On the Necessity of Rewinding in Secure Multiparty Computation

Michael Backes[1], Jörn Müller-Quade[2], and Dominique Unruh[1]

[1] Saarland University, Saarbrücken, Germany
{backes,unruh}@cs.uni-sb.de
[2] Universität Karlsruhe, Germany
muellerq@ira.uka.de

Abstract. We investigate whether security of multiparty computation in the information-theoretic setting implies their security under concurrent composition. We show that security in the stand-alone model proven using black-box simulators in the information-theoretic setting does not imply security under concurrent composition, not even security under 2-bounded concurrent self-composition with an inefficient simulator and fixed inputs. This in particular refutes recently made claims on the equivalence of security in the stand-alone model and concurrent composition for perfect and statistical security (STOC'06). Our result strongly relies on the question whether every rewinding simulator can be transformed into an equivalent, potentially inefficient non-rewinding (straight-line) simulator. We answer this question in the negative by giving a protocol that can be proven secure using a rewinding simulator, yet that is not secure for any non-rewinding simulator.

1 Introduction

Multiparty computation allows a set of parties with private inputs to jointly compute a given function on their inputs such that the function evaluation does not reveal any information about the inputs of other parties except for what can already be deduced from the result of the evaluation. These properties should hold even in the presence of a malicious adversary which fully controls the network and which may control some subset of the parties that then may arbitrarily deviate from the protocol.

Defining the security of a multiparty computation via an ideal execution with an incorruptible trusted party has proven a salient technique in the past. More precisely, the trusted party receives the inputs of all parties, correctly evaluates the considered function and hands back the result. In this work, we consider multiparty computation in the information-theoretic setting, where the adversary is computationally unbounded, and where consequently no underlying complexity-theoretic assumptions are required.

Multiparty computation has been investigated for a variety of different security levels and execution scenarios. As far as security levels are concerned, *perfect security* means that the result obtained in the real protocol run with the real adversary is identical to the result obtained in an ideal protocol run with the simulator; *statistical security* is defined analogously but allows the real and ideal results to deviate from each other by a small amount. As far as different execution scenarios are concerned in which the protocol is executed in, we distinguish between security in the *stand-alone model*, security under *concurrent self-composition*, and security under *concurrent general composition*. The stand-alone model only considers a single execution of the protocol under consideration, and no other protocol is run concurrently. While this constituted the standard setting for analyzing distributed security protocols in the past, a common understanding arose that protocols have to be secure even when executed many times in parallel (concurrent self-composition), or even when run in an arbitrary network, where many different protocols may run concurrently (concurrent general composition).

A considerable amount of work has been dedicated to carrying over results obtained in the stand-alone model into the more realistic concurrent setting. In particular, it is highly desirable to analyze protocols in the stand-alone model with its much simpler execution scenario and restricted adversary capabilities, and to derive theorems that allow for subsequently carrying these analyses over into the more sophisticated models of concurrent composition. Our main result however constitutes a separation of these two notions, i.e., we show that stand-alone security and security under concurrent composition do not coincide in the information-theoretic setting, neither for perfect nor for statistical security, and not even for fixed inputs and 2-bounded concurrent self-composition (i.e., only two executions of the same protocol are executed concurrently). We believe that this helps to foster our understanding of the relationships of the respective security notions, thereby refuting some recently made claims, see below.

1.1 Related Work

Defining the security of a protocol by comparing it with an ideal specification has proven a salient technique in the past, see e.g. [7,8,2,16,3,17,18,4,1], since it entails strong compositionality properties. In recent years, several results have been obtained concerning the relation of concurrent composition of function evaluations to other security notions. First, [14] showed that a protocol that can be concurrently composed and used as a subprotocol of another protocol (concurrent general composition) is already secure with respect to specialised-simulator UC (a variant of the UC notion with another order of quantifiers). It was left open, however, whether concurrent general composition was also necessary for specialised-simulator UC. Then [15] showed that for a large class of functions (those which can be used to transfer a bit), the possibility to compose a protocol concurrently already implies the possibility to use that protocol as a subprotocol in arbitrary contexts, i.e., concurrent self-composition and concurrent general composition coincide for these functions. The relations left open by [14] were proven by [10,11] who showed that concurrent general composition is equivalent to specialised-simulator UC in the case of statistical and perfect security, and strictly stronger in the case of computational security.

All these results relate concurrent composition only to stronger notions, e.g., variants of the UC notion. To get feasibility results, it is necessary to look for relations to weaker security notions, e.g. variants of the stand-alone model. This approach was taken by [12], who could show that stand-alone security with a *non-rewinding* black-box simulator already implies concurrent self-composition (and in the perfect case even concurrent general composition). Unfortunately, they also showed that stand-alone security with a non-rewinding black-box simulator is not sufficient for concurrent general composition in the case of statistical or computational security. A similar approach had earlier successfully been pursued in [5] in a different security model based on [16]. They showed that perfect security with non-rewinding simulators allows for concurrent composition.

The central question left to solve consequently was how these results behave in the presence of a rewinding simulator, which arguably constitutes a crucial scenario in modern cryptography. It was thus investigated in [12] in which ways the requirement that the simulator has to be non-rewinding can be weakened such that the established implications remain valid. They gave theorems that every rewinding black-box simulator can be replaced by an equivalent, computationally unbounded non-rewinding simulator. A consequence of these theorems was that stand-alone security with a rewinding black-box simulator is already sufficient for concurrent self-composition in the statistical case and even for concurrent general composition in the perfect case. Our results however refute these claims.

