J Cryptol (2018) 31:1120–1163
https://doi.org/10.1007/s00145-018-9296-x

Journal of CRYPTOLOGY

On the (Im-)Possibility of Extending Coin Toss*

Dennis Hofheinz · Jörn Müller-Quade
Karlsruhe Institute of Technology, Karlsruhe, Germany
Dennis.Hofheinz@kit.edu

Dominique Unruh
University of Tartu, Tartu, Estonia

Communicated by Yevgeniy Dodis.

Received 22 December 2010 / Revised 27 March 2018
Online publication 26 April 2018

Abstract. We consider the task of extending a given coin toss. By this, we mean the two-party task of using a single instance of a given coin toss protocol in order to interactively generate *more* random coins. A bit more formally, our goal is to generate n common random coins from a single use of an ideal functionality that gives $m < n$ common random coins to both parties. In the framework of universal composability, we show the impossibility of securely extending a coin toss for statistical and perfect security. On the other hand, for computational security, the existence of a protocol for coin toss extension depends on the number m of random coins that can be obtained "for free." For the case of stand-alone security, i.e., a simulation-based security definition without an environment, we present a protocol for statistically secure coin toss extension. Our protocol works for superlogarithmic m, which is optimal as we show the impossibility of statistically secure coin toss extension for smaller m. Combining our results with already known results, we obtain a (nearly) complete characterization under which circumstances coin toss extension is possible.

Keywords. Coin toss, Universal composability, Reactive simulatability, Cryptographic protocols.

1. Introduction

Blum showed in [6] how to flip a coin over the telephone line. His protocol guarantees that even if one party does not follow the protocol, the other party still gets a uniformly distributed coin toss outcome. This general concept of generating common randomness in a way such that no dishonest party can dictate the outcome proved very useful in cryptography, for example, in the construction of protocols for general secure multiparty computation.

*This is the full version of the paper presented at Eurocrypt 2006.

Here, we are interested in the task of *extending* a given coin toss. That is, suppose that two parties already have the possibility of making a single m-bit coin toss. Is it possible for them to get $n > m$ bits of common randomness? The answer we come up with is basically: "It depends on the security model and on the length of the coin toss used as seed."

The first thing the extensibility of a given coin toss depends on is the required kind of security. In this work, we will consider simulation-based security notions, in which a protocol is secure if and only if it "imitates" an ideal functionality. For the case of coin toss, this ideal functionality will act as a trusted host that simply equips both parties with common random coins. However, we would like to stress that we would like to model an *interactive* coin toss protocol. Hence, the coin toss ideal functionality first expects an "activation signal" from both parties before handing out the random coins. This is quite different from a "common random string" (CRS) functionality that does not require such activation signals. (In fact, in this work, we will investigate also CRSs and CRS extension protocols, with somewhat different results compared to the coin toss case.)

A little more technically, one specific kind of security requirement (which we call "standalone simulatability" here) is that the protocol imitates the ideal coin toss functionality in the sense of [14], where a simulator has to invent a realistic protocol run after learning the outcome of the ideal coin toss. A stronger type of requirement is to demand universal composability, which basically means that the protocol imitates an ideal coin toss functionality even in arbitrary protocol environments. Security in the latter sense can conveniently be captured in a simulatability framework like the universal composability framework [7] (see also [19]) or the reactive simulatability model [3,24].

Orthogonal to this, one can vary the level of fulfillment of each of these security requirements. For example, one can demand stand-alone simulatability of the protocol with respect to polynomial-time adversaries in the sense that real protocol and ideal functionality are only computationally indistinguishable. This specific requirement is already fulfilled by the protocol of Blum. Alternatively, one can demand, for example, universal composability of the protocol with respect to unbounded adversaries. This would then yield statistical or even perfect security. We show that whether such a protocol exists depending on the asymptotic behavior of m.

Finally, we clarify that in this paper, we consider coin toss protocols that do not necessarily guarantee output in case one party is corrupted. (We only require that when both parties are honest and all messages are delivered, both parties will give the same output.) Our definition aligns in a natural way with other simulation-based definitions (and in particular, universal composability) that usually do not guarantee output. Yet, our definition is weaker than, for example, the one considered by Cleve [11], Moran et al. [23] (which guarantees a uniformly distributed output in any case). Specifically, in our case, a dishonest party could abort the protocol (and potentially cause the other party not to output anything) once it learns that the result would have been unfavorable. We consider this weaker notion for the both assumed and achieved coin toss in a coin toss extension protocol. Hence, it is not clear to what extent even our negative results also hold for the stronger notion of coin toss. We note, however, that at least for stand-alone statistical security, we give a positive result (i.e., a coin toss extension protocol) that does guarantee output.

Table 1. Summary of our results on coin toss extension.

Security model	Level		
	Computational	Statistical	Perfect
Stand-alone simulatability	Yes	**Depends**	**No**
Universal composability	**Depends**	**No**	**No**

"Depends" means that the entry is "yes" for superlogarithmic m, and "no" otherwise

Our results are summarized in Table 1. A "yes" or "no" indicates whether a protocol for coin toss extension exists in that setting. "Depends" means that the answer depends on the size of the seed (the m-bit coin toss that is available by assumption). **Boldface** indicates novel results.

1.1. *Known Results in the Perfect and Statistical Case*

A folklore theorem states that (perfectly non-trivial) statistically secure coin toss is impossible from scratch (even in very lenient security models). Kitaev extended this result to protocols using quantum communication (cf. [1]). The task of *extending* a given coin toss was first investigated by Bellare et al. [4]. They presented a statistically secure protocol for extending a given coin toss (pre-shared among many parties using a verifiable secret sharing scheme), if less than $\frac{1}{6}$ of all parties are corrupted.

Their result does not apply to the two-party case.

1.2. *Our Results in the Perfect and Statistical Case*

Our results in the perfect case are most easily explained. For the perfect case, we show impossibility of *any* coin toss extension, no matter how (in)efficient. We show this for stand-alone simulatability (Corollary 11) and for universal composability (Corollary 16).

We first observe (and abstract in a helper lemma) that we may assume that any (not necessarily efficient) coin toss extension has a certain outer form, both in the stand-alone and UC security settings. Most interestingly, we may assume that the protocol partners run the m-bit coin toss only at the end of the protocol, after all party-to-party communication. A little more formally, we show that every coin toss extension protocol can be transformed into an inefficient one in which parties do not communicate any more after initiating the m-bit coin toss. Our transformation runs 2^m instances of the original protocol in parallel, one for each possible seed (i.e., outcome of the m-bit coin toss). Note that these instances can all be run without knowledge of the actual seed. The m-bit coin toss is only initiated after all instances have terminated, and the resulting seed selects the protocol instance whose outcome is returned. We will show in the proof of the helper lemma that this modified protocol provides a secure coin toss, assuming the original protocol is secure.

We now outline our argument for the stand-alone case. The impossibility of perfectly secure coin toss extension in the case of universal composability then follows directly from that in the case of stand-alone simulatability because universal composability implies stand-alone simulatability.

So assume a coin toss extension protocol that extends an m-bit coin toss to an n-bit outcome in the perfect stand-alone setting. Without loss of generality, we can assume that the protocol partners run the m-bit coin toss only at the end of the protocol, after all party-to-party communication. A little more formally, we show that every coin toss extension protocol (efficient or not, but perfectly secure in the stand-alone setting) can be transformed into an inefficient one in which parties do not communicate any more after initiating the m-bit coin toss. Our transformation runs 2^m instances of the original protocol in parallel, one for each possible seed (i.e., outcome of the m-bit coin toss). Note that these instances can all be run without knowledge of the actual seed. The m-bit coin toss is only initiated after all instances have terminated, and the resulting seed selects the protocol instance whose outcome is returned. We will show in the full proof that this modified protocol provides a secure coin toss, assuming the original protocol is secure.

Now a run of a protocol (of the form above) up to the point where the m-bit coin toss is started yields a set of 2^m possible outcomes, each with probability 2^{-m} (corresponding to the probability of each single possible seed). This protocol run without the last step (i.e., without the m-bit coin toss) can hence be interpreted as a finite game with total information. At the end of that game, there are at most 2^m possible candidates for the final outcome.

The goal of the game for a corrupted Alice is to end in a state in which the all-zero string has a probability greater than zero (and thus greater–equal to 2^{-m}), whereas a corrupted Bob will try to end in a state in which the all-zero string has probability 0. A finite game like this has a winning strategy, and either Alice can make the probability of the all-zero string nonzero (and thus $\geq 2^{-m} > 2^{-n}$) or Bob can make the probability of the all-zero string equal to zero. In either case, we have a contradiction to the perfect security of the coin toss extension (in which the probability of an all-zero outcome of the whole protocol is exactly 2^{-n}).

Now for the statistical case. When demanding only stand-alone simulatability, the situation depends on the number of already available common coins. Namely, we give an efficient protocol to extend m common coins to any polynomial number (in the security parameter), if m is superlogarithmic. The basic idea of the protocol is to have Alice and Bob each provide a bit string. The final outcome of the coin toss is then computed by applying a randomness extractor to (the concatenation of) both bit strings. The seed provided by the given m-bit coin toss will be used as the seed for the randomness extraction. (See Theorem 14 and its proof for details.)

A stand-alone coin toss extension even from m to $m + 1$ bits is impossible in the statistical case if the seed is too short, i.e., not superlogarithmic (Corollary 11). To sketch our argument, assume (for contradiction) a coin toss extension protocol that achieves statistical security for non-superlogarithmic m. Without loss of generality, we may again assume that the m-bit coin toss used as seed is only queried after all party-to-party communication.

Now consider a malicious protocol party (Alice) whose goal is to keep an all-zero outcome (of the constructed $(m + 1)$-bit coin toss) possible. More specifically, for each point p in a protocol run, we consider the probability of three possible conditions:

(1) Starting from p, an optimally playing Alice will "win," in the sense that in the last protocol step before the seed is chosen, the all-zero string has a probability not

equal to zero. This nonzero probability is taken only over the value of the seed and hence must be noticeable (because the seed is short). In particular, the probability for an all-zero outcome is noticeably different from an ideal coin toss.

(2) Starting from p, the protocol will not abort, and optimally playing Bob "wins," in the sense that the probability of the all-zero string is zero. Since m (and thus also $m + 1$) is not superlogarithmic, this also constitutes a noticeable difference to an ideal coin toss.

(3) Starting from p, the protocol will abort with a nonzero probability even if both parties are uncorrupted.

We will show by induction that one of the conditions will be true with "high" probability at every point p in the protocol. Hence, the protocol will abort with "high" probability in the uncorrupted case, or one of Alice or Bob will be able to "win" and hence cheat in the coin toss extension.

In the statistical universal composability setting, the situation is more clear: We show that there is no protocol with polynomially many rounds that extends m to $m + 1$ coins, no matter how large m is (Theorem 15). (Note, however, that our result does not exclude the existence of coin toss protocols that run in a superpolynomial number of rounds.)

As above, we may assume that the protocol obtains the seed in the last protocol step. The core of our proof rests on the following observation: Given the communication between the environment and the adversary up to the point when the seed is chosen, at least half of the strings from $\{0, 1\}^n$ are no longer possible. The protocol proceeds in polynomially many rounds, and at the end (before the seed is chosen), a superpolynomial amount of strings has become impossible. Hence, there must have been a single "critical message" excluding a superpolynomial number of strings. The environment lets the adversary corrupt a party that sends such a critical message. Then, the environment chooses at random whether the critical message is sent or replaced by a different message. Replacing the critical message then has a noticeable effect on the probability distribution of the final outcome; this effect cannot be mimicked by the simulator in the ideal model.

1.3. *Known Results in the Computational Case*

In [6], Blum gave a computationally secure coin toss protocol. In [14, Proposition 7.4.8], this protocol is shown to be stand-alone simulatable, and together with the sequential composition theorem [14, Proposition 7.4.3] for stand-alone simulatable protocols, this gives a computationally stand-alone simulatable protocol for tossing polynomially many coins. This makes coin toss extension trivial in that setting; one just ignores the m-bit coin toss and tosses n-bit from scratch.

In the computational universal composability setting, it has been shown in [8] that coin toss cannot be achieved from scratch. However, they showed that a sufficiently large common random string (CRS) implies (an arbitrary number of) ideal bit commitments. Such ideal bit commitments allow to implement a coin toss of arbitrary length (e.g., using Blum's protocol [6]). Thus, a sufficiently large CRS (and therefore, also a sufficiently large coin toss) can be extended to any polynomial length. However, it was unclear what the minimum size required from the CRS or the coin toss is.

Note that there is a subtle difference between the notion of a CRS and a coin toss. A CRS is randomness that is available to all parties at the beginning of the protocol, while with coin toss the randomness is only generated when all parties agree to run the coin toss. This makes the coin toss actually the stronger primitive, since in some situations, it is necessary to guarantee that not even corrupted parties learn the outcomes of the coin toss prior to a given protocol step.

In [11], the task of coin toss is considered in a scenario slightly different from ours.[1] Cleve [11] shows that in their setting, coin toss is generally not possible even against computationally limited adversaries. However, to the best of our knowledge, an *extension* of a given coin toss (in any setting) has not been considered so far in the computational setting.

1.4. *Our Results in the Computational Case*

We show that under suitable (and plausible) computational assumptions, it is possible to extend *any* coin toss of superlogarithmic length m. (Recall that if m is not superlogarithmic, we show—unconditionally—that coin toss extension is not possible. Hence, this positive result complements our negative result from Theorem 8, albeit under certain computational assumptions.) More specifically, [9] show that when assuming the existence of (polynomially secure) dense pseudorandom permutations, an m-bit coin toss (for *linear* m) can be used to implement (arbitrarily many) bit commitments. These bit commitments can then be used in the coin-tossing protocol of Blum [6] to derive arbitrarily many fresh random coins. We show (in Theorem 7) that suitably scaling the security parameter of this construction, an m-bit coin toss can be extended for any superlogarithmic m, assuming *exponentially strong* dense pseudorandom permutations. We leave it as an open problem to find coin toss extension protocols under weaker assumptions.

1.5. *CRS Extension*

A common random string (CRS) can be considered to be cryptographically weaker than a coin toss functionality. The random string of a coin toss functionality is chosen only after both parties have initiated the functionality. A CRS does not give such a guarantee and the adversary may know the value of the CRS from the start. Our coin toss extension protocol from Theorem 14 strongly depends on this guarantee; it is vital that the adversary cannot make his choices dependent on the seed before both parties have initiated the choice of the seed. Hence, the results for coin toss extension do not immediately apply to the task of CRS extension.

A natural question is under which conditions a given CRS can be extended. In this work, we give a characterization, summarized in Table 2.

