathcal{S}_\pi,\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \tag{8}$$

Thus, it holds that:

$$\{HYBRID^{\mathcal{F}}_{\pi,\mathcal{S}_\pi}(k,\bar{z})|P_{\mathcal{Z}}\}_{k\in\mathbb{N}} \overset{\mathcal{D}}{\equiv} \{EXEC_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N}} \tag{9}$$

By combining relations (6),(7) and (9), we can conclude the proof.

We are now able to prove the main result:

Theorem 3 (Equivalence between Weak Security under 1-bounded Concurrent General Composition and 1-bit Specialized Simulator UC Security). *Let ρ be a protocol and $\mathcal{F}$ an ideal functionality. Then ρ securely computes $\mathcal{F}$ under 1-bounded concurrent general composition with weak security if and only if ρ securely implements $\mathcal{F}$ under 1-bit specialized simulator UC security.*

Proof. The theorem follows immediately by combining lemma 4 and lemma 1.

functionality. As the simulator is responsible for the delivery of messages, in this way it will learn that inputs have been sent to the ideal functionality.