In [9], it was shown that the task of performing a coin toss given a shorter coin toss as seed can be realised with respect to rewinding black-box simulators but not with respect to specialised-simulator UC. This resembles our results (see Section 5 for a discussion) but applies only to the hybrid model (i.e., with access to some ideal functionality) while the results in [12] were formulated in the bare model. Furthermore, in contrast to the examples given here, those in [9] do not cover the case of perfect security or of deterministic ideal functions, and they did not explicitly apply their examples to the problem of rewinding vs. non-rewinding simulators.

1.2 Our Results

We first show that rewinding constitutes a necessary ingredient for proving certain protocols secure:

Theorem 1 (Necessity of rewinding – informal). *There exist protocols that are secure in the information-theoretic stand-alone setting with a rewinding black-box simulator, and yet are not secure in this setting with* any *non-rewinding black-box simulator.*

This disproves the following claim from [12]: *Any black-box simulator for a perfect or statistically secure protocol can be transformed into a rewinding black-box simulator.*[1] However, it still leaves open the question if stand-alone security for protocols with rewinding black-box simulators implies security under

[1] The wording has been adapted to our notation.

at least concurrent self-composition (it only invalidates the existing proof chain). However, also this implication turns out not to hold, already if only two instances of the same protocol are run concurrently and if only fixed inputs are considered:

Theorem 2 (Separating stand-alone model and concurrent (self-)composition – informal). *There exist protocols that are secure in the stand-alone model with a rewinding black-box simulator, and yet are not secure under 2-bounded concurrent self-composition, not even with an inefficient simulator and fixed inputs. This holds for both the perfect and the statistical case.*

This refutes the following claim from [12]: *Every protocol that is perfectly/statistically secure in the stand-alone model, and has a black-box simulator, is secure under concurrent self-composition with fixed inputs, with an inefficient simulator.*

The counterexample for proving this theorem exploits a specific protocol realization for a specific function. Thus one might still ask if there are other protocols that can securely implement the considered function while at the same time providing security under concurrent composition. If this was the case, the impact of Theorem 2 would be considerably weakened as one might identify the good protocol realizations for the troublesome function under consideration and then still achieve strong compositionality guarantees using those realizations.

However, we show that this is not the case in general, at least not for probabilistic functionalities, statistical security, and concurrent general composition: The task of extending coin toss (i.e., obtaining $k+1$ random coins from an ideal functionality which gives only k random bits) can be securely implemented with statistical security in the stand-alone model. However, there provably does not exist any protocol for coin toss extension with respect to statistical concurrent general composition.

Theorem 3 (A stronger separation – informal). *There exists a probabilistic function that can be securely implemented using a single instance of a probabilistic function in the stand-alone model with statistical security and an efficient rewinding black-box simulator, but that cannot be securely implemented by any protocol with a polynomial number of rounds with respect to statistical concurrent general composition.*

2 Notation and Definitions

The stand-alone model. In the *stand-alone model*, a protocol π *securely implements* an ideal function f if for every set of corrupted parties C and for every adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that the families of random variables $\mathrm{REAL}_{\pi,\mathcal{A},x}(k)$ and $\mathrm{IDEAL}_{f,\mathcal{S},x}(k)$ are indistinguishable in the security parameter k for all inputs $x = (x_1, \dots, x_n)$. Here $\mathrm{REAL}_{\pi,\mathcal{A},x}$ is the output of the adversary and of the uncorrupted parties in the following interaction: The uncorrupted parties $i \notin C$ get input x_i. Then the parties interact as prescribed by the protocol π. The adversary controls the corrupted parties, i.e., he can send messages in the name of a party $i \in C$ and receives all messages for parties $i \in C$. Similarly, $\mathrm{REAL}_{f,\mathcal{S},x}$ consists of the output of the simulator $\mathcal{S}$ and of the results of the function f. Here the inputs of f corresponding to the uncorrupted parties are chosen according to x, and the inputs of the corrupted parties are entered by the simulator. The simulator can choose his output in dependence of the output of the function.[2]

In this paper, we distinguish two main flavors of the stand-alone model: *perfect* and *statistical* security. In the case of perfect security, $\mathrm{REAL}_{\pi,\mathcal{A},x}(k)$ and $\mathrm{IDEAL}_{f,\mathcal{S},x}(k)$ have to be identically distributed, while in the case of statistical security they must be statistically indistinguishable. Both cases do not impose any limitations on the adversary and the simulator. For completeness, we also mention *computational security* which requires the simulator and the adversary to be polynomially bounded and the two families of random variables to be computationally indistinguishable. (Sometimes, one also requires the simulator to be efficient in the case of statistical and perfect security. We address this case by explicitly stating whether the simulator is efficient or inefficient in the respective theorems.) A more detailed exposition of the stand-alone model can be found in [6, Chapter 7].

Concurrent self-composition. The stand-alone model does not a-priori guarantee that two or more concurrent executions of the same protocol are secure, even if a single instance is secure. Therefore one is interested in the notion of concurrent self-composition, which roughly says that several instances of a given protocol securely implement the same number of instances of the ideal function. In more detail, a protocol π securely implements a function f with respect to *g-bounded concurrent self-composition* if g instances of π (considered as a single protocol) securely implement g instances of the ideal function f in the stand-alone model. Here we distinguish two cases: either the inputs to the different instances of the protocol are all fixed in advance (i.e., each party i receives a vector $x_i = (x_{i,1}, \dots, x_{i,g})$ of inputs and uses $x_{i,j}$ as input for the j-th instance), or the inputs to some instances can be chosen adaptively in dependence of messages sent in other instances. For us, only the first case is relevant, which is called *concurrent self-composition with fixed inputs*. More details on this definition can be found in [13]. The case of adaptive inputs is discussed in [15].