Differences to the case of coin toss extension are printed in boldface. A CRS extension is impossible in the case of statistical stand-alone simulatability even for long seeds. To

[1] In [11], parties may not abort protocol execution without generating output. In contrast, in our setting, a party may abort at any time, for example, when detecting a cheating other party, or when it becomes clear that the overall output may be undesirable. We note that in this setting without aborts, any secure coin toss must be "fair," in the sense that a party is *guaranteed* to obtain a random output.

Table 2. Summary of our results on CRS extension.

Security type	Level		
	Computational	Statistical	Perfect
Stand-alone simulatability	Yes	**No**	No
Universal composability	Depends	No	No

"Depends" means that the entry is "yes" if and only if the seed has superlogarithmic length

show the impossibility, we look at the protocol after the CRS has been chosen. Given a concrete CRS s to extend, some bits of the protocol outcome should be undetermined at the start of the protocol. Otherwise, the resulting extended CRS would have at most the entropy of s. However, given a concrete CRS s, the situation for the undetermined bits is similar to a coin toss from scratch. These bits can be biased by Alice, or they can be biased by Bob. This proof illustrates the difference between a coin toss and a CRS. Recall that in the coin toss case, a given m-bit coin toss can be moved to the end of a coin toss extension protocol without loss of generality. We stress, however, that the choice of a CRS always happens at the beginning of the protocol.

For the computational case, the results correspond to the findings for coin toss extension (with roughly the same proofs).

1.6. *Notation*

- A function f is *negligible*, if for any $c > 0$, $f(k) \leq k^{-c}$ for sufficiently large k.
- A function f is *non-negligible* if it is not negligible, i.e., if there is a $c > 0$ such that $f(k) > k^{-c}$ for infinitely many k (not to be confused with noticeable).
- f is *noticeable*, if for some $c > 0$, $f(k) \geq k^{-c}$ for sufficiently large k. Note that functions exist that are neither negligible nor noticeable.
- f is *exponentially small*, if there exists a $c > 0$, such that $f(k) \leq 2^{-k^c}$ for sufficiently large k.
- f is *overwhelming*, if $1 - f$ is negligible.
- f is *polynomially bounded*, if for some $c > 0$, $f(k) \leq k^c$ for sufficiently large k.
- f is *polynomially large*, if there is a $c > 0$ such that $f(k)^c \geq k$ for sufficiently large k.
- f is *superpolynomial*, if for any $c > 0$, $f(k) > k^c$ for sufficiently large k.
- f is *superlogarithmic*, if $f/\log k \to \infty$ (i.e., $f \in \omega(\log k)$). It is easy to see that f is superlogarithmic if and only if 2^{-f} is negligible.
- f is *superpolylogarithmic*, if for any $c > 0$, $f(k) > (\log k)^c$ for sufficiently large k.
- f is *subexponential*, if for any $c > 0$, $f(k) < 2^{k^c}$ for sufficiently large k.

2. Security Definitions

In this section, we roughly sketch the security definitions used throughout this paper. We distinguish between two notions: stand-alone simulatability as defined in [14][2] and universal composability (UC) as defined in [7].

2.1. *Stand-alone Simulatability*

In [14], a definition for the security of two-party secure function evaluations is given (called *security in the malicious model*). We will give a sketch; for more details, we refer to [14].

A protocol consists of two parties that alternatingly send messages to each other. The parties may also invoke an ideal functionality as an oracle. In our case, the parties invoke a smaller coin toss to realize a larger one. We remark that the ideal functionality can only be invoked once. Thus, in our case, the parties have only access to one smaller coin toss.

We say the protocol π stand-alone simulatably realizes a probabilistic function f, if for any efficient adversary A that may replace none or a single party, there is an efficient simulator S such that for all inputs, the following random variables are computationally indistinguishable:

- *The real protocol execution* This consists of the view of the corrupted parties upon inputs x_1 and x_2 for the parties and the auxiliary input z for the adversary, together with the outputs I of the parties.
- *The ideal protocol execution* Here, the simulator first learns the auxiliary input z and possibly the input for the corrupted party (the simulator must corrupt the same party as the adversary). Then, he can choose the input of the corrupted party for the probabilistic function f, and the other inputs are chosen honestly (i.e., the first input is x_1 if the first party is uncorrupted, and the second input x_2 if the second party is). Then, the simulator learns the output I of f (we assume the output to be equal for all parties). It may now generate a fake view v of the corrupted parties. The ideal protocol execution then consists of v and I.

Of course, in our case, the probabilistic function f (the coin toss) has no input, so the above definition gets simpler.

What we have sketched above is what we call *computational* stand-alone simulatability. We further define *statistical* stand-alone simulatability and *perfect* stand-alone simulatability. In these cases, we do not consider efficient adversaries and simulators, but computationally unbounded ones. In the case of statistical stand-alone simulatability, we require the real and ideal protocol execution to be statistically (and not only computationally) indistinguishable, and in the perfect case, we even require these distributions to be identical.

[2] In fact, Goldreich [14] does not use the name stand-alone simulatability but simply speaks about *security in the malicious model*. We adopt the name stand-alone simulatability for this paper to be able to better distinguish the different notions.

2.2. *Universal Composability*

In contrast to stand-alone simulatability, universal composability [7] is a much stricter security notion. The main difference is the existence of an environment that may interact with protocol and adversary (or with ideal functionality and simulator) and try to distinguish between real and ideal protocol. This additional strictness brings the advantage of a versatile composition theorem (the UC composition theorem [7]). We only sketch the model here and refer to [7] for details.

A protocol consists of several machines that may (a) get input from the environment (also during the execution of the protocol), (b) give output to the environment (also during the execution of the protocol), and (c) send messages to each other.

The *real protocol execution* consists of a protocol π, an adversary $\mathcal{A}$, and an environment $\mathcal{Z}$. Here, the environment may freely communicate with the adversary, and the latter has full control over the network, i.e., it may deliver, delay, or drop messages sent between parties. We assume the authenticated model in this paper, so the adversary learns the content of the messages but may not modify it. When $\mathcal{Z}$ terminates, it gives an output. The adversary may choose to corrupt parties at any point in time.[3]

The *ideal protocol execution* is defined analogously, but instead of a protocol π, there is an *ideal functionality* $\mathcal{F}$, and instead of the adversary, there is a simulator $\mathcal{S}$. The simulator can also corrupt parties, but does not see any inputs/outputs exchanged between uncorrupted parties and the ideal functionality. If the simulator corrupts a party, the simulator can choose all inputs from that party into the functionality and get the corresponding outputs to that party. Uncorrupted parties simply act as relays (or "dummy parties") who forward inputs/outputs between $\mathcal{Z}$ and $\mathcal{F}$.

The *hybrid protocol execution* is defined like the real protocol execution, except that parties also have access to an ideal functionality (also called *hybrid* functionality in this context), in addition to their ability to communicate over the network. The adversary $\mathcal{A}$ controls the network in the same way as in the real protocol execution, but cannot control communication to/from the hybrid functionality (like in the ideal protocol execution). We remark that while Canetti [7] allows parties access to an unbounded number of instances of hybrid functionalities, here we are only interested in protocols that invoke and access at most one instance.

We say a protocol π universally composably (UC-)implements an ideal functionality $\mathcal{F}$ (or, if $\mathcal{F}$ is clear from the context: That π is universally composable), if for any efficient adversary $\mathcal{A}$, there is an efficient simulator $\mathcal{S}$, such that for all efficient environments $\mathcal{Z}$ and all auxiliary inputs z for $\mathcal{Z}$, the distributions of the output of $\mathcal{Z}$ in the real[4] and the ideal protocol executions are computationally indistinguishable.[5]

What has been sketched above is called *computational* UC. We further define *statistical* and *perfect* UC. In these notions, we allow adversary, simulator, and environment to be computationally unbounded machines. Further, in the case of perfect UC, we require

[3] If the adversary always only corrupts parties before the start of the protocol, it is called *static*. Otherwise, the adversary is called *adaptive*. The results in this paper hold for both types of adversaries, resp., corruptions.

[4] Or, if π uses a hybrid functionality, in the hybrid execution.

[5] In the original UC definition of [7], the output of $\mathcal{Z}$ is restricted to a single bit. However, our definition (with a potentially multi-bit output) will be more convenient for our purposes and is easily seen equivalent to the one-bit definition from [7].

the distributions of the output bit of $\mathcal{Z}$ to be *identical* in real/hybrid and ideal protocol executions.

2.3. *The Ideal Functionality for Coin Toss*

To describe the task of implementing a universally composable coin toss, we have to define the ideal functionality of n-bit coin toss. In the following, let n denote a positive integer-valued function.

Below is an informal description of our ideal functionality for a n-bit coin toss. First, the functionality waits for initialization inputs from both parties P_1 and P_2.[6] As soon as both parties have this way signaled their willingness to start, the functionality selects n coins in the form of an n-bit string κ uniformly and sends this κ to the adversary. (Note that a coin toss does not guarantee secrecy of any kind.)[7]

A more detailed description follows:

Ideal functionality CT_n (n-bit Coin Toss)

1. Wait until there have been "`init`" inputs from P_1 and P_2. Ignore messages from the adversary, but immediately inform the adversary about the `init`.
2. Select $\kappa \in \{0,1\}^n$ uniformly and send κ to the adversary. From now on:
   - on the first (and only the first) "`deliver to 1`" message from the adversary, send κ to P_1,
   - on the first (and only the first) "`deliver to 2`" message from the adversary, send κ to P_2.

We will consider protocols that implement CT_n (either in the sense of stand-alone or UC security). Unfortunately, the trivial protocol (that never generates any output) implements *any* functionality. (The corresponding simulator simply never delivers any outputs.) Hence, we require the following definition (see also [9] and [2, Sect. 5.1]):

Definition 1. A two-party protocol π is *non-trivial* if the probability is overwhelming that both parties generate identical outputs in a setting in which both parties are honest and all messages are delivered. π is *perfectly non-trivial* if that probability is zero.

Using CT_n, we can also formally express what we mean by *extending* a coin toss. Namely:

Definition 2. Let $n = n(k)$ and $m = m(k)$ be positive, polynomially bounded, and computable functions such that $m(k) < n(k)$ for all k. Then, a protocol is a *universally composable* $(m \to n)$*-coin toss extension protocol* if it is non-trivial and securely

[6] Here, our formalization of the coin toss functionality differs from that of [7]. They define a coin toss as a uniformly distributed common random string. In particular, their functionality does not wait for both parties to initialize the coin toss.

[7] If the functionality now sent κ directly and without delay to the parties, this behavior would not be implementable by any protocol (this would basically mean that the protocol output is immediately available, even without interaction). So the functionality lets the adversary decide when to deliver κ to each party. Note, however, that the adversary may not in any way influence the κ that is delivered.

implements CT_n by having access only to a single instance of CT_m. This security can be computational, statistical, or perfect.

2.4. *The Ideal Functionality for Common Random Strings*

In Sect. 5, we will also consider the extension of common reference strings. The difference between a common random string (CRS) and a coin toss is subtle but important. Both the coin toss and the CRS produce a uniform random bit string of a certain length and make it available to all parties. In the case of the CRS, this random bit string is publicly available at the beginning of the protocol. In the case of a coin toss, however, the random bit string is published only when both parties explicitly initialize the coin toss. Therefore, the coin toss is the stronger primitive: The security of a protocol can be based on the fact that the value of the coin toss is not known before a certain point in time. For example, the protocol given in the upcoming proof of Theorem 13 explicitly uses that the values a and b chosen by Alice and Bob do not depend on the seed s returned by the coin toss. Thus, this protocol would not be secure when using a CRS instead of a coin toss. In the following, we analyze which of the results given in the preceding sections apply to the problem of extending a CRS, that is, of producing a longer CRS from a shorter CRS.

Ideal functionality CRS_n (n-bit CRS)

1. In the first activation, select $\kappa \in \{0,1\}^n$ uniformly and send κ to the adversary.
2. When receiving `getcrs` from party P_1 or P_2, inform the adversary.
3. When receiving `deliver to` i from the adversary for some $i \in \{1,2\}$, and P_i did already send `getcrs`, send κ to P_i.

Definition 3. Let $n = n(k)$ and $m = m(k)$ be positive, polynomially bounded, and computable functions such that $m(k) < n(k)$ for all k. Then, a protocol is a *universally composable* $(m \to n)$*-CRS extension protocol* if it securely and non-trivially implements CRS_n by having access only to CRS_m. This security can be computational, statistical, or perfect.

2.5. *On Unbounded Simulators*

Following [3], we have modeled statistical and perfect stand-alone and UC security using computationally unbounded simulators. Another approach is to require the simulators to be polynomial in the running time of the adversary. Our results also hold in such a model. For the impossibility results, this is straightforward, since the security notion gets stricter when the simulators become more restricted. The only possibility result for statistical/perfect security is given in Theorem 14. There, the simulator we construct is in fact polynomial in the runtime of the adversary.

In the following sections, we investigate the existence of such coin toss extension protocols, depending on the desired security level (i.e., computational / statistical / perfect security) and the parameters n and m.

3. Coin Toss Extension: The Computational Case

3.1. *Universal Composability*

In this section, we present two positive results (combined in Theorem 7) and a negative result (Theorem 8). Our positive results state that as long as m is superlogarithmic, we can achieve coin toss extension (under a computational assumption, whose strength depends on m). Our negative result states that for non-superlogarithmic m, no coin toss extension is possible (unconditionally).

In the following, we start with our positive results and first need to introduce the corresponding computational assumption. Specifically, we need the assumption of (doubly-) enhanced trapdoor permutations with pseudorandom public keys (called ETDs henceforth). Roughly, these are trapdoor permutations with the additional properties that (i) one can sample an image of the permutation in an oblivious fashion, i.e., even given the coins used for sampling of the image, it is infeasible to invert the function, (ii) one can sample a uniform preimage along with the *random coins* needed to sample the corresponding image, and (iii) the public keys are computationally indistinguishable from random strings.

We inherit the assumption that ETDs exist from [9] (they use it for the case of a uniform CRS). Although we are not aware of any concrete candidates for ETDs, that assumption seems plausible.

We will show that when ETDs exist, then so do efficient protocols for extending a suitably long coin toss. The idea is simple: First, a suitably long coin toss trivially implements a common random string (CRS), from which we can bootstrap a UC-secure multi-use bit commitment protocol using the techniques of [9]. (This is captured in Lemmas 5 and 6.) Next, the UC-secure bit commitment protocol can be used to implement any polynomially long coin toss, using Blum's coin toss protocol [6]. (This is detailed in Theorem 7.)

We start off with the definition of ETDs. The definition of ETDs follows that of doubly enhanced trapdoor permutations in [15]; only the requirement for *pseudorandom public keys* has been added.