The special case, that g-bounded concurrent self-composition is given for any polynomial g we call *polynomially-bounded concurrent self-composition* or simply *concurrent self-composition*.

Concurrent general composition The notion of *concurrent general composition* further extends the notion of concurrent self-composition. A protocol π securely implements an ideal function f with respect to *g-bounded concurrent general composition* if for any protocol σ that uses g copies of π as subprotocols, σ securely implements σ^f in the stand-alone model (where σ^f denotes σ with all

[2] In case the function gives different output to different parties (i.e., are asymmetric), the situation gets slightly more complicated. However, all functions given in this paper are symmetric, so the issue does not arise.

instances of π replaced by ideal evaluations of the function f). More details on this notion are found in [14].

Black-box simulators. A natural restriction on the simulators is to require *black-box simulators*, i.e., the simulator is not chosen in dependence of the adversary, but instead we require that there is an oracle Turing machine $\mathcal{S}$ (the black-box simulator) such that for every adversary $\mathcal{A}$ we have indistinguishability of $\mathrm{REAL}_{\pi,\mathcal{A},x}(k)$ and $\mathrm{IDEAL}_{f,\mathcal{S}^{\mathcal{A}},x}(k)$ where $\mathcal{S}^{\mathcal{A}}$ is $\mathcal{S}$ with black-box access to $\mathcal{A}$. A fine point in this definition is whether the simulator may rewind the adversary, i.e., whether the simulator may at some point in time make a snapshot of the state of the adversary and then return the adversary to that state. Normally, one permits this operation and speaks of *rewinding* black-box simulators. On the other hand, we may also require the simulator to be *non-rewinding*, so that it can perform only one execution of the black-box adversary. This is often also called a *straight-line simulator*.

3 The Necessity of Rewinding

We show that for certain functions and corresponding protocols, the ability to rewind a black-box simulator is a crucial and unavoidable ingredient for achieving simulation-based security proofs in secure multi-party computation. Throughout this section, we consider the multiplication function f_{mult} receiving two inputs from a simple domain.

Definition 4 (Function f_{mult}). *The function f_{mult} takes an input $a \in \{0,1\}$ from Alice and an input $b \in \{1,2\}$ from Bob and returns $a \cdot b$.*

The corresponding protocol π_{mult} that is intended to securely implement f_{mult} is defined as follows.

Definition 5 (Protocol π_{mult}). *Alice and Bob get inputs $a \in \{0,1\}$ and $b \in \{1,2\}$, respectively.*

- *Alice sends a to Bob. If $a \notin \{0,1\}$, Bob assumes $a = 0$.*
- *Bob sends $c := a \cdot b$ to Alice. If $a = 0$, but $c \neq 0$, Alice assumes $c = 0$. If $a = 1$, but $c \notin \{1,2\}$, Alice assumes $c = 1$.*
- *Both parties output c.*

We first show that π_{mult} securely implements f_{mult} if rewinding of the black-box adversary is permitted. After that, we show that rewinding is also necessary, i.e., π_{mult} securely implements f_{mult} if and only if rewinding is permitted.

Lemma 6 (π_{mult} securely implements f_{mult}). *The protocol π_{mult} securely implements f_{mult} with perfect security in the stand-alone model with an efficient rewinding black-box simulator.*

We start with a short overview of the proof for the sake of illustration and subsequently delve into the details. First, consider the case that Bob is corrupted. In this case, the simulator conducts a simulation of the real protocol by executing the real adversary in the role of Bob and choosing Alices input as 1. Since in this case the result of the function equals Bob's input, the simulator learns his input b as chosen by the adversary. Then the simulator enters this input b into the ideal function f_{mult}. From the result of the function f_{mult} one can deduce Alice's input a (the result is 0 if and only if a is zero). Then the simulator rewinds and restarts the adversary, this time choosing the true input a that Alice has input. Thus the simulator learns the output *out* the adversary gives when the input of Alice is a. Finally, the simulator outputs *out*. This constitutes a perfect simulation since the simulator enters the same input b into the function f_{mult} and produces the same output *out* as the adversary does in the real model.

Now consider the case that Alice is corrupted. In this case simulation is straightforward: Alice's input a is sent in the clear as the first message of the protocol (by the black-box adversary), so the simulator enters this input into the ideal function f_{mult}. The simulator finally has to simulate the message $c = a \cdot b$ sent by Bob. This is straightforward since the simulator knows the correct value of c from the output of f_{mult}. Finally, the simulator gives the same output as the simulated adversary, thus achieving a perfect simulation.

We now transform these intuitions into a rigorous proof.

Proof. In the case that no party is corrupted, the security (correctness) of the protocol is obvious.

Now first consider the case that Bob is corrupted. The simulator $\mathcal{S}$ for this case proceeds as follows:

- First fix the random tape of the adversary $\mathcal{A}$, which is given as a black-box.
- Then send the message $\hat{a} = 1$ to the adversary.
- Let $\hat{c}$ be the reply of the adversary. If $\hat{c} \notin \{1, 2\}$, set $\hat{c} := 1$ instead (as Alice would have done herself).
- Set $b := \hat{c}$ and use b as Bob's input to the function f_{mult}.
- Let *res* be the result of the function f_{mult}. Let $\tilde{a} := 0$ if $res = 0$ and $\tilde{a} := 1$ otherwise.
- Rewind the adversary (but use the same random tape) and send the message $\tilde{a}$ to the adversary.
- When the adversary outputs *out*, output *out*.

Let now an adversary $\mathcal{A}$ be given. Without loss of generality, assume $\mathcal{A}$ to be deterministic (this is indeed no restriction since the random tape for which the simulator is least successful can be hardwired into $\mathcal{A}$). Then the following values are defined:

- The message $\hat{c}_a$ sent by the adversary when receiving a message $a \in \{0, 1\}$.
- The output $\widehat{out}_a$ of the adversary when he receives a message $a \in \{0, 1\}$.

Without loss of generality again, assume $\hat{c}_0 = 0$ and $\hat{c}_1 \in \{1, 2\}$ (since Alice and the simulator replace other values by valid ones).

Given the values of $\hat{c}_a$ and out_a, and the input a, we can calculate the different values that occur during the run of the ideal protocol, in particular the values $\mathrm{IDEAL}_{f_{mult},\mathcal{S}^{\mathcal{A}},a}(k) = (res, out)$. We summarise these values in the left table of Figure (1). Furthermore, we can also calculate the different values that occur during a real protocol run, i.e., c being the message sent by $\mathcal{A}$ to Alice, and $\mathrm{REAL}_{\pi_{mult},\mathcal{A},a}(k) = (res, out)$ the result of the function and the output of the real adversary. These values are summarized in the right table of Figure (1).

Ideal protocol:

a	$\hat{c}$	res	$\tilde{a}$	out
0	$\hat{c}_1$	0	0	out_0
1	$\hat{c}_1$	$\hat{c}_1$	1	out_1

Real protocol:

a	c	res	out
0	0	0	out_0
1	$\hat{c}_1$	$\hat{c}_1$	out_1

Fig. 1. Values occuring in the run of the ideal protocol (left side) and the real protocol (right side).

Since both *out* and *res* have the same values in real and ideal model (for all values of a), perfect security in the case that Bob is corrupted follows.

Now we consider the case that Alice is corrupted. In this case, the following simple simulator $\mathcal{S}$ achieves a perfect simulation:

- Query $\mathcal{A}$ for the first message a.
- If $a \notin \{0, 1\}$, set $a := 0$.
- Then a is passed to the function f_{mult} as Alice's input.
- The result $c := res = a \cdot b$ is given to the adversary as the answering message from Bob.
- Finally, output the simulated black-box adversary's output.

It is again straightforward to check that this constitutes a perfect simulation in the case of a corrupted Alice. □

The next lemma shows that considering only non-rewinding simulators is not sufficient to prove that π_{mult} securely implements f_{mult}.

Lemma 7 (π_{mult} needs rewinding). *The protocol π_{mult} does* not *securely implement f_{mult} in the stand-alone model with respect to perfect, statistical, or computational security, with any* non-rewinding *black-box simulator (not even with inefficient ones).*

We again start with a proof sketch. We consider the case that Bob is corrupted. To give the correct input to the ideal function f_{mult}, the simulator needs to interact with the black-box adversary before invoking f_{mult}. Furthermore, to get the correct value of Bob's input, the simulator has to choose Alice's input to be $a = 1$ in the interaction with the black-box adversary (this is exactly how the simulator in Lemma 6 was constructed). In addition to causing the result of the function to be correct, the simulator also needs to output what the black-box adversary would output in the same situation. The simulator already executed the adversary with $a = 1$; consequently if the true input turns out to be 0, the simulator cannot learn what the adversary would output in that situation unless he rewinds the adversary and executes it with Alice's input a set to 0. However, we assumed that rewinding is not permitted, and hence the simulation fails.

Proof. For contradiction, we assume that there is a non-rewinding black-box simulator $\mathcal{S}$ such that for all adversaries $\mathcal{A}$ corrupting Bob and all inputs $a \in \{0,1\}$ from Alice, we have

$$\mathrm{REAL}_{\pi_{mult},\mathcal{A},a}(k) \approx \mathrm{IDEAL}_{f_{mult},\mathcal{S}^{\mathcal{A}},a}(k). \tag{1}$$

For brevity, we write $\mathcal{S}$ instead of $\mathcal{S}^{\mathcal{A}}$.

Furthermore, we construct a family of adversaries $\mathcal{A} = \mathcal{A}(b, o)$ with $b \in \{1, 2\}$ and $o \in \{0, 1\}$. The adversary $\mathcal{A}$ corrupts Bob and behaves as follows: When receiving a message a from Alice, he sends $c := a \cdot b$ to Alice. Finally, he outputs o if $a = 0$ and $\perp$ otherwise.

To describe the real model, we use the following notation: Let a denote the input of Alice. Since we only consider the case that Bob is corrupted, a is also the message from Alice to Bob in the real model. Let c denote Bob's answer. Since Alice is uncorrupted, the result of the function evaluation in the real model is also c. When using the adversary $\mathcal{A}$ given above, it is $c = a \cdot b$. Finally let out denote $\mathcal{A}$'s output.

To prevent confusion, we add a swung dash ($\sim$) to the random variables in the ideal model. That is, $\tilde{c}$ is the result of the function f_{mult}, and $\tilde{b}$ is the input given by $\mathcal{S}$ in Bob's stead to the function f_{mult}. Further, let $\tilde{a}$ be the first message given by $\mathcal{S}$ to the black-box $\mathcal{A}$ (which corresponds to the message sent by Alice in the real protocol). Finally, let $\widetilde{out}$ denote the output of the simulator $\mathcal{S}$. The input of the uncorrupted Alice is still called a, since it is the same in real and ideal model (a, b and o are not random variables).