Definition 4. (*Doubly enhanced trapdoor permutations with pseudorandom public keys*)

A system of *doubly enhanced*[8] *trapdoor permutations with pseudorandom public keys (ETD)* consists of the following efficient algorithms: a key generation algorithm I that (given security parameter k) generates public keys pk and corresponding trapdoors td (we treat pk and td as efficiently computable functions to facilitate notation) and a domain sampling algorithm S that given pk outputs an element in the domain of pk. Additionally, the ETD defines a set of *valid public keys*. I, S must satisfy the following conditions:

For any non-uniform probabilistic polynomial-time algorithm A, there is a negligible function μ such that the following conditions are satisfied:

[8] We require *doubly* enhanced trapdoor permutations because they are actually required by current constructions of NIZK proofs, see [15].

– *Permutations* $\mathsf{Pr}\left[(pk, td) \leftarrow I(1^k) : pk \text{ is a valid public key and } td = pk^{-1}\right] \geq 1 - \mu(k)$, and any valid public key is a permutation.
– *Almost uniform sampling.* For any valid public key pk in the range of $I(1^k)$, the statistical distance between the output of $S(pk)$ and randomly chosen elements in the domain (=range) of pk is bounded by $\mu(k)$.
– *Enhanced hardness* For all $k \in \mathbb{N}$

$$\mathsf{Pr}\left[(pk, td) \leftarrow I(1^k), y \leftarrow S(pk), x' \leftarrow A(1^k, pk, y, r) : pk(x') = y\right] \leq \mu(k)$$

Here, r denotes the randomness used by S.
– *Doubly enhanced* There exists an efficient algorithm that, on input a valid public key pk, outputs (x, r) with $pk(x) = S(pk)$, where r denotes the randomness used by S and is distributed uniformly. In other words, it is efficiently possible to sample a preimage x along with the random coins needed to choose the image $pk(x)$.
– *Pseudorandom public keys* There is a polynomially bounded, efficiently computable function s (not depending on A) such that

$$\left|\mathsf{Pr}\left[(pk, td) \leftarrow I(1^k) : A(1^k, pk) = 1\right] - \mathsf{Pr}\left[pk \leftarrow \{0,1\}^{s(k)} \text{ uniformly} : A(1^k, pk) = 1\right]\right| \leq \mu(k).$$

An ETD is *exponentially hard* if for every subexponential-time A, there exists an exponentially small μ such that all of the above conditions (i.e., permutations, almost uniform sampling, enhanced hardness, doubly enhanced, and pseudorandom public keys) hold.

Lemma 5. *There is a constant $d \in \mathbb{N}$ such that the following holds for all polynomially bounded functions s computable in time polynomial in k:*

Assume that ETD exists such that the size of the circuits describing the ETD is bounded by $s(k)$ for security parameter k.[9]

Then, there is a protocol π using a uniform common random string (CRS) of length $s(k)^d$ such that π securely UC-realizes a bit commitment that can be used polynomially many times.

Proof. The main work (i.e., finding the protocol and proving its security) has been done in [9]. It is left to show that for their construction, a CRS of length $poly(s)$ is sufficient. By $poly(s)$, we mean a polynomially bounded function in s that is independent of the chosen ETD. (In [9], it is only shown that a CRS of length $p(k)$ is sufficient, where k is the security parameter and p a polynomial depending on the ETD.)

In [9], there is a protocol UAHC that, assuming a CRS and the existence of ETD, implements multiple commitments.[10] The CRS is assumed to contain the following: (i)

[9] By the size of the circuits, we means the total size of the circuits describing both the key generation and the domain sampling algorithm. Note that then trivially also the size of the resulting keys and the amount of randomness used by the domain sampling algorithm are bounded by $s(k)$.

[10] Canetti et al. [9] state their result with respect to enhanced, not doubly enhanced trapdoor permutations. This is due to the fact that, at the time, it was believed that enhanced trapdoor permutations are sufficient

a random image under a one-way function f_k (that depends on the security parameter k), (ii) a public key for a semantically secure cryptosystem E, and (iii) a public key for a CCA2-secure cryptosystem E_{cca}. We discuss how to instantiate f_k, E, E_{cca} so that we get a CRS that is indistinguishable from uniform.

The one-way function f may be constructed from the ETD as follows: f interprets its input r as randomness to be used in the ETD key generation algorithm and outputs the resulting public key. Then, for security parameter k, the images of f have length $s_1 \leq s$ (since they are public keys). Further, since the public keys are indistinguishable from uniform randomness by definition of the ETD, random images of f are computationally indistinguishable from s_1-bit randomness.

Second, a semantically secure cryptosystem E can be constructed from the ETD using the construction from [16,17]. Then, the public key for E is just a public key for the ETD. It follows that the length of the public keys is $s_1(k)$, and random public keys are indistinguishable from s_1-bit randomness.

The construction of E_{cca} is more involved but still standard. Specifically, we use the construction by Dolev et al. [12]. For this, we first need a non-interactive zero-knowledge proof system (NIZK) to prove consistency of a ciphertext. For instance, [13, Constructions 4.10.4 and 4.10.7] together with the additional remarks in [15] present a suitable scheme, based on doubly enhanced trapdoor permutations. We will now examine the size of the CRS needed for that protocol. To prove a statement that is described by a circuit of size s_2, the CRS consists—for one iteration of the proof—of $poly(s_2)$ commitments to random bits using a trapdoor permutation. The length of each commitment is $O(s)$ since s bounds the size of the circuits describing the trapdoor permutation scheme. To guarantee soundness, $poly(s_2) \cdot m$ parallel executions of the scheme are necessary (using the same trapdoor permutation, see [13, Construction 4.10.4]), where m is a superlogarithmic function in the security parameter. So if we choose $m := s$, the length of the CRS used by the NIZK scheme is bounded by $poly(s(k) + s_2(k))$. The CRS consists of images of uniformly random preimages under a permutation; thus, it is uniformly random.

Another ingredient we need is a universal family of one-way hash functions. In [25] (see also [18,22]), a scheme is presented that converts a one-way function f into a universal family of one-way hash functions. Here, the image of the hash function has length $s_3 \in poly(s_4)$, where s_4 is the length of the images of f. And if the one-way function is keyless, so is the hash function. If we use the keyless one-way functions f constructed above, $s_4 \leq s$, and the one-way hash is keyless.

Now, we come back to the construction of E_{cca}. In this construction, the public key consists of (i) a hash function h from the abovementioned family (s_3 bit), (ii) $2s_4$ public keys for a semantically secure encryption scheme, say the scheme E constructed above ($2s_4 s_1$ bit), and (iii) a CRS for the NIZK scheme above to show a statement that can be described by a circuit of size polynomial in $2s_4$ and the size of the circuits describing the trapdoor permutation scheme (that is bounded by s). So the CRS has a length of at

Footnote 10 continued
for constructing non-interactive zero-knowledge proofs. It was, however, pointed out in [15] (following an observation by Jonathan Katz) that this is not the case. In particular, we stress that also [9] actually require the existence of *doubly* enhanced trapdoor permutations. In case of a uniform CRS, in fact ETDs according to Definition 4 are required.

most $poly(s+s_4)$ bit. Putting this together, and noting that $s_4 \leq s$, we see that the public key of E_{cca} has a length in $s_3 + 2s_4s_1 + poly(s+s_4) = poly(s)$.

Since the key of the hash function we constructed is a zero bit string (the hash function is keyless), and the public key of E as well as the CRS of the NIZK scheme is indistinguishable from uniform, the public key of E_{cca} is also indistinguishable from uniform.

Finally, the protocol UAHC from [9] uses a CRS consisting of a public key for E, a public key for E_{cca}, and an image of f. By our calculations above, the total length of that CRS lies in $poly(s)$, and the CRS is indistinguishable from uniform.

Let π be the protocol that results from UAHC by using a uniformly random string of length $poly(s)$. Since the new CRS is indistinguishable from the old CRS, and since UAHC is a UC-secure commitment, the protocol π is also UC-secure commitment with uniform CRS. □

Lemma 6. *Let $s(k)$ be a polynomially bounded function that is computable in time polynomial in k.*

Assume one of the following holds:

- *ETD exists and s is a polynomially large function.*
- *Exponentially hard ETD exists and s is a superpolylogarithmic function.*

Then, there also exists an ETD, such that the size of the circuits describing the ETD is bounded by $s(k)$ for security parameter k.

Proof. This is shown by scaling the security parameter of the original ETD. Let I be the key generation algorithm and S be the sampling algorithm of the ETD.

Since I and S are efficient algorithms, there is a $c \in \mathbb{N}$ such that the size of the circuits of (I, S) is bounded by k^c. Then, set $\tilde{s}(k) := \lfloor s(k)^{1/c} \rfloor$. Obviously, if s is superpolylogarithmic or polynomially large, respectively, then so is $\tilde{s}$. We now construct a new scheme (I', S') as follows: $I'(k') := I(\tilde{s}(k'))$ and $S' := S$. Then, for security parameter k', the circuits of (I', S') have size $\tilde{s}(k')^c \leq s(k')$, as required. It is left to show that (I', S') is a system of ETD.

We will use the following notation: When talking about the original ETD (I, S), we will use the names from Definition 4 (e.g., A, k, μ). When talking about (I', S'), we will add a prime (e.g., A', k', μ').

Let a polynomial-time algorithm A' be given. We then construct a machine A as follows: Upon input 1^k, A chooses k' to be a uniform k' with $\tilde{s}(k') = k$ (i.e., k' is uniformly chosen from the set $\tilde{s}^{-1}(\{k\})$).

After k' is chosen, A runs $A'(1^{k'})$.

As we will show below, A runs in polynomial time (or subexponential time in the case of exponentially hard ETD). So there is a negligible (or exponentially small) μ with respect to which all conditions in Definition 4 hold. Let

$$\mu'(k') := |\tilde{s}^{-1}(\{\tilde{s}(k')\})| \cdot \mu(\tilde{s}(k')).$$

Then, by construction, all the conditions in Definition 4 also hold for A', μ', and the modified system (I', S') (to see this, simply substitute $\tilde{s}(k')$ for k and take into account the uniform choice of k' over $\tilde{s}^{-1}(k)$). Since μ' is negligible if μ is negligible

or exponentially bounded, respectively (as we will show below), it follows that (I', S') is a system for ETD. It is left to show that A runs in polynomial time (or subexponential time in the case of exponentially hard ETD) and that μ' is negligible.

First, we show that A runs in polynomial/subexponential time in k. Since A simulates A' in time polynomial in $\tilde{k} := \max \tilde{s}^{-1}(\{k\})$, it is sufficient to show that $\tilde{k}$ is polynomially bounded (or subexponential, respectively) in k. We distinguish two cases. Case 1: If $\tilde{s}$ is polynomially large, then there is a d such that $\tilde{s}(k')^d \geq k'$ for almost all k'. Then, we have $\tilde{s}(k') \geq k'^{1/d}$ and then $\tilde{k} = \min \tilde{s}^{-1}(\{k\}) \leq k^d$ for almost all k.

Case 2, $\tilde{s}$ is superpolylogarithmic: Let $d \in \mathbb{N}$ be arbitrary. Since $\tilde{s}$ is superpolylogarithmic, there exists $K_d \in \mathbb{N}$ with $\tilde{s}(k') \geq (\log k')^d$ for all $k' \geq K_d$. Now let $k \in \mathbb{N}$ be arbitrary and $\tilde{k} := \max \tilde{s}^{-1}(\{k\})$. By definition of K_d, we must have $(\log \tilde{k})^d \leq \tilde{s}(\tilde{k}) = k$ or $\tilde{k} < K_d$. If $(\log \tilde{k})^d \leq k$, then $\tilde{k} \leq 2^{k^{1/d}}$. Thus, $\tilde{k} \leq \max\{2^{k^{1/d}}, K_d\}$, and so $\tilde{k} \leq 2^{k^{1/d}}$ for sufficiently large k. Since $d \in \mathbb{N}$ was arbitrary, this shows that $\tilde{k}$ is subexponential in k.

It remains to show that μ' is negligible. As a preparation, note that

$$\mu'(k') = |\tilde{s}^{-1}(\{\tilde{s}(k')\})| \cdot \mu(\tilde{s}(k')) \leq \max \tilde{s}^{-1}(\{\tilde{s}(k')\}) \cdot \mu(\tilde{s}(k')).$$

Now in the first case (where $\tilde{s}$ and s are polynomially large), μ is negligible. Since $\tilde{s}$ is polynomially large and computable in polynomial time, there exist $d, e \in \mathbb{N}$ such that for sufficiently large k', it is $k'^{1/d} \leq \tilde{s}(k') \leq k'^e$. Then,

$$\mu'(k') \leq \max \tilde{s}^{-1}(\{\tilde{s}(k')\}) \cdot \mu(\tilde{s}(k')) \leq k'^{ed} \cdot \mu(k'^{1/d})$$

is negligible.

In the second case, $\tilde{s}$ is superpolylogarithmic and μ is exponentially small. Hence, there exists $d > 0$ with $\mu(k) \leq 2^{-k^d}$ for all sufficiently large k. Furthermore, since $\tilde{s}$ is superpolylogarithmic, for every $c > 0$, there exists a $D_c \in \mathbb{N}$ with $\tilde{s}(k'') \geq (\log k'')^c - D_c$ for all k''. The latter fact implies that $k'' \leq 2^{(\tilde{s}(k'')+D_c)^{1/c}}$ for any k''. Hence, with $k'' := \max \tilde{s}^{-1}(\{\tilde{s}(k')\})$, we get

$$\max \tilde{s}^{-1}(\{\tilde{s}(k')\}) \leq 2^{(\tilde{s}(k')+D_c)^{1/c}} \tag{1}$$

for any k'. We now show that $\mu'(k')$ is negligible. To this end, we set $c := 3/d$ and thus derive

$$\begin{aligned}\mu'(k') &\leq \max \tilde{s}^{-1}(\{\tilde{s}(k')\}) \cdot \mu(\tilde{s}(k')) \overset{(1)}{\leq} 2^{(\tilde{s}(k')+D_c)^{d/3}} \cdot 2^{-\tilde{s}(k')^d} \\ &\leq 2^{\tilde{s}(k')^{d/2}} \cdot 2^{-\tilde{s}(k')^d} = 2^{-\tilde{s}(k')^{d/2}} \overset{(*)}{\leq} 2^{-(\log k')^2} = k'^{-\log k'}\end{aligned}$$

for sufficiently large k', where $(*)$ follows from the assumption that $\tilde{s}$ is superpolylogarithmic. Thus, $\mu'(k')$ is negligible in k' as desired. □

Theorem 7. *Let $n = n(k)$ and $m = m(k)$ be polynomially bounded and efficiently computable functions. Assume one of the following conditions holds:*

- Upon input (*init*), party P_1 commits to n random bits r_1.
- Upon input (*init*), and after P_1 has committed itself, party P_2 sends n random bits r_2 to P_1.
- Finally, P_1 unveils the bits r_1.
- Both parties output the n-bit string $r = r_1 \oplus r_2$, where $\oplus$ denotes the bit-wise exclusive or.