The simulator $\mathcal{S}$ has two possibilities: Either he queries the function f_{mult} (with some input $\tilde{b}$) before giving the message $\tilde{a}$ to the black-box adversary $\mathcal{A}$ (we call this event F), or he first sends the message $\tilde{a}$ to $\mathcal{A}$.

Assume that event F occurs with non-negligible probability $P(F)$. In that case, $\tilde{b}$ is chosen independently of b, so there exists a b, s.t. the probability that $b \neq \tilde{b}$ is at least $\frac{1}{2}P(F)$. Then, in the case $a = 1$ the probability that $a \cdot b \neq a \cdot \tilde{b} = \tilde{c}$ is also at least $\frac{1}{2}P(F)$. In the real model however, we have $c = a \cdot b$. Since c and $\tilde{c}$ denote the result of the function f_{mult} in the real and ideal model, this is a contradiction to Equation (1).

So event F happens only with negligible probability. Therefore, we can assume without loss of generality that the simulator $\mathcal{S}$ always first sends $\tilde{a}$ to the adversary, and only then inputs $\tilde{b}$ into f_{mult}. Assume now that the probability $P(\tilde{a} = 0)$ is non-negligible. Then consider the case $a = 1$. For $\tilde{a} = 0$ the adversary $\mathcal{A}$ answers with $\tilde{a} \cdot b = 0$, which is independent of b. In that case, $\tilde{b}$ is chosen by the simulator independently of b. Therefore, $P(\tilde{b} \neq b) \geq \frac{1}{2}P(\tilde{a} = 0)$ for some choice of b. Since $\tilde{c} = a \cdot \tilde{b}$, $P(\tilde{c} \neq a \cdot b)$ is non-negligible. But in the real model we have $c = a \cdot b$, in contradiction to Equation (1). Therefore $P(\tilde{a} = 0)$ is negligible, so we can assume without loss of generality that the simulator always sends $\tilde{a} = 1$ to $\mathcal{A}$ (before invoking f_{mult}).

By construction, when receiving $\tilde{a} = 1$, the adversary will give output $\perp$. Therefore, the simulator's output $\widetilde{out}$ is independent of o, so for some o we have $P(out \neq o) \geq \frac{1}{2}$. But in the case $a = 0$, the adversary outputs $out = o$ in the real model. This contradicts (1). So there cannot exist a non-rewinding simulator that fulfills Equation (1). □

The general statement induced by the previous results is summarized in the following corollary. Its proof is a direct consequence of Lemma 6 and 7.

Theorem 8. *Let f be a function and π a protocol that securely implements f with perfect, statistical, or computational security in the stand-alone model. Then π does not necessarily securely implement f with perfect, statistical, or computational security, respectively, with any* non-rewinding *black-box simulator.*

In the proofs we have assumed the following variant of the stand-alone model: After running the protocol, the *output* of the adversary should be indistinguishable in the real and the ideal model. Another popular variant of the model instead considers the *view* of the adversary. Our examples can be easily transferred to the latter setting. Instead of giving some output *out*, the adversary sends a protocol message containing *out* that is ignored by the protocol. Then our proofs directly carry over to that variant of the stand-alone model. This also applies to the proofs in the next section.

The example given in this section could also be used to show that stand-alone security does not imply concurrent self-composition: When two instances of π_{mult} run concurrently, a corrupted Bob can enforce the sum of the outputs to equal 2 (unless Alice inputs 0 in both cases), which is impossible given access to two copies of f_{mult}. However, instead of showing this in detail, we will show the separation between stand-alone security and concurrent self-composition in the next section using another example which we consider to be more instructive.

4 Perfect Stand-Alone Security does not imply Concurrent Self-Composition

In this section, we show that for certain functions and corresponding protocols, security in the stand-alone model is not necessarily sufficient for guaranteeing security under concurrent self-composition. Throughout this section, we consider the functions f_{min_x}, where x is a natural number that constitutes a parameter of the function. The function f_{min_x} outputs the minimum of its inputs.

Definition 9 (Function f_{min_x}). *Let $x \geq 2$ be an integer. The function f_{min_x} takes two inputs $a, b \in \{1, \dots, x\}$ from Alice and Bob, where a is odd and b is even. The result $f_{min_x}(a, b)$ is the minimum of a and b.*

Definition 10 (Protocol π_{min_x}). *Let $x \geq 2$ be an integer. Alice gets an odd input $a \in \{1, \dots, x\}$, Bob an even input $b \in \{1, \dots, x\}$.*

- *The protocol π_{min_x} proceeds in at most $x-2$ rounds $1, \dots, x-2$.*
- *In round r for an odd value r, Alice sends* `no` *if $r \neq a$, and* `yes` *if $r = a$.*
- *In round r for an even value r, Bob sends* `no` *if $r \neq b$, and* `yes` *if $r = b$.*
- *If any other message is sent, the message is assumed by the recipient to be* `no`*.*

– *As soon as a message* `yes` *has been sent (in some round r), the protocol terminates, and both parties output r.*
– *If no message* `yes` *is sent in any round, the output is $x - 1$.*

Lemma 11 **(π_{min_x} securely implements f_{min_x}).** *Let $x \geq 2$ be an integer. The protocol π_{min_x} securely implements f_{min_x} with perfect security in the stand-alone model and with an efficient rewinding black-box simulator.*

The proof can be sketched as follows; for the sake of readability, we only elaborate on the case that Bob was corrupted. The simulator starts a simulation of a real protocol where the (black-box) real adversary plays the role of Bob and Alice's input is set to its largest possible value a_{max}. Then the minimum of Alice's and Bob's input, which is returned by the ideal function f_{min_x}, allows us to calculate Bob's input b. This value b is then used as Bob's input in the ideal evaluation of f_{min_x}. So far, we have already guaranteed that the result of the function is identical in both the real and the ideal model. The simulator now has to learn what the adversary would output. Learning this output requires the simulator to perform a second simulation of the real protocol with the adversary using the correct value of Alice's input a (here we need the possibility to rewind). In the case that the result of the function is smaller than Bob's input b, this is an easy task since the input of Alice is equal to the result of the function. If the function result is however equal to the input of Bob, we can only deduce that Alice's input is larger than Bob's input. In this case, the simulator simply assumes the largest possible value a_{max} that Alice might have input. Since the protocol terminates in the round corresponding to Bob's input b, the adversary will never learn whether Alice used her maximum input or just some input greater than b. So in both cases, the simulator learns what output the adversary gives in the real model, and it can thus perform a perfect simulation.

Proof. Let a and b denote the inputs of Alice and Bob, respectively. Let a_{max} be the largest odd integer with $a_{max} \leq x$ (Alice's largest possible input), and b_{max} the largest even integer with $b_{max} \leq x$ (Bob's largest possible input).

If Alice and Bob are uncorrupted, it is easy to check, that the protocol indeed calculates the minimum of a and b. Hence correctness (security) in this case is clear.

Now consider the case that Bob is corrupted. In this case, the following simulator $\mathcal{S}$ achieves a perfect simulation:

– First fix the random tape of the adversary $\mathcal{A}$, which is given as a black-box.
– Simulate a protocol run of π_{min_x} with $\mathcal{A}$ where Bob is corrupted and Alice gets input a_{max}. Let $\widetilde{res}$ be Alice's output in this function evaluation.
– Let $\tilde{b} := \widetilde{res}$ if $\widetilde{res} < a_{max}$, and $\tilde{b} := b_{max}$ otherwise.
– Invoke the ideal function π_{min_x} using $\tilde{b}$ as Bob's input. Let res be the result of the function.
– Let $\tilde{a} := res$ if $res < \tilde{b}$, and let $\tilde{a} := a_{max}$ otherwise.
– Then simulate a protocol run of π_{min_x} with adversary $\mathcal{A}$ where Bob is corrupted and Alice gets input $\tilde{a}$. Let *out* be the adversary's output in that protocol run.

– Output out.

Now, consider an adversary $\mathcal{A}$ that corrupts Bob. Without loss of generality, assume $\mathcal{A}$ to be deterministic (cf. the proof of Lemma 6). Assume further that $\mathcal{A}$ only sends messages `yes` and `no`, and that these messages only occur in the appropriate (i.e., even) rounds.

Then we can associate the following values with this adversary $\mathcal{A}$:

- By $\hat{b}$ we denote the number of the first round in which the adversary answers with `yes` when Alice always sends `no`. If $\mathcal{A}$ never sends `yes`, let $\hat{b} := b_{max}$. (Intuitively, $\hat{b}$ denotes Bob's input as chosen by the adversary.)
- By out_a we denote the output made by the adversary in the execution of a real protocol when Alice has input a.

Note that both $\hat{b}$ and out_a are obtained deterministically since both $\mathcal{A}$ and the protocol π_{min_x} are deterministic. The following facts are easy to observe:

(i) In a protocol run of π_{min_x} with $\mathcal{A}$ where Alice gets input a, Alice's output (i.e., the result of the function) is always $\min(a, \hat{b})$. (This holds since in case $\hat{b} < a$, the adversary cannot distinguish Alice from an Alice with input a_{max}, and if $\hat{b} > a$, the protocol guarantees that a is output. The case $\hat{b} = a$ does not occur, since $\hat{b}$ is even and a is odd.)

(ii) For $a, a' > \hat{b}$ it is $out_a = out_{a'}$. (Because then the protocol terminates in round $\hat{b}$, so Alice behaves identically with inputs a and a'.)

We now show, that $\mathcal{S}$ entails a perfect simulation in the case that Bob is corrupted. Consider an ideal protocol run consisting of the simulator $\mathcal{S}$ (having black-box access to the adversary $\mathcal{A}$), and the ideal function f_{min_x} receiving some input a on Alice's side.

First, the simulator simulates a real protocol run with input a_{max} for Alice. By fact (i), Alice's output in that protocol run is $\min(a_{max}, \hat{b})$. Therefore it is $\widetilde{res} = \min(a_{max}, \hat{b})$.

If $\min(a_{max}, \hat{b}) = \widetilde{res} < a_{max}$, it follows $\widetilde{res} = \hat{b}$. If $\min(a_{max}, \hat{b}) = \widetilde{res} \geq a_{max}$, if follows $\hat{b} \geq a_{max}$, and therefore $\hat{b} = b_{max}$. In both cases the value $\tilde{b}$ calculated by the simulator equals $\hat{b}$.

Since the simulator enters $\tilde{b}$ as Bob's input into the function f_{min_x}, the result of the function is $res = \min(a, \tilde{b}) = \min(a, \hat{b})$.

Consider the case $\min(a, \tilde{b}) = res < \tilde{b}$. Then $a = res$ and the simulator sets $\tilde{a} := res$, so $out_a = out_{\tilde{a}}$. In the case $\min(a, \tilde{b}) = res \geq \tilde{b}$, it is $a > \tilde{b}$ (since a and $\tilde{b}$ cannot be equal, being of different parity). The simulator then chooses $\tilde{a} := a_{max} \geq a > \tilde{b}$, so $a, \tilde{a} > b$. By fact (ii), we then have $out_a = out_{\tilde{a}}$. So in both cases, the output of the simulator and the output of the adversary coincide, i.e., we have $out = out_a$.