Fig. 1. Coin extension protocol from Theorem 7.

- *m is polynomially large and ETD exists, or*
- *m is superpolylogarithmic and exponentially hard ETD exists.*

Then, there is a polynomial-time computationally universally composable protocol π for $(m \to n)$-coin toss extension.

Proof. Let d be as in Lemma 5. If m is polynomially large or superpolylogarithmic, then $s := m^{1/d}$ is polynomially large or superpolylogarithmic, respectively. So, by Lemma 6, there is ETD, such that the size of the circuits describing the ETD is bounded by $s = m^{1/d}$. Then, by Lemma 5, there is a UC-secure protocol for implementing n-bit commitments using an $(m^{1/d})^d = m$-bit CRS.

Given n-bit commitments, the following protocol π UC-realizes an n-bit coin toss (based on the protocol of [6]):

Now consider the coin toss extension protocol from Fig. 1. It is easy to see that this protocol UC-realizes an n-bit coin toss. We sketch the simulator $\mathcal{S}$: As soon as all uncorrupted parties got input (*init*), $\mathcal{S}$ learns what value r the ideal n-bit coin toss has. When P_1 is or gets corrupted, $\mathcal{S}$ learns the value r_1 as soon as P_1 commits, so the simulated r_2 can be chosen as $r_1 \oplus r$. When P_2 is or gets corrupted, but P_1 is uncorrupted at least during the commitment to r_1, the simulator $\mathcal{S}$ unveils value r_1 to $r_2 \oplus r$. In the case that both parties get corrupted, the environment does not learn the value from the ideal coin toss, so the simulator can simply choose it to be $r_1 \oplus r_2$.

Furthermore, an m-bit CRS can be trivially implemented using an m-bit coin toss. Using the UC composition theorem [7], we can put the above constructions together and get a protocol that UC-realizes an n-bit coin toss using an m-bit coin toss. □

Note that given stronger, but possibly unrealistic assumptions, the lower bound for m in Theorem 7 can be decreased. If we assume that for any superlogarithmic m, there is ETD such that the size of their circuits is bounded by $m^{1/d}$ (where d is the constant from Lemma 5), we get coin toss extension even for superlogarithmic m (using the same proof as for Theorem 7, except that instead of Lemma 6, we use the stronger assumption).

However, we cannot expect an even better lower bound for m, as the following theorem shows:

Theorem 8. *Let $n = n(k)$ and $m = m(k)$ be functions with $n(k) > m(k) \geq 0$ for all k, and assume that m is not superlogarithmic (i.e., 2^{-m} is non-negligible). Then, there is no non-trivial polynomial-time computationally universally composable protocol for $(m \to n)$-coin toss extension.*

We first give a proof sketch. We note that our proof generalizes a similar result from [8] (that shows Theorem 8 for $m = 0$). Canetti [8] argue that a hypothetical simulator for coin toss would have to be able to "convince" the other party of an arbitrary outcome of the coin toss. We show that a similar property holds even when an ideal (but short) m-bit coin toss is available.

More specifically, first, we recall how the impossibility of a universally composable coin toss is shown in the case that we have no seed (i.e., without the functionality CT_m). Assume for contradiction that a protocol π with parties P_1 and P_2 exists such that π implements CT_n. (Here, n is as in the theorem.) Then, assume an adversary $\mathcal{A}_1$ that corrupts P_1 and simply reroutes all communication with P_1 to the environment. (E.g., messages sent by P_2 are forwarded to the environment; cf. also the left-hand side of Fig. 2.) Assume an environment $\mathcal{Z}_1$ that internally simulates an instance $\overline{P}_1$ of P_1 and instructs $\mathcal{A}_1$ to forward the messages produced by the simulated $\overline{P}_1$. The outputs made by $\overline{P}_1$ and P_2 we call $\overline{\kappa}_1$ and κ_2, respectively. Since the network consisting of $\mathcal{Z}_1$, $\mathcal{A}_1$, and P_2 essentially is an honest execution of π, we have that with overwhelming probability, $\overline{\kappa}_1$ and κ_2 are n-bit strings and $\overline{\kappa}_1 = \kappa_2$. $\mathcal{Z}_1$ outputs 1 iff $\overline{\kappa}_1 = \kappa_2$; thus, $\mathcal{Z}_1$ outputs 1 with overwhelming probability when running with π and $\mathcal{A}_1$ as above.

Since we assume that π is universally composable, there is a simulator $\mathcal{S}_1$ that simulates the adversary $\mathcal{A}_1$. That is, $\mathcal{Z}_1$ cannot distinguish between $\mathcal{A}_1$ with P_2 (the real model) and $\mathcal{S}_1$ with CT_n (the ideal model). Since $\mathcal{A}_1$ just forwards messages from P_2, the simulator $\mathcal{S}_1$ effectively produces a simulation of P_2's messages. Furthermore, in the ideal model, $\mathcal{Z}_1$ gets the n-bit string κ_2 from CT_n. Since $\mathcal{Z}_1$ cannot distinguish between the real and the ideal models, we have that $\overline{\kappa}_1 = \kappa_2$ with overwhelming probability also in the ideal model. This implies that $\mathcal{S}_1$ is a machine that manages to make $\overline{P}_1$ (which is identical to the honest party P_1) output an externally given n-bit string κ_2. This, however, violates the assumption that P_1 is part of a secure coin toss protocol. In a secure coin toss protocol, $\mathcal{S}_1$ would succeed only with probability 2^{-m} in making P_1 output κ_2. Thus, our assumption that π was a universally composable coin toss protocol is false.

Now consider the case where we additionally have an m-bit seed ω given by the ideal functionality CT_m used in the real model by P_1 and P_2. In this case, the simulator $\mathcal{S}_1$ is allowed to simulate the value ω. Thus, $\mathcal{S}_1$ now is a machine that can make the honest party P_1 output an externally given n-bit string κ_2 if $\mathcal{S}_1$ is allowed to choose the seed ω. If $\mathcal{S}_1$ may not choose the seed ω, it will only succeed if the seed ω accidentally is the one that $\mathcal{S}_1$ would have chosen. This happens with probability 2^{-m}. Thus, $\mathcal{S}_1$ manages to make P_1 output an externally given value κ with probability 2^{-m} (up to a negligible error). However, since π is a secure coin toss protocol, $\mathcal{S}_1$ should succeed with probability at most 2^{-n} (up to a negligible error). Since the difference between 2^{-n} and 2^{-m} is non-negligible (as n is not superlogarithmic), it follows $\mathcal{Z}_1$ can distinguish between the real model and the ideal model with $\mathcal{S}_1$. Thus, π is not universally composable.

We proceed with the full proof.

Proof (of Theorem 8). We use the notation from the proof sketch. So assume for contradiction that π, using CT_m, implements CT_n. We start with a network C_1 of machines as in a real protocol run with corrupted P_1. More specifically, C_1 consists of a party P_2, a helping coin toss functionality CT_m, an adversary $\mathcal{A}_1$ that takes the role of a corrupted

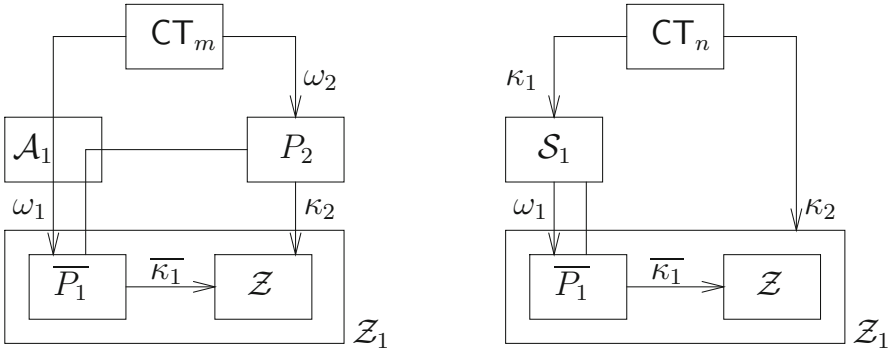

Fig. 2. *Left* Real protocol C_1 with corrupted P_1 and relaying $\mathcal{A}_1$. *Right* Ideal protocol C_2 with corrupted P_1 and simulator $\mathcal{S}_1$.

P_1, and an environment $\mathcal{Z}_1$. Note that the corrupted party P_1 has been removed, since it is taken over by the adversary.

The machine $\mathcal{A}_1$ simply relays the connections of the corrupted P_1 to $\mathcal{Z}_1$. That is, every message sent from CT_m or P_2 to the corrupted P_1 is forwarded to $\mathcal{Z}_1$, and $\mathcal{A}_1$ lets $\mathcal{Z}_1$ send messages to CT_m or P_2 in the name of P_1. Now $\mathcal{Z}_1$ in turn internally simulates an instance $\overline{P_1}$ of party P_1 and lets this simulation take part in the protocol through $\mathcal{A}_1$. Additionally, $\mathcal{Z}_1$ simulates a machine $\mathcal{Z}$. That machine $\mathcal{Z}$ only gives "`init`" inputs to the parties $\overline{P_1}$ and P_2 and then collects their outputs. At the end of the execution, $\mathcal{Z}$ gives output 1 iff both parties give output and both outputs are identical. The output is passed through by $\mathcal{Z}_1$. The situation is depicted in Fig. 2.

Our first claim is that in runs of this network C_1, eventually identical $\overline{\kappa_1}$ and κ_2 are observed by $\mathcal{Z}_1$ with overwhelming probability. Indeed, by definition of CT_n, in an ideal protocol run with no corruptions, the outputs κ_1 and κ_2 must be identical *if* both are output. Since π UC-implements CT_n, this must also hold with overwhelming probability in runs of the real protocol without corruptions. Since protocol π is non-trivial, in such a case output is guaranteed, and we have thus $\kappa_1 = \kappa_2$ with overwhelming probability. This carries over to C_1, since C_1 is formed from an uncorrupted real protocol simply by relaying some messages through $\mathcal{A}_1$ and by regrouping machines. So in C_1, $\mathcal{Z}_1$ gives output 1 with overwhelming probability.

Now since π UC-implements CT_n, there must be a simulator $\mathcal{S}_1$ in the ideal setting with CT_n that simulates attacks carried out by $\mathcal{A}_1$. In our situation (depicted in Fig. 2), this simulator must in particular achieve that $\overline{\kappa_1} = \kappa_2$ with overwhelming probability. In other words, $\mathcal{S}_1$ must "convince" the simulation of P_1 to output the κ_1 that was chosen by the ideal CT_n. To this end, $\mathcal{S}_1$ may make up an initial seed ω_1 from a machine CT_m that is actually not present in the ideal model. Also, $\mathcal{S}_1$ may make up suitable responses from a faked party P_2 (that is also not present in the ideal model) in communication with $\overline{P_1}$. Call this network (consisting of $\mathcal{S}_1$, CT_n, and $\mathcal{Z}_1$) C_2. Since the probability that $\mathcal{Z}$ gave output 1 was overwhelming in C_1, the same holds for C_2 by the definition of UC security.

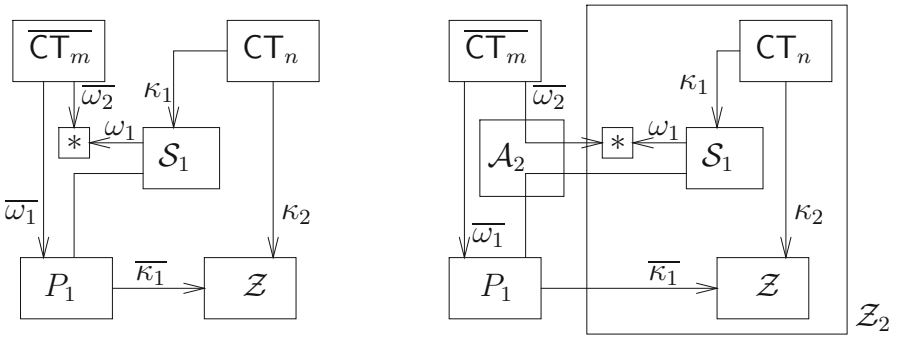

Fig. 3. *Left* Modification C_3 of C_2. *Right* Regrouping C_4 of C_3.

Now we modify network C_2. First, we regroup machines and make the simulation $\overline{P_1}$ of P_1 a machine of its own. (This machine is then identical to P_1.) Then, we introduce a new machine CT_m that ideally selects and delivers an $\overline{\omega_1}$ to $\overline{P_1}$. (In C_2, $\overline{P_1}$ got that value from $\mathcal{S}_1$.) Both the seed $\overline{\omega_2}$ output by $\overline{\mathsf{CT}_m}$ and the seed ω_1 output by $\mathcal{S}_1$ are simply collected by a dummy machine $*$ that discards them. The resulting network is called C_3 and depicted in Fig. 3.

The networks C_2 and C_3 provide completely identical views for P_1 when $\omega_1 = \overline{\omega_1}$ in C_3. This again happens with probability 2^{-m} by definition. Since in C_2, the environment $\mathcal{Z}$ gave output 1 with some overwhelming probability p, it follows that in C_3 the probability is at least $2^{-m}(1-p) = 2^{-m} - \mu$ for some negligible μ.

Now comes the crucial part: We combine $\mathcal{Z}$, $\mathcal{S}_1$, CT_n, and the dummy machine $*$ (that is to say, all machines but P_1 and $\overline{\mathsf{CT}_m}$) into a protocol environment $\mathcal{Z}_2$. A new real adversary is added that only relays the connection between $\mathcal{S}_1$ and P_1 and the connection between $*$ and $\overline{\mathsf{CT}_m}$.

This regrouping of machines gives a new network C_4 (cf. Fig. 3). Note that C_4 is only a regrouping of C_3 (followed by the insertion of a machine $\mathcal{A}_2$) that just forwards messages, and hence it still holds that $\mathcal{Z}$ gives output 1 with probability $2^{-m} - \mu$. Note further that C_4 actually is the network in which $\mathcal{A}_2$ corrupts the party P_2 in the real protocol π and runs with environment $\mathcal{Z}_2$.

Now since π UC-implements CT_n, there must be a simulator $\mathcal{S}_2$ that in an ideal setting with $\overline{\mathsf{CT}_n}$ simulates the situation from network C_4. (Here, we use the different name $\overline{\mathsf{CT}_n}$ only to avoid conflicting names with the CT_n-instance inside $\mathcal{Z}_2$). This simulator simulates attacks carried out by $\mathcal{A}_2$ on the real protocol. The network consisting of $\mathcal{S}_2$, $\mathcal{Z}_2$, and $\overline{\mathsf{CT}_n}$ we call C_5; see Fig. 4.

In particular, this simulator achieves that the probability of $\mathcal{Z}$ outputting 1 is negligibly less than in the real setting C_4. However, in C_4 this probability has a lower bound of $2^{-m} - \mu$ for some negligible μ. On the other hand, in the ideal setting of C_5, both $\overline{\kappa_1}$ and κ_2 are chosen in an ideal manner as independent uniform n-bit strings by instances of the functionality CT_n. So the probability that $\overline{\kappa_1} = \kappa_2$ in C_5 is at most 2^{-n} (note that it is also possible that no output is generated), and therefore the probability of an 1-output in C_5 is bounded by 2^{-n}. So the difference between the probabilities that $\mathcal{Z}$ outputs 1

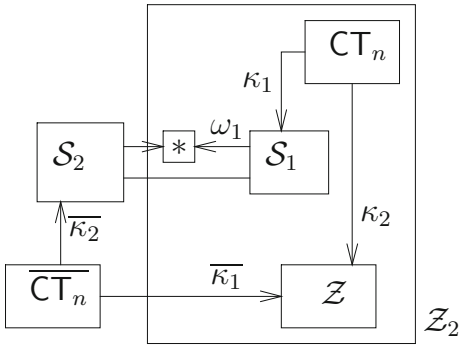

Fig. 4. Ideal setting C_5 corresponding to the real setting of C_4.

in runs of C_4 and C_5 is at least

$$2^{-m} - \mu - 2^{-n} \geq 2^{-m-1} - \mu$$

which is not negligible since m is not superlogarithmic. This contradicts the fact derived above that the difference of the probabilities is negligible. So π cannot securely implement an n-bit coin toss. □

4. Coin Toss Extension: The Statistical and the Perfect Case

4.1. *A Technical Lemma*

We first show that we can make certain simplifying assumptions about the protocols we consider.

Lemma 9. *If there is a non-trivial (computationally, statistically, or perfectly) stand-alone secure (resp. universally composable) protocol for $(m \to n)$-coin toss extension, then there is also (a potentially inefficient) one in which*

- *in the honest case, both parties either output the same bit string $z \in \{0,1\}^n$, or both parties output nothing (in which case, we write $z = \perp$),*
- *this output z (for the honest parties) is a deterministic function of the messages sent and the value s of the m-bit coin toss,*
- *each party sends in each protocol run at most one message to CT_m, and this is always an "*`init`*" message,*
- *the internal state of each of the two parties consists only of the messages exchanged (with the other party and CT_m) so far,*[11] *and*
- *after P_i sends "*`init`*" to CT_m, it does not further communicate with P_{3-i} (for $i = 1, 2$ and in case of no corruptions).*

[11] In particular, the randomness required to produce a message is always chosen directly before sending that message.

Proof. First, we modify a given protocol as follows, to enforce the first two requirements: Each party sends a confirmation messages at the end, where it tells the other party what it is going to output. If these exchanged values do not match, both parties output nothing. The same modification also achieves that the outcome is a deterministic function of the protocol transcript and the used m-bit coin toss s. This modification only suppresses outputs in certain cases and thus can be simulated perfectly, without any simulation error.

Next, straightforward syntactic modifications show that we can assume that each party sends at most one message to CT_m in each run, and this is always an "`init`" message. (Other messages to CT_m would be ignored anyway.) An application of Lemma 21 further shows we can assume that the internal state each party consists only of the messages exchanged so far with the other party and CT_m. The remaining transformation modifies π such that no further communication between P_1 and P_2 is necessary after CT_m has been invoked.

First, we change each P_i (for $i \in \{1, 2\}$) so as to signal the other party P_{3-i} before it sends an "`init`" message to CT_m. Then, P_i proceeds to send "`init`" to CT_m only after it has received an acknowledgement message from P_{3-i}. We call the modified protocol π_1. It is easy to see that π_1 statistically implements (resp., UC-implements) the original protocol π. The simulator only has to produce the additional message "`init`"; he can do so because the functionality CT_m informs him when it is invoked.

Second, each P_i is modified to wait for CT_m-output as soon as P_i itself has sent "`init`" to CT_m and P_{3-i} has also signaled to do so. All messages from P_{3-i} are buffered and processed by P_i only when that CT_m-output arrives. This protocol π_2 implements (resp., UC-implements) π_1 (and by transitivity also π) since the modified behavior of the π_2-parties can be simulated by a simulator in π_1 simply by delaying message delivery in π_1.

Now comes the interesting part: We modify each P_i so as to postpone the "`init`" message to CT_m to the end of the protocol run. Instead, P_i carries on with π_2 as if it *had* sent "`init`." When it goes into the waiting state (for CT_m-output ω, which will now certainly not arrive), it immediately leaves that waiting state. Then, P_i makes 2^m copies of its current internal state and carries on with 2^m parallel executions of π_2. In execution number j ($0 \leq j < 2^m$), P_i behaves as if it had gotten a seed $\omega = j$ from CT_m. At the end of the protocol run, when all the parallel executions have fixed their output, P_i then queries CT_m with an "`init`" message and waits for a seed ω to arrive. Finally, P_i outputs whatever the ωth execution of the parallelized protocol would have output.[12] Call the protocol with these modified parties π_3.

This protocol obviously fulfills the requirements in the lemma statement, and it only remains to show that π_3 implements (resp. UC-implements) π_2 (and thus π) and hence is a stand-alone secure (resp., universally composable) protocol for coin toss extension. We sketch a simulator $\mathcal{S}$ that simulates attacks (performed by a an adversary $\mathcal{A}$) on π_3 in the setting of π_2. Recall that π_3 and π_2 proceed identically until CT_m is queried (which causes $\mathcal{S}$ to be notified by CT_m). Hence, $\mathcal{S}$ can proceed like $\mathcal{A}$ until then.

[12] Note that it is crucial here that the machines do not have any secret internal state, since otherwise some protocol instances might reveal secrets that make the other instances insecure. This fact is used in the construction of the simulator below.

Once a party queries CT_m in π_3, however, that party internally forks into 2^m parallel executions, one for each possible CT_m (as described above). In interacting with π_2, $\mathcal{S}$ will only see one of those protocol executions, namely the one for the actual CT_m-output. Hence, in order to simulate π_3, $\mathcal{S}$ will have to simulate an additional $2^m - 1$ instances of (the remaining part of) π_2. However, $\mathcal{S}$ can easily start and maintain such simulations, since the state of the corresponding parties (which only consists of the exchanged messages and the hypothetical CT_m-output for that instance) is known. □

4.2. *Stand-alone Simulatability*

4.3. *Negative Results*

Theorem 10. *Let $m < n$ be functions in the security parameter k. Assume that m is not superlogarithmic. There is no (efficient or inefficient) non-trivial (in the sense of Definition 1) two-party $(m \to n)$-bit coin toss extension protocol with the following property:*

- *For any (possibly unbounded) adversary corrupting one of the parties, there is a negligible function μ such that for every security parameter k and every $c \in \{0,1\}^n$, the probability for protocol output c is at most $2^{-n} + \mu(k)$.*

If we require perfect non-triviality and perfect security (the probability for a given output c is at most 2^{-n}), no such a protocol π exists for any m (even superlogarithmic m).

Note that the notion of security used in this theorem is intentionally very weak. For example, if the first bit of the outcome is 0, and all other bits are uniformly random (and n is superlogarithmic), this notion of security is satisfied. Since the theorem is an impossibility result, using a weaker security notion strengthens the theorem. In Corollary 11, we will instead use the familiar simulation-based security notions.

We start with a proof sketch for the first statement (for the non-perfect case with non-superlogarithmic m).

Using Lemma 9 (and the fact that an n-bit coin toss immediately implies an m-bit coin toss for any $m < n$), we may assume that

- the available m-bit coin toss is only used at the end of the protocol,
- in the honest case, the parties never output distinct or invalid values, and
- $n = m + 1$.

To show the theorem, we first consider "complete transcripts" of the protocol. By a complete transcript, we mean all messages sent during the run of a protocol, but excluding the value of the m-bit coin toss. We define three sets of complete transcripts:

- the set $\mathfrak{A}$ of transcripts having nonzero probability for the protocol output 0^n,
- the set $\mathfrak{B}$ of transcripts having zero probability of output 0^n and zero probability that the protocol gives no output,
- and the set $\mathfrak{C}$ of transcripts having nonzero probability of giving no output.

Since for a complete transcript, the protocol output only depends on the m-bit coin toss, any of the nonzero probabilities in the definitions of $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ is at least 2^{-m}. Besides, by definition, $\mathfrak{B} = \{0,1\}^n \setminus (\mathfrak{A} \cup \mathfrak{C})$, but $\mathfrak{A}$ and $\mathfrak{C}$ need not be disjoint.

For any partial transcript p (i.e., a situation *during* the run of the protocol), we define three values α, β, and γ. The value α denotes the probability with which a corrupted Alice can enforce a transcript in $\mathfrak{A}$ starting from p, the value β denotes the probability with which a corrupted Bob can enforce a transcript in $\mathfrak{B}$, and the value γ denotes the probability that the complete protocol transcript will lie in $\mathfrak{C}$ if no one is corrupted. We show inductively that for any partial transcript p, we have $(1-\alpha)(1-\beta) \leq \gamma$. In particular, this holds for the beginning of the protocol. For simplicity, we assume that 2^{-m} is not only non-negligible, but noticeable (in the full proof, the general case is considered). Since a transcript in $\mathfrak{C}$ gives no output with probability at least 2^{-m}, the probability that the protocol generates no output (in the uncorrupted case) is at least $2^{-m}\gamma$. By the non-triviality condition, this probability is negligible, so γ must be negligible, too. So $(1-\alpha)(1-\beta)$ is negligible, too. Therefore, $\min\{1-\alpha, 1-\beta\}$ must be negligible. For now, we assume that $1-\alpha$ is negligible or $1-\beta$ is negligible (for the general case, see the full proof).

If $1-\alpha$ is negligible, α is overwhelming. The probability for output 0^n is at least $2^{-m}\alpha$. Since α is overwhelming and 2^{-m} noticeable, this is greater than $2^{-n} = \frac{1}{2}2^{-m}$ by a noticeable amount which contradicts the security property.

If $1-\beta$ is negligible, we have that Bob can ensure an output in $\{0,1\}^n \setminus \{0^n\}$ with overwhelming probability β. By the security property, however, such an output should occur at most with probability $(2^n-1)2^{-n}$ plus a negligible amount. $(2^n-1)2^{-n} = 1-2^{-n} = 1-2^{-m}/2$ is not overwhelming since m is not superlogarithmic by assumption, so we have a contradiction.

The perfect case is proven similarly.

We proceed with the full proof.

Proof (of Theorem 10). We first consider the statistical (i.e., non-perfect) case. Let us assume that such a π exists. Invoking Lemma 9, we can assume the following:

(i) If no party is corrupted, both parties always give the same (or no) output, and this output is a deterministic function of the sent messages, and the value of the used m-bit coin toss.
(ii) No messages are sent after invoking the m-bit coin toss.
(iii) The honest parties maintain no internal state except for the list of the messages sent so far.

Finally, we assume without loss of generality that $n = m+1$.

We call the parties Alice and Bob.

In the following, by a complete transcript t, we mean (the sequence of) all messages sent during a run of the protocol π, excluding the value s of the m-bit coin toss. The protocol outcome (of the honest parties) is then $f(t,s) \in \{0,1\}^n \cup \{\perp\}$ for some deterministic function f. By a partial transcript p, we mean a prefix of a complete transcript. We write $p \leq p'$ to denote that partial transcript p is a prefix of partial transcript p', and we write $p <_1 p'$ to denote that p is the immediate prefix of p' (i.e., the maximal $p \leq p'$ with of $p \neq p'$). Finally, let $\texttt{last}(p)$ denote the last message for a non-empty partial transcript p.

We can now distinguish three sets of complete transcripts t:

$$\mathfrak{A} := \left\{t : \exists s \in \{0,1\}^m : f(t,s) = 0^n\right\}$$
$$\mathfrak{B} := \left\{t : \forall s \in \{0,1\}^m : f(t,s) \neq 0^n, f(t,s) \neq \bot\right\}$$
$$\mathfrak{C} := \left\{t : \exists s \in \{0,1\}^m : f(t,s) = \bot\right\}$$

We now associate with each of the partial transcript p values α_p, β_p, and γ_p. The value α_p is defined as the maximum probability, going over all adversaries, that with corrupted Alice, the complete transcript of the protocol will lie in $\mathfrak{A}$, when starting with the partial transcript p (this is well defined, since the honest parties do not maintain a state except for the transcript so far). In other words, α_p denotes the probability that a corrupted Alice can enforce a complete transcript in $\mathfrak{A}$. Similarly, β_p is defined as the maximum probability that the complete transcript will lie in $\mathfrak{B}$ for corrupted Bob. And finally, γ_p is the probability that in the uncorrupted case, the complete transcript will lie in $\mathfrak{C}$ when starting from p. □

Claim 1. $(1-\alpha_p)(1-\beta_p) \leq \gamma_p$ for every partial transcript p.

Proof of Claim 1. Let first t be a complete transcript. Then, $\alpha_t, \beta_t, \gamma_t \in \{0,1\}$. Furthermore, since $\mathfrak{A} \cup \mathfrak{B} \cup \mathfrak{C}$ contains all complete transcripts, at least one of $\alpha_t, \beta_t, \gamma_t$ is not 0. So, for every complete transcript t, it holds $(1-\alpha_t)(1-\beta_t) \leq \gamma_t$.