In a nutshell, the result of the function (i.e., Alice's output) in an ideal protocol run is $res = \min(a, \hat{b})$, and the output of the simulator is $out = out_a$.

In the real model the result of the function is $\min(a, \hat{b})$ according to Fact (i). Moreover, the output of the adversary is out_a by definition.

Consequently, the output of the adversary and and the result of the function evaluation are identical in real and ideal model so that perfect security in the case that Bob is corrupted follows.

The case that Alice is corrupted is proven identically, except for exchanging the roles of Alice and Bob, the letters a and b and the words *odd* and *even*. □

Lemma 12 (π_{min_x} does not compose concurrently). *Let $x \geq 4$ be an integer. The protocol π_{min_x} does* not *securely implement f_{min_x} with perfect, statistical, or computational 2-bounded concurrent self-composition with fixed inputs.*

This even holds if we allow unbounded non-black-box simulators that may adaptively query the two ideal instances of f_{min_x}.

The basic idea underlying the proof of the lemma can be given as follows. A corrupted Bob will start two parallel sessions of the real protocol π_{min_x} with Alice and subsequently forward all protocol messages from one protocol session into the other and vice versa. The output of both protocols will then be equal to the smaller input that Alice has made (± 1). This is impossible to achieve when concurrently interacting with two ideal functions: The simulator has to invoke one of the functions first. If it uses a large value for Bob's input, the simulation will fail if Alice gave a large input to that first function, and a small input to the second one. If it uses a small value for Bob's input, it will fail if Alice gave large inputs to both functions.

Actually, the proof gives a slightly stronger result than Lemma 12 since it shows that π_{min_x} does not even allow for *parallel* composition.

Proof. We construct an adversary $\mathcal{A}$ corrupting Bob and attacking two concurrently composed instances of π_{min_x} as follows:

- In each odd round, he receives messages $m_{A,1}, m_{A,2}$ from the two instances A_1, A_2 of Alice.
- In each even round, he sends $m_{A,1}$ to the second instance A_2 of Alice, and $m_{A,2}$ to the first instance A_1.

Let a_1 denote the input of the first instance of Alice, and a_2 the input of the second instance of Alice. Let further res_1 and res_2 denote the respective outputs.

Then if $a_1 < a_2$, one easily sees that $res_1 = a_1$ and $res_2 = a_1 + 1$ (since the forwarded `yes`-message reaches A_2 only in the $(a_1 + 1)$-st round). Similarly, if $a_1 > a_2$, it is $res_1 = a_2 + 1$ and $res_2 = a_2$. Finally, if $a_1 = a_2 < a_{max}$, we get $res_1 = res_2 = a_1 = a_2$, and if $a_1 = a_2 = a_{max}$ we finally have $res_1 = res_2 = x - 1$.

Now, consider an arbitrary simulator $\mathcal{S}$. This simulator has access to two instances $f^1_{min_x}, f^2_{min_x}$ of the function f_{min_x}, which receive inputs a_1 and a_2 from Alice, respectively. The simulator may now invoke the functions one after the other. Let $f^i_{min_x}$ denote the function invoked first (i.e., i is a random variable), and b_i the Bob-input given to $f^i_{min_x}$ by $\mathcal{S}$. Since both i and b_i cannot depend on the inputs a_1 and a_2 from Alice, at least one of the following two cases occurs with probability at least $\frac{1}{4}$ for suitable choices of a_1 and a_2: (i) It is $a_1 = a_2 = a_{max}$ and $b_i < x - 1$. (ii) It is $a_i = a_{max}$, $a_{3-i} = 1$ and $b_i \geq x - 1$.

In case (i) the result of $f^i_{min_x}$ will be $b_i < x - 1$. However, as we have shown above, in the real model, with inputs $a_1 = a_2 = a_{max}$ the result of $f^1_{min_x}$ and $f^2_{min_x}$ (i.e., the outputs of the Alice-instances) would be $x - 1$. Therefore the results differ in real and ideal model.

In case (ii) the result of $f^i_{min_x}$ will be greater or equal $x-1$ (since $a_{max} \geq x-1$). In the real model however, the results of the functions would be 1 and 2 (in some order), because $a_{3-i} = 1$. Since $x \geq 4$, both results are smaller than $x - 1$ and so the results differ in real and ideal model.

So with non-negligible probability, the function results in the ideal model do not match those in the real model. The insecurity of π_{min_x} under 2-bounded concurrent self-composition follows. □

These lemmas yield the following theorem. Its proof is a direct consequence of Lemma 11 and 12.

Theorem 13. *Let f be a function and π a protocol that securely implements f with perfect, statistical, or computational security in the stand-alone model with an efficient rewinding black-box simulator. Then π does not necessarily securely implement f with perfect, statistical, or computational security under 2-bounded concurrent self-composition, not even with an inefficient non-black-box simulator and fixed inputs.*

5 A Stronger Separation

The results proven so far show that certain protocols can be secure in the stand-alone model, require rewinding, and do not allow composition. The natural question arising here is whether this issue of composition depends on a specific choice of the protocol while some other protocol for the same task might be composable. We show that, at least for the case of a probabilistic functionality, statistical security, and concurrent *general* composition, this is not the case: The task of extending coin toss (i.e., obtaining $k+1$ random coins from an ideal funtionality which gives only k random bits) can be realised with statistical security in the stand-alone model. However, there does not exist a protocol for coin toss extension with respect to statistical concurrent general composition.