Now consider a partial transcript p that is not complete. Let us assume that at that point of the protocol, it is Alice's turn to send a message. Consider the set

$$M_p := \{i \mid p <_1 i\}$$

of partial transcripts that can immediately succeed p. For each $i \in M_p$, there is a well-defined probability r_i that, given an uncorrupted Alice and a previous partial transcript p, Alice indeed sends $\texttt{last}(i)$. It is $\sum_{i \in M_p} r_i = 1$. Then, we have

$$\alpha_p = \max_{i \in M_p} \alpha_i, \qquad \beta_p = \sum_{i \in M_p} r_i \beta_i \qquad \gamma_p = \sum_{i \in M_p} r_i \gamma_i, \tag{2}$$

since a corrupted Alice may choose the partial transcript i that maximizes α, while if only Bob or no one is corrupted, the next partial transcript is chosen according to the probabilities r_i prescribed by the protocol. Let us assume that $(1-\alpha_i)(1-\beta_i) \leq \gamma_i$ holds for all $i \in M_p$. Then, we can conclude $(1-\alpha_p)(1-\beta_p) \leq \gamma_p$ as follows: First we write $\bar{\alpha}_p$ for $1-\alpha_p$ and analogously for the other values. Note that since $\sum_{i \in M_p} r_i = 1$, (2) also holds for $\bar{\beta}_{\ldots}$ and $\bar{\gamma}_{\ldots}$ instead of $\beta_{\ldots}$ and $\gamma_{\ldots}$). Hence,

$$\bar{\alpha}_p \bar{\beta}_p \overset{(i)}{=} \sum_i r_i \bar{\alpha}_p \bar{\beta}_i \overset{(ii)}{\leq} \sum_i r_i \bar{\alpha}_i \bar{\beta}_i \overset{(iii)}{\leq} \sum_i r_i \gamma_i \overset{(iv)}{=} \gamma_p, \tag{3}$$

where (i), (ii), (iv) follow from the middle, left, and right part of (2), respectively, and (iii) follows from our assumption.

Analogous reasoning can be applied when it is Bob's turn to send a message.

By induction, we therefore get $(1-\alpha_p)(1-\beta_p) \leq \gamma_p$ for any partial transcript p. This concludes the proof of Claim 1.

Now let $\emptyset$ denote the empty partial transcript, i.e., the beginning of the protocol. Then, for $\alpha := \alpha_\emptyset$, $\beta := \beta_\emptyset$, $\gamma := \gamma_\emptyset$, Claim 1 implies $(1-\alpha)(1-\beta) \leq \gamma$. We will construct a contradiction to the non-triviality and security properties of the protocol, which will finish the proof. □

Claim 2. $1-\alpha$ or $1-\beta$ is negligible on an infinite subset K' of security parameters.

Proof of Claim 2. If a protocol reaches a complete transcript in $\mathfrak{C}$, it will output $\perp$ with probability at least 2^{-m}, so the probability that π outputs $\perp$ is at least $2^{-m}\gamma$. On the other hand, since π is non-trivial, the probability that the protocol gives output $\perp$ in the uncorrupted case is negligible. Hence, $2^{-m}\gamma$ is negligible. Since 2^{-m} is non-negligible by assumption, there exists an infinite set K of security parameters k such that 2^{-m} is noticeable on K. If γ was non-negligible on K, $2^{-m}\gamma$ would be non-negligible on K. So γ must be negligible on K. Since $(1-\alpha)(1-\beta) \geq \gamma$ for each $k \in K$, one of $1-\alpha$ and $1-\beta$ is bounded by $\sqrt{\gamma}$ which is negligible on K. So there is an infinite set $K' \subseteq K$, such that $1-\alpha$ is negligible on K' *or* $1-\beta$ is negligible on K'. This shows Claim 2.

We can now finish the proof by showing a contradiction to the security of the protocol in either case of Claim 2. Let us consider the first case, i.e., α is overwhelming on K'. By assumption, the probability P for protocol output 0^n (with corrupted Alice) is bounded from above by $2^{-n}+\mu$ for negligible μ. But since a complete transcript in $\mathfrak{A}$ has probability at least 2^{-m} of giving output 0^n, we have $P \geq 2^{-m}\alpha = 2^{-n} + (\alpha - \frac{1}{2})2^{-m}$ (note $n = m+1$), so $\mu \geq (\alpha - \frac{1}{2})2^{-m}$. Since α is overwhelming and 2^{-m} noticeable on K', μ is not negligible, which concludes the proof in this case.

Let us consider the second case, i.e., β is overwhelming on K'. By assumption, the maximum probability P for an output in $\{0,1\}^n \setminus \{0^n\}$ (with corrupted Bob) is at most $2^{-n}(2^n-1) + \mu(2^n-1)$ for some negligible μ. On the other hand, since a complete transcript in $\mathfrak{B}$ has probability 1 of giving output in $\{0,1\}^n \setminus \{0^n\}$, we have $P \geq \beta$. It is

$$P \geq \beta = 2^{-n}(2^n-1) + \left(\tfrac{1}{2^n(2^n-1)} - \tfrac{1-\beta}{2^n-1}\right)(2^n-1)$$

and thus $\mu \geq \frac{1}{2^n(2^n-1)} - \frac{1-\beta}{2^n-1}$. Since 2^{-m} is noticeable on K', $2^n = 2 \cdot 2^m$ is polynomially bounded on K', so $\frac{1}{2^n(2^n-1)}$ is noticeable on K'. Further, $1-\beta$ is negligible, so the lower bound for μ is also noticeable on K'. It follows that μ is not negligible, which concludes the proof in the statistical (non-perfect) case.

For the perfect case, the proof of $(1-\alpha)(1-\beta) \leq \gamma$ is performed identically (since we did not use the non-triviality and the security of π in that part of the proof). By the perfect non-triviality, we get $\gamma = 0$, so for every k, at least one of α, β is 1. If $\alpha = 1$, the probability for an output of 0^n is (for suitable adversary) $\geq 2^{-m} > 2^{-n}$. If $\beta = 1$, the probability for an output in $\{0,1\}^n \setminus \{0^n\}$ is $1 > (2^n-1)2^{-n}$. Both cases contradict the security property. □

Corollary 11. *Let m be* not superlogarithmic *and $n > m$. Then, there is no non-trivial (in the sense of Definition 1) protocol realizing n-bit coin toss using an m-bit coin toss in the sense of* statistical *stand-alone simulatability.*

Let m be any *function (possibly superlogarithmic) and $n > m$. Then, there is no perfectly non-trivial protocol realizing n-bit coin toss using an m-bit coin toss in the sense of* perfect *stand-alone simulatability.*

Proof. A statistically secure protocol would have the security property from Theorem 10 and thus, if non-trivial, contradict Theorem 10 analogously for perfect security. □

4.4. *Positive Results*

Now we will prove that there exists a protocol for coin toss extension from m to n bits that is statistically stand-alone simulatably secure. The basic idea is to have the parties P_1 and P_2 contribute random strings to generate one string with sufficiently large min-entropy (the min-entropy of a random variable X is defined as $\min_x -\log \mathsf{Pr}[X = x]$). The randomness from this string is then extracted using a randomness extractor. The amount of perfect randomness (i.e., the size of the m-bit coin toss) one needs to invest is smaller than the amount extracted. This makes coin toss extension possible.

For our protocol, we need a family of strong randomness extractors with suitable parameters. The following lemma states the existence of these extractors.

Lemma 12. *For every m, there exists a function $h_m : \{0,1\}^m \times \{0,1\}^{m-1} \to \{0,1\}$, $(s,x) \mapsto r$ such that for a uniformly distributed s and for an x with min-entropy of at least t, the statistical distance between $s \| h_m(s,x)$ and the uniform distribution on $\{0,1\}^{m+1}$ is at most $2^{-t/2}/\sqrt{2}$. The functions h_m are efficiently computable.*

Proof. Let $h_m(s,x) := \langle s_1 \dots s_{m-1}, x\rangle \oplus s_m$. Here, $\langle \cdot,\cdot\rangle$ denotes the inner product and $\oplus$ the addition over GF(2). It is easy to verify that $h_m(s,\cdot)$ constitutes a family of universal hash functions [10], where s is the index selecting from that family. Therefore, the Leftover Hash Lemma [20,26] guarantees that the statistical distance between $s \| h_m(s,x)$ and the uniform distribution on $\{0,1\}^{m+1}$ is bounded by $\frac{1}{2}\sqrt{2 \cdot 2^{-t}} = 2^{-t/2}/\sqrt{2}$. □

With this family of functions h_m, a simple protocol is possible that extends $m(k)$ coin tosses to $m(k)+1$ if the function $m(k)$ is superlogarithmic.

Theorem 13. *Let $m(k)$ be a superlogarithmic function. Then, there exists a constant round statistically stand-alone simulatable protocol with efficient simulator that realizes an $(m+1)$-bit coin toss using an m-bit coin toss.*

Proof. Let h_m be as in Lemma 12. Then, the following protocol realizes a coin toss extension by one bit.

1. P_1 uniformly chooses $a \in \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor}$ and sends a to P_2.
2. P_2 uniformly chooses $b \in \{0,1\}^{\lceil \frac{m-1}{2} \rceil}$ and sends b to P_1.

3. If one party fails to send a string of appropriate length or aborts, then this string is assumed by the other party to be an all-zero string of the appropriate length.
4. P_1 and P_2 invoke the m-bit coin toss functionality and obtain a uniformly distributed $s \in \{0,1\}^m$. If one party P_i fails to invoke the coin toss functionality or aborts, then the other party chooses s at random.
5. Both P_1 and P_2 compute $s \| h_m(s, a\|b)$ and output this string.

Note that the protocol is constructed in a way that the adversary is not able to abort the protocol. Hence, we can safely assume that the adversary will send some message of the correct length and will invoke the coin toss functionality. (We follow [14, Construction 7.4.7] in this.) We assume the adversary to corrupt P_2; corruption of P_1 is handled analogously. Further, we assume without loss of generality that the random tape t of $\mathcal{A}$ to be fixed. (The advantage of a probabilistic adversary is bounded by that of the deterministic adversary with a worst-case random tape.) Due to these assumptions, there exists a function $f_{\mathcal{A}} : \{0,1\}^{\lfloor m/2 \rfloor} \to \{0,1\}^{\lceil m/2 \rceil}$ for each real adversary $\mathcal{A}$ such that the message b sent in step 2 of the protocol equals $f_{\mathcal{A}}(a)$. (Since t is fixed, it does not have to be included as an argument to $f_{\mathcal{A}}$.) There is no loss in generality if we assume the view of the adversary to consist of just a, b, s since the complete view can be reconstructed given these values and the (fixed) random tape t.

Now for a specific adversary $\mathcal{A}$ with fixed random tape corrupting P_2, the output distribution of the real protocol (i.e., view and output) is completely described by the following game: Choose $a \overset{R}{\in} \{0,1\}^{\lfloor m/2 \rfloor}$, let $b \leftarrow f_{\mathcal{A}}(a)$, choose $s \overset{R}{\in} \{0,1\}^{m(k)}$, let $r \leftarrow s \| h_m(s, a\|b)$, and return $((a, b, s), r)$.

We now describe the simulator. To distinguish the random variables in the ideal model from their real counterparts, we decorate them with a $\sim$, e.g., $\tilde{a}, \tilde{b}, \tilde{s}$. The simulator in the ideal model obtains a string $\tilde{r} \overset{R}{\in} \{0,1\}^{m+1}$ from the ideal n-bit coin toss functionality and sets $\tilde{s} = r_1 \dots r_m$. Then, the simulator chooses $\tilde{a} \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor}$ and computes $\tilde{b} = f_{\mathcal{A}}(\tilde{a})$ by giving $\tilde{a}$ to a simulated copy of the real adversary. If $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) = \tilde{r}_{m+1}$, the simulator gives $\tilde{s}$ to the simulated real adversary expecting the coin toss. Then, the simulator outputs the view $(\tilde{a}, \tilde{b}, \tilde{s})$. If, however, $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) \neq \tilde{r}_{m+1}$, then the simulator *rewinds* the adversary, i.e., the simulator chooses a fresh $\tilde{a} \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor}$ and again computes $\tilde{b} = f_{\mathcal{A}}(a)$. If now $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) = \tilde{r}_{m+1}$, the simulator outputs $(\tilde{a}, \tilde{b}, \tilde{s})$. If again $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) \neq \tilde{r}_{m+1}$, then the simulator rewinds the adversary again. If after k invocations of the adversary no triple $(\tilde{a}, \tilde{b}, \tilde{s})$ was output, the simulator aborts and outputs *fail*.

To show that the simulator is correct, we have to show that the following two distributions are statistically indistinguishable: $((a, b, s), r)$ as defined in the real model, and $((\tilde{a}, \tilde{b}, \tilde{s}), \tilde{r})$.

By construction of the simulator, it is obvious that the two distributions are identical under the condition that $r_m = 0, \tilde{r}_m = 0$ and that the simulator does not fail. The same holds given $r_m = 1, \tilde{r}_m = 1$ and that the simulator does not fail. Therefore, it is sufficient to show two things: (i) The statistical distance between r and the uniform distribution on n bits is negligible, and (ii) the probability that the simulator fails is negligible. Property (i) is shown using the properties of the randomness extractor h_m. Since a is chosen at random, the min-entropy of a is at least $\lfloor \frac{m-1}{2} \rfloor \geq \frac{m}{2} - 1$, so the min-entropy of $a\|b$ is also

at least $\frac{m}{2} - 1$. Since s is uniformly distributed, it follows by Lemma 12 that the statistical distance between $r = s \| h_m(s, a \| b)$ and $\tilde{r}$ is bounded by $2^{-m/4-1/2}/\sqrt{2} = (2^{-m})^{1/4}/2$. Since for superlogarithmic m, we have that 2^{-m} is negligible, this statistical distance is negligible.

Property (ii) is then easily shown: From (i), we see that after each invocation of the adversary, the distribution of $h_m(\tilde{s}, \tilde{a} \| \tilde{b})$ is negligibly far from uniform. So the probability that $h_m(\tilde{s}, \tilde{a} \| \tilde{b}) \neq \tilde{r}_m$ is at most negligibly higher than $\frac{1}{2}$. Since the $h_m(\tilde{s}, \tilde{a} \| \tilde{b})$ in the different invocations of the adversary are independent, the probability that $h_m(\tilde{s}, \tilde{a} \| \tilde{b}) \neq \tilde{r}_m$ for all m is negligibly far from 2^{-k}. So the simulator fails only with negligible probability.

It follows that the real and the ideal protocol executions are indistinguishable, and the protocol stand-alone simulatably implements an $(m+1)$-bit coin toss. □

The idea of the one bit extension protocol can be extended by using an extractor that extracts a larger amount of randomness. This yields constant round coin toss extension protocols. However, the simulator needed for such a protocol does not seem to be efficient, even if the real adversary is. To get a protocol that also fulfills the property of both computational stand-alone simulatability and statistical stand-alone simulatability, we need a simulator that is efficient if the adversary is.

Below, we give such a coin toss extension protocol for superlogarithmic $m(k)$. This protocol is statistically *and* computationally secure, i.e., the simulator for polynomial-time adversaries is polynomially bounded, too. The basic idea here is to extract one bit at a time in polynomially many rounds.

Theorem 14. *Let $m(k)$ be superlogarithmic, and $p(k)$ be a positive polynomially bounded function, then there exists a statistically and computationally stand-alone simulatable protocol with efficient simulator that realizes an $(m + p)$-bit coin toss using an m-bit coin toss.*

Proof. Let h_m be as in Lemma 12. Then, the following protocol realizes a coin toss extension by $p(k)$ bits.