Corollary 14. *There exists a probabilistic function $\mathcal{F}$ that can be securely implemented using a single instance of a probabilistic function $\mathcal{G}$ in the stand-alone model with statistical security and an efficient rewinding black-box simulator, but that cannot be securely implemented using a single instance of $\mathcal{G}$ by any protocol with a polynomial number of rounds with respect to statistical concurrent general composition.*

Proof. Let $\mathcal{F}$ be the $(k+1)$-bit coin-toss functionality (i.e., the functionality that provides *one* uniformly chosen string of length $k+1$, cf. [9]). $\mathcal{F}$ is a function that ignores its inputs. Let $\mathcal{G}$ be the k-bit coin-toss functionality.

In [9] it is shown, that there is a protocol securely implementing $\mathcal{F}$ using $\mathcal{G}$ in the stand-alone model with statistical security (and an efficient rewinding black-box simulator).

Assume that there is a polynomial-round protocol π securely implementing $\mathcal{F}$ using a single instance of $\mathcal{G}$ with respect to statistical concurrent general composition. Then by the results in [14], π also securely implements $\mathcal{F}$ with respect to statistical specialised-simulator UC ([14] only shows the computational case, but the proof easily carries over to the statistical case). But in [9] it is shown that no polynomial-round protocol π using a single instance of $\mathcal{G}$ exists that securely implements $\mathcal{F}$ with respect to specialised-simulator UC (actually, [9] state the theorem for UC, but their proof also shows the case of specialised-simulator UC, since the environments constructed in their proof do not depend on the simulator). Thus we have a contradiction and the lemma follows. □

6 Conclusion and Open Questions

We have shown that in the information-theoretic setting, the existence of a rewinding simulator in the stand-alone model is not sufficient for the existence of a non-rewinding simulator, nor for achieving concurrent composition. In that light, the question naturally arises which additional constraints may be imposed on the black-box simulator so that it can be converted into a non-rewinding black-box simulator (which then in turn allows for concurrent composition, see [12]). A major problem in coming up with a constructive transformation of a rewinding simulator into a non-rewinding one seems to be the following: The original simulator's program may explicitly require several executions of the black-box adversary, e.g., the knowledge-extractor in most proofs of knowledge executes the adversary (i.e., the prover) twice or more often, and then uses the results of *several of the executions* to construct the required output. Such a knowledge-extractor cannot easily be transformed into a non-rewinding one, since then its program would suddenly find itself in the unexpected situation of terminating without having run the black-box adversary twice, yielding undefined results. The simulators in the counterexamples given in this work are of this form as well. On the other hand, many simulators found in the literature use rewinding only to backtrack from wrong choices. After having backtracked, they forget that they rewound the adversary and start anew, hoping to select the right choice this time. Protocol with simulators of this kind include, e.g., the well-known zero-knowledge proofs of graph-isomorphism and graph-3-colouring.

More formally we call a simulator *obliviously rewinding*, if it is an oracle Turing machine with the following extension: at any point during its execution, the simulator may set a *marker* M. Then a snapshot of the state of the simulator and of the black-box adversary is taken. Furthermore, the simulator may then at any other point of its execution choose to return to a marker M. If he chooses to do so, the state of the black-box adversary *and of the simulator itself* are restored to the snapshot that was taken when the marker M was set. The program of an obliviously rewinding simulator does not run into an undefined situation when the simulation suddenly goes through without any rewinding. Therefore, it may be possible that such an obliviously rewinding simulator can indeed be transformed into a non-rewinding one as proposed in [12]. We leave this as an open question.

Acknowledgements. We thank the anonymous referees for helpful comments.

References

1. M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. IACR Cryptology ePrint Archive 2004/082, Mar. 2004.
2. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
3. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.
5. Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In M. Bellare, editor, *Advances in Cryptology, Proceedings of CRYPTO '00*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92. Springer-Verlag, 2000.
6. O. Goldreich. *Foundations of Cryptography – Volume 2 (Basic Applications)*. Cambridge University Press, May 2004. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.
7. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
8. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.
9. D. Hofheinz, J. Müller-Quade, and D. Unruh. On the (im)possibility of extending coin toss. In S. Vaudenay, editor, *Advances in Cryptology, Proceedings of EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 504–521. Springer-Verlag, 2006. Full version online available at `http://eprint.iacr.org/2006/177`.
10. D. Hofheinz and D. Unruh. Comparing two notions of simulatability. In J. Kilian, editor, *Theory of Cryptography, Proceedings of TCC 2005*, Lecture Notes in Computer Science, pages 86–103. Springer-Verlag, 2005. Online available at `http://iaks-www.ira.uka.de/home/unruh/publications/hofheinz05comparing.html`.
11. D. Hofheinz and D. Unruh. Simulatable security and polynomially bounded concurrent composition. In *IEEE Symposium on Security and Privacy, Proceedings of SSP '06*, pages 169–182. IEEE Computer Society, 2006. Full version online available at `http://eprint.iacr.org/2006/130.ps`.
12. E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. In *38th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2006*, pages 109–118. ACM Press, 2006. Online available at `http://www.cs.biu.ac.il/~lindell/abstracts/IT-composition_abs.html`.

13. Y. Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *35th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2003*, pages 683–692. ACM Press, 2003.
14. Y. Lindell. General composition and universal composability in secure multi-party computation. In *44th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2003*, pages 394–403. IEEE Computer Society, 2003. Online available at `http://eprint.iacr.org/2003/141`.
15. Y. Lindell. Lower bounds for concurrent self composition. In M. Naor, editor, *Theory of Cryptography, Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, 2004.
16. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.
17. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.
18. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.