1. `for` $i = 1$ `to` $p(k)$ `do`
   (a) P_1 uniformly chooses $a_i \in \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor}$ and sends a_i to P_2.
   (b) P_2 uniformly chooses $b_i \in \{0,1\}^{\lceil \frac{m-1}{2} \rceil}$ and sends b_i to P_1.
   (c) If one party fails to send a string of appropriate length or aborts, then this string is assumed by the other party to be an all-zero string of the appropriate length.
2. P_1 and P_2 invoke the m-bit coin toss functionality and obtain a uniformly distributed $s \in \{0,1\}^m$. If one party P_i fails to invoke the coin toss functionality or aborts, then the other party chooses s at random.
3. P_1 and P_2 compute $s \| h_m(s, a_1 \| b_1) \| \ldots \| h_m(s, a_{p(k)} \| b_{p(k)})$ and output this string.

We describe the proof for the case of a corrupted P_2; for corrupted P_1, the proof is analogous. Without loss of generality, we can assume that the adversary computes the values b_i by evaluating $b_i := f_i(a_1, \ldots, a_i)$ for some deterministic function f_i. We can also assume that the view of the adversary consists only of $(a_1, b_1, \ldots, a_p, b_p, s)$

(as we did in proof of Theorem 13). The output of the honest party P_1 is $r := s\|h_m(s, a_1\|b_1)\|\ldots\|h_m(s, a_p\|b_p)$.

To show security, we have to construct a polynomial-time simulator that, after obtaining a random $\tilde{r} \in \{0,1\}^{m+p}$ from the ideal n-bit coin toss functionality, outputs $(\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_p, \tilde{b}_p, \tilde{s})$ such that $(a_1, b_1, \ldots, a_p, b_p, s, r)$ and $(\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_p, \tilde{b}_p, \tilde{s}, \tilde{r})$ are statistically indistinguishable.

To construct the simulator, we first construct auxiliary algorithms S_i: Given a seed $\tilde{s}$, values $a_1, \ldots, a_{i-1}$, and a bit $\tilde{r}_i$, $S_i(\tilde{s}, a_1, \ldots, a_{i-1}, \tilde{r}_i)$ picks a random $\tilde{a}_i \in \{0,1\}^{\lceil \frac{m-1}{2} \rceil}$, sets $\tilde{b}_i := f_i(a_1, \ldots, a_{i-1}, \tilde{a}_i)$, and checks whether $h_m(\tilde{s}, \tilde{a}_i\|\tilde{b}_i) = \tilde{r}_i$. If so, S_i returns $\tilde{a}_i, \tilde{b}_i$. Otherwise, S_i tries again (picking a new $\tilde{a}_i$). S_i performs up to k tries. (k is the security parameter.)

We claim that the outputs of the following two games have, for all $a_i, \ldots, a_{i-1}$, statistical distance at most μ for some negligible function μ:

$$\begin{aligned} &\tilde{s} \overset{R}{\in} \{0,1\}^m,\ a_i \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_i := f_i(a_1, \ldots, a_i),\ r_{m+i} := h_m(\tilde{s}, a_i\|b_i),\\ &\quad \text{output } (a_i, b_i, \tilde{s}, r_{m+i})\\ \text{and } &\tilde{s} \overset{R}{\in} \{0,1\}^m,\ \tilde{r}_{m+i} \overset{R}{\in} \{0,1\},\ \tilde{a}_i, \tilde{b}_i := S_i(\tilde{s}, a_1, \ldots, a_{i-1}, \tilde{r}_i),\\ &\quad \text{output } (\tilde{a}_i, \tilde{b}_i, \tilde{s}, \tilde{r}_{m+i}) \end{aligned} \tag{4}$$

As this claim is shown exactly as the indistinguishability of $((a, b, s), r)$ and $((\tilde{a}, \tilde{b}, \tilde{s}), \tilde{r})$ in the proof of Theorem 13, we omit the proof of the claim. Note that μ can be chosen to be independent of i.

In the real model, we have that the distribution of $(a_1, b_1, \ldots, a_p, b_p, s, r)$ can be described by the following game G_R:

$$\begin{aligned} &s \overset{R}{\in} \{0,1\}^m,\ a_1 \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_1 := f_1(a_1),\ r_{m+1} := h_m(\tilde{s}, a_1\|b_1),\ \ldots,\\ &a_p \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_p := f_p(a_1, \ldots, a_p),\ r_{m+p} := h_m(\tilde{s}, a_p\|b_p),\\ &\quad r := s\|r_{m+1}\|\ldots\|r_{m+p},\ \text{output } (a_1, b_1, \ldots, a_p, b_p, s, r). \end{aligned}$$

In the ideal model, the distribution of $(\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_p, \tilde{b}_p, \tilde{s}, \tilde{r})$ can be described by the following game G_I:

$$\begin{aligned} &\tilde{s} \overset{R}{\in} \{0,1\}^m,\ \tilde{r}_{m+1} \overset{R}{\in} \{0,1\},\ \tilde{a}_1, \tilde{b}_1 := S_1(\tilde{s}, \tilde{r}),\ \ldots,\\ &\tilde{r}_{m+p} \overset{R}{\in} \{0,1\},\ \tilde{a}_p, \tilde{b}_p := S_p(\tilde{s}, \tilde{a}_1, \ldots, \tilde{a}_{p-1}, \tilde{r}),\ \tilde{r} := \tilde{s}\|\tilde{r}_{m+1}\|\ldots\|\tilde{r}_{m+p},\\ &\text{output } (\tilde{a}_1, \tilde{b}_1, \ldots, \tilde{a}_p, \tilde{b}_p, \tilde{s}, \tilde{r}). \end{aligned}$$

To show that the outputs of those two games are indistinguishable, we first introduce a hybrid game H_i:

$$
\begin{aligned}
&s \overset{R}{\in} \{0,1\}^m,\\
&a_1 \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_1 := f_i(a_1),\ r_{m+1} := h_m(\tilde{s}, a_1 \| b_1),\ \ldots,\\
&a_i \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_i := f_i(a_1,\ldots,a_i),\ r_{m+i} := h_m(\tilde{s}, a_i \| b_i),\\
&r_{m+i+1} \overset{R}{\in} \{0,1\},\ a_{i+1}, b_{i+1} := S_{i+1}(s, r, a_1, \ldots, a_i),\ \ldots,\\
&r_{m+p} \overset{R}{\in} \{0,1\},\ a_p, b_p := S_p(s, a_1, \ldots, a_{p-1}, r),\\
&r := s \| r_{m+1} \| \ldots \| r_{m+p}, \text{output } (a_1, b_1, \ldots, a_p, b_p, s, r).
\end{aligned}
$$

For $i = 0$, this is the game G_I, and for $i = p$, this is the game G_R.

Note that we can reorder the computations in H_{i-1} as follows:

$$
\begin{aligned}
&a_1 \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_1 := f_i(a_1),\ \ldots,\\
&a_{i-1} \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_{i-1} := f_{i-1}(a_1,\ldots,a_{i-1}),\\
&s \overset{R}{\in} \{0,1\}^m,\ r_{m+i} \overset{R}{\in} \{0,1\},\ a_i, b_i := S_i(s, r, a_1, \ldots, a_{i-1}),\\
&r_{m+1} := h_m(\tilde{s}, a_1 \| b_1),\ \ldots,\ r_{m+i-1} := h_m(\tilde{s}, a_{i-1} \| b_{i-1}),\\
&r_{m+i+1} \overset{R}{\in} \{0,1\},\ a_{i+1}, b_{i+1} := S_{i+1}(s, r, a_1, \ldots, a_i),\ \ldots,\\
&\quad r_{m+p} \overset{R}{\in} \{0,1\},\ a_p, b_p :=:= S_p(s, r, a_1, \ldots, a_p)
\end{aligned}
$$

And we can reorder the computations in H_i as follows:

$$
\begin{aligned}
&a_1 \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_1 := f_i(a_1),\ \ldots,\\
&a_{i-1} \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_{i-1} := f_{i-1}(a_1,\ldots,a_{i-1}),\\
&s \overset{R}{\in} \{0,1\}^m,\ a_i \overset{R}{\in} \{0,1\}^{\lfloor \frac{m-1}{2} \rfloor},\ b_i := f_i(a_1,\ldots,a_i),\ r_{m+i} := h_m(s, a_i \| b_i)\\
&r_{m+1} := h_m(\tilde{s}, a_1 \| b_1),\ \ldots,\ r_{m+i-1} := h_m(\tilde{s}, a_{i-1} \| b_{i-1}),\\
&r_{m+i+1} \overset{R}{\in} \{0,1\},\ a_{i+1}, b_{i+1} := S_{i+1}(s, r, a_1, \ldots, a_i),\ \ldots,\\
&\quad r_{m+p} \overset{R}{\in} \{0,1\},\ a_p, b_p :=:= S_p(s, r, a_1, \ldots, a_p)
\end{aligned}
$$

When reordered like this, only the second line is different in H_{i-1} and H_i, and it corresponds to the second line and first line of (4), respectively. Thus, by (4), the statistical distance between the output of H_{i-1} and H_i is at most μ. Thus, the statistical distance between G_I and G_R, which is the same as the statistical distance between H_0 and H_p, is at most $p\mu$ which is negligible.

This shows security in the case of corrupted P_2. □

Remark. We note that this protocol defines a "default value" for values not sent by the other party. Hence, it guarantees output even in the face of uncooperative parties and

thus even achieves the stronger notion of a "fair" coin toss from [11,23] (see also our discussion in Sect. 1).

4.5. *Universal Composability (Statistical/Perfect Case)*

In contrast to the stand-alone case, in the UC setting, statistically secure coin toss extension protocols are impossible. Intuitively, the reason for this difference is that our positive result for stand-alone security (Theorem 14) rewinds an adversary in a simulation, while this is not possible for UC security.

More precisely, we show that there is no protocol that runs a polynomial number of rounds, uses an m-bit coin toss functionality as a seed, and statistically UC-implements the n-bit coin toss functionality for $n > m$.

The proof of this statement is done by contradiction. Invoking Lemma 9, we can assume that a protocol for statistically universally composable coin toss extension has a certain outer form. Then, we show that any such protocol (of this particular outer form) is insecure.

More concretely, our plan of action will be as follows. For contradiction, assume a statistically universally composable $(m \to n)$-coin toss extension protocol. We may assume that the m-bit seed coin toss is only invoked at the end of the extension protocol.

Also, slightly simplifying things, we can think of the produced n-bit coin toss as a deterministic function $f(c, s)$ of the protocol transcript c (i.e., the transcript of all messages exchanged between the parties) and the m-bit coin toss s. Now for every transcript c, the set $\{f(c, s) \mid s \in \{0, 1\}^m\}$ of possible (valid) protocol outputs after transcript c is at most half the size of $\{0, 1\}^n$. On the other hand, initially, almost all outputs of $\{0, 1\}^n$ should be roughly equally probable. Hence, a full transcript c "cuts away" about half of all possible protocol outputs.

By assumption, the transcript c is generated interactively from scratch, without using the m-bit coin toss s. Also, every party contributes only polynomially many messages to c. Hence, there is a single message that "cuts away" a non-negligible fraction of all possible outputs. Call such a message "critical." Our adversary $\mathcal{A}$ will corrupt one party passively and detect the first such critical message. When encountering such a message, $\mathcal{A}$ will then internally toss a coin. If heads comes out, $\mathcal{A}$ will continue the protocol run, and let the corrupted party send that critical message. If tails comes out, $\mathcal{A}$ will rewind the party and let it send a different message. We will show that this decision (whether to let the party send the critical message) has a non-negligible impact on the protocol's output distribution. More concretely, the probability that the protocol output lies in exactly that subset of possible outputs that would have been "cut away" by the critical message is highly correlated with the outcome of $\mathcal{A}$'s coin flip.

We will now proceed to formalize this proof outline. This will require some preparations.

For the following statements, we always assume that $m = m(k), n = n(k)$ are arbitrary functions, only satisfying $0 \leq m(k) < n(k)$ for all k. We also restrict to protocols that proceed in a polynomial number of rounds. That is, by a "protocol," we mean in the following one in which each party halts after at most $p(k)$ activations, where $p(k)$ is a polynomial that depends only on the protocol. (We do not, however, require the parties to be computationally limited.) We stress that a protocol in which the honest parties run

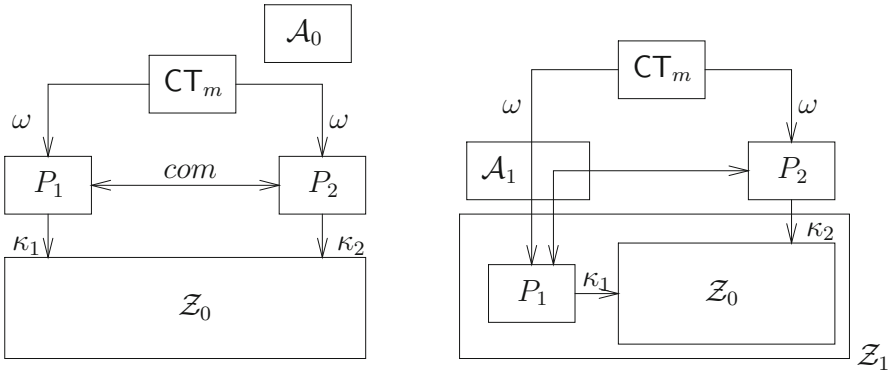

Fig. 5. *Left* Initial setting D_0 for the statistical case. (Some connections that are not important for our proof have been omitted.) *Right* Setting D_1 with a corrupted P_1. Setting D_2 (with P_2 corrupted instead of P_1) is defined analogously.

in polynomial time automatically has a polynomial number of rounds; the restriction to a polynomial number of rounds is thus a very weak one.

Theorem 15. *There is no non-trivial statistically or perfectly universally composable protocol for $(m \to n)$-coin toss extension that proceeds in a polynomial number of rounds.*

Proof. Assume for contradiction that π, using CT_m, is a statistically universally composable implementation of CT_n. By Lemma 9, we may also assume that π satisfies the requirements from that lemma.

Setting D_0 Fix an environment $\mathcal{Z}_0$ that gives both parties "`init`" input and then waits for both parties to output a coin toss outcome. Consider an adversary $\mathcal{A}_0$ that delivers all messages between the parties immediately. The resulting setting D_0 is depicted in Fig. 5.

Denote the protocol communication in a run of D_0, i.e., the ordered list of messages sent between P_1 and P_2, by com. Denote by κ_1 and κ_2 the final outputs of the parties. For $M \subseteq \{0,1\}^n$ and a possible protocol communication prefix $\overline{c}$, let $\mathsf{E}(M, \overline{c})$ be the probability that the protocol outputs are identical and in M, provided that the protocol communication starts with $\overline{c}$, i.e.,

$$\mathsf{E}(M, \overline{c}) := \mathsf{Pr}[\kappa_1 = \kappa_2 \in M \mid \overline{c} \leq \text{com}],$$

where $x \leq y$ means that x is a prefix of y.

Note that the parties have, apart from their communication com, only the seed $\omega \in \{0,1\}^m$ provided by CT_m for computing their final output κ. So we may assume that there is a deterministic function f for which $\kappa_1 = \kappa_2 = f(\text{com}, \omega)$ with overwhelming probability.

For a fixed protocol communication com $= c$, consider the set

$$M_c := \{0,1\}^n \setminus \{ f(c,s) \mid s \in \{0,1\}^m \}$$

of "improbable outputs" after communication c. Then, obviously $|M_c| \geq 2^n - 2^m \geq 2^{n-1}$. Since the protocol π is non-trivial, we have that $\Pr[\kappa_1, \kappa_2 \neq \perp]$ is overwhelming. Hence, $\left| \Pr[\kappa_1 = \kappa_2 \in M_c] - |M_c| 2^{-n} \right|$ is negligible (otherwise, one could distinguish the real and the ideal models). Thus, for sufficiently large security parameters k, the probability that $\kappa_1 = \kappa_2 \in M_c$ is at least $2/5$. (Here, any number strictly between 0 and $1/2$ would have done as well.) Since $\mathsf{E}(M_c, \emptyset)$ (for the empty transcript $\emptyset$) is, by definition, the probability that $\kappa_1 = \kappa_2 \in M_c$, we have $\mathsf{E}(M_c, \emptyset) \geq 2/5$ for sufficiently large k. Also, $\mathsf{E}(M_c, c)$ is negligible by definition of M_c, so M_c satisfies

$$\mathsf{E}(M_c, \emptyset) - \mathsf{E}(M_c, c) \geq \frac{1}{3} \tag{5}$$

for sufficiently large k.

Since the protocol consists by assumption only of polynomially many rounds, c is a list of size at most $p(k)$ for a fixed polynomial p. This means that there are a prefix $\overline{c}$ of c and a single message m (either sent from P_1 to P_2 or vice versa) such that $\overline{c}m \leq c$ and

$$\mathsf{E}(M_c, \overline{c}) - \mathsf{E}(M_c, \overline{c}m) \geq \frac{1}{3p(k)} \tag{6}$$

for sufficiently large k. Intuitively, this means that at a certain point during the protocol run, a single message m had a significant impact on the probability that the protocol output is in M_c.

A message m that satisfies (6) for $c :=$ com we call *critical*. (Remember that com is the random variable describing the communication in an execution of the real protocol.)

Setting D_j Note that in any execution, a critical m is sent by at least one party. So there is a $j \in \{1,2\}$ such that for infinitely many k, party P_j sends a critical m with probability at least $1/2$. We describe a modification D_j of setting D_0. In setting D_j, party P_j is corrupted and simulated (honestly) inside $\mathcal{Z}_j$. Furthermore, adversary $\mathcal{A}_j$ simply relays all communication between this simulation inside $\mathcal{Z}_j$ and the external machines P_{3-j} and CT_m. For supplying inputs to the simulation of P_j and to the uncorrupted P_{3-j}, a simulation of $\mathcal{Z}_0$ is employed inside $\mathcal{Z}_j$. The situation (for $j = 1$) is depicted in Fig. 5.

Since D_j is basically only a regrouping of D_0, the random variables com, ω, and κ_i are distributed exactly as in D_0, so we simply identify them with the corresponding random variables in D_0. In particular, in D_j, for infinitely many k, a critical message is sent by P_j.

Setting D'_j. Now we slightly change the environment $\mathcal{Z}_j$ into an environment $\mathcal{Z}'_j$. Each time the simulated P_j sends a message m to P_{3-j}, $\mathcal{Z}'_j$ checks whether

$$\exists M \subseteq \{0,1\}^n : \quad \mathsf{E}(M, \overline{c}) - \mathsf{E}(M, \overline{c}m) \geq \frac{1}{3p(k)}, \tag{7}$$

where $\overline{c}$ denotes the communication between P_j and P_{3-j} so far.

If (7) holds at some point for the first time, then $\mathcal{Z}'_j$ tosses a coin b uniformly at random and proceeds as follows: If $b = 0$, then $\mathcal{Z}'_j$ keeps going just as $\mathcal{Z}_j$ would have. In particular, $\mathcal{Z}'_j$ then lets P_j send m to P_{3-j}. However, if $b = 1$, then $\mathcal{Z}'_j$ rewinds the simulation of P_j to the point *before* that activation and activates P_j again with fresh randomness, thereby letting P_j send a possibly different message m'. In the further proof, $\bar{c}$, m, and M refer to these values for which (7) holds.

In any case, after having tossed the coin b once, $\mathcal{Z}'_j$ remembers the set M from (7) and does not check (7) again. After the protocol finishes, $\mathcal{Z}'_j$ outputs (b, β). Here, b is as above, and $\beta := 1$ iff $\kappa_1 = \kappa_2 \in M$ and $\beta := 0$ otherwise. ($b, \beta := \perp$ if (7) was never fulfilled.)

Now by our choice of j, and since a critical m fulfills (7), $\mathsf{Pr}[b \neq \perp] \geq 1/2$ for infinitely many k.

Also, Lemma 9 guarantees that the internal state of the parties at the time of tossing b consists only of $\bar{c}$. So, when $\mathcal{Z}'_j$ has chosen $b = 1$, and rewound the simulated P_j, the probability that at the end of the protocol $\kappa_1 = \kappa_2 \in M$ holds is the same as the probability of that event in the setting D_j under the condition that the communication , begins with $\bar{c}$. This probability again is exactly $\mathsf{E}(M, \bar{c})$ by definition.

Similarly, when $\mathcal{Z}'_j$ has chosen $b = 0$, the probability that at the end of the protocol $\kappa_1 = \kappa_2 \in M$ is the same as the probability of that event in the setting D_j under the condition that the communication , begins with $\bar{c}m$, i.e., $\mathsf{E}(M, \bar{c}m)$.

Therefore, just before $\mathcal{Z}'_j$ chooses b (i.e., when $\bar{c}$ and M are already determined), the probability that at the end we will have $\beta = 1 \wedge b = 1$ is $\frac{1}{2}\mathsf{E}(M, \bar{c})$ and the probability of $\beta = 1 \wedge b = 0$ is $\frac{1}{2}\mathsf{E}(M, \bar{c}m)$. Therefore, the difference between these probabilities is at least $\frac{1}{2}\big(\mathsf{E}(M, \bar{c}) - \mathsf{E}(M, \bar{c}m)\big) \geq \frac{1}{6p(k)}$.

Since $\mathsf{Pr}[b \neq \perp] \geq \frac{1}{2}$ for infinitely many k, it follows that

$$\mathsf{Pr}[\beta = 1 \wedge b = 1] - \mathsf{Pr}[\beta = 1 \wedge b = 0] \geq \frac{1}{12p(k)} \tag{8}$$

for infinitely many k when $\mathcal{Z}'_j$ runs with the real protocol as described above.

The contradiction We show that no simulator $\mathcal{S}_j$ can achieve property (8) in the ideal model, where $\mathcal{Z}'_j$ runs with CT_n and $\mathcal{S}_j$. To distinguish random variables during a run of $\mathcal{Z}'_j$ in the ideal model from those in the real model, we add a tilde to a random variable in a run of $\mathcal{Z}'_j$ in the ideal model, for example, $\tilde{b}$, $\tilde{\beta}$.

Since the protocol π is non-trivial, for any $\mathcal{S}_j$ achieving indistinguishability of real and ideal model, we can assume without loss of generality that $\mathcal{S}_j$ always delivers the outputs $\tilde{\kappa}_1 = \tilde{\kappa}_2 =: \tilde{\kappa}$.

Recall that $\tilde{b}$ is independently and uniformly chosen after $\tilde{M}$ is determined and that $\tilde{\kappa}$ is chosen independently by CT_n. Hence, the variable $\tilde{b}$ and the tuple $(\tilde{M}, \tilde{\kappa})$ are independent given $\tilde{b} \neq \perp$. Hence, since $\tilde{\beta}$ is a function of $\tilde{M}$ and $\tilde{\kappa}$,

$$\mathsf{Pr}\Big[(\tilde{b}, \tilde{\beta}) = (0, 1)\Big] = \mathsf{Pr}\Big[(\tilde{b}, \tilde{\beta}) = (1, 1)\Big]. \tag{9}$$

Combining (9) with (8), we get that $\mathcal{Z}'_j$'s output (i.e., (b, β), resp. $(\tilde{b}, \tilde{\beta})$) differs non-negligibly in real and ideal model. So no simulator $\mathcal{S}_j$ can simulate attacks carried out by $\mathcal{Z}'_j$ and $\mathcal{A}_j$, which gives the desired contradiction. □

Actually, in the case of perfect security, impossibility holds even for protocols with arbitrarily many rounds. Namely, in the proof of Theorem 15, we have used that the protocol has only polynomially many rounds only in one place. Namely, we obtained in (6) that one party sends a message that has non-negligible impact on the probability that $\kappa \in M$. For perfect security, we need only that one party has some *nonzero* impact on that probability, i.e., we can drop the requirement on the polynomial number of protocol rounds in the perfect case. The reasoning in the proof stays exactly the same only that we end up with the left-hand side of (8) being nonzero instead of non-negligible. This suffices to show that the considered protocol is not perfectly secure and thus:

Corollary 16. *There is no non-trivial perfectly universally composable protocol for $(m \to n)$-coin toss extension (the number of rounds does not matter here).*

However, we do not know whether or not there is a protocol for the statistical case that proceeds in a superpolynomial number of rounds.

Note that all discussions above assume that statistical security means security with respect to computationally unbounded adversaries, simulators, and environments, i.e., machines that can implement any probabilistic function, even, for example, the halting problem or similar. Often, however, statistical security is instead defined with respect to (computationally unbounded) Turing machines, i.e., machines that can only implement computable functions. To show the above results for this case, one could try and check whether all constructions given in the proof above are indeed computable or can be replaced by computable approximations. Fortunately, however, there is an easier way, using results from [27].

Corollary 17. *Say a protocol is bounded time if there is a (not necessary small or computable) bound on the execution time of that protocol (e.g., all efficient protocols are bounded time). Let further n, m be computable functions, and $m > n$.*

Then, there is no non-trivial bounded-time protocol for $(m \to n)$-coin toss extension that proceeds in a polynomial number of rounds and that is statistically universally composable with respect to adversaries / environments / simulators that are computationally unbounded Turing machines.

Proof. [27] shows that a bounded-time protocol universally composably implements a bounded-time functionality with respect to computationally unbounded adversaries / environments / simulators if and only if it universally composably implements that functionality with respect to computationally unbounded Turing adversaries / environments / simulators. Since the n-bit and m-bit coin toss functionalities are bounded time, too ($n(k)$ can be evaluated in finite time), a protocol contradicting this corollary would also contradict Theorem 15. □

Similar reasoning applies to the perfect case, and we omit the details here.

5. CRS Extension

Before we go through the results one by one, we summarize the results of this section in the following table. The only case where CRS extension differs from coin toss extension is highlighted in boldface.

Security type	Level		
	Computational	Statistical	Perfect
Stand-alone simulatability	Yes	**No**	No
Universal composability	Depends[a]	No	No

[a]CRS extension is impossible if the seed does not have superlogarithmic length. The possibility result depends on the complexity assumption we use

5.1. *The Computational Case*

As already mentioned in Introduction, [14, Proposition 7.4.8] and [14, Proposition 7.4.3] show the existence of an n-bit coin toss protocol π for any polynomially bounded, efficiently computable n. This makes $(m \to n)$-CRS extension trivial: One can ignore the m-bit seed and use the protocol π to produce an n-bit random string which is then used as the CRS.

In the setting of computational universal composability, the results from Sect. 3.1 carry over directly. To state these results, we first have to specify the ideal functionality CRS.

The following corollary shows that CRS extension is possible in the computational UC setting given sufficiently long seeds.

Corollary 18. *Let $n = n(k)$ and $m = m(k)$ be polynomially bounded and efficiently computable functions. Assume one of the following conditions holds:*

- *m is polynomially large and ETD exists, or*
- *m is superpolylogarithmic and exponentially hard ETD exists.*

Then, there is a polynomial-time computationally universally composable $(m \to n)$-CRS extension protocol π.

Proof. The proof of Theorem 7 actually shows that an n-bit coin toss can be realized from an m-bit *CRS*. Furthermore, from an n-bit coin toss, we can trivially realize an n-bit CRS. Thus, from an m-bit CRS, we can realize an n-bit CRS. □

The following corollary shows that extending coin toss is impossible in the computational UC setting for short seeds.

Corollary 19. *Let $n = n(k)$ and $m = m(k)$ be functions with $n(k) > m(k) \geq 0$ for all k and assume that m is not superlogarithmic (i.e., 2^{-m} is non-negligible). Then, there*

is no non-trivial polynomial-time computationally universally composable protocol for $(m \to n)$*-CRS extension.*

The proof is identical to that of Theorem 8. (Except, of course, that we have to replace the mentions of the functionality CT by CRS and that the environment $\mathcal{Z}$ sends `getcrs` instead of `init`.)

5.2. *The Statistical and the Perfect Case*

For superlogarithmic m and $n > m$, Theorem 13 states that an $(m \to n)$-coin toss extension is possible with respect to statistical stand-alone simulatability. This is not true, however, for CRS extension. We will show that CRS extension is impossible for any length m of the seed, both for statistical and perfect security, and both for stand-alone simulatability and UC.

Theorem 20. *Let* $0 \leq m < n$ *be polynomially bounded functions in the security parameter* k*. Then, there is no non-trivial (in the sense of Definition 1) two-party* n*-bit CRS protocol* π *(not even an inefficient one) that uses an* m*-bit CRS and has the following property:*

- *There is a negligible function* μ *in the security parameter such that for any (possibly unbounded) adversary corrupting one of the parties, for every security parameter* k*, and for every set* $M \subseteq \{0,1\}^n$*, we have that the probability that the output of the honest party lies in* M *is at most* $2^{-n}|M| + \mu$*.*[13]

We begin with a proof sketch. For contradiction, assume a protocol π with bounds μ and ν as in the statement of the theorem. Let S denote the value of the seed (the initial CRS), and let R denote the outcome of the protocol (the extended CRS).

First, we find that there is an index i such that the i-th bit R_i of R is not completely determined by S (up to some negligible error). If there was no such index, each bit of R would be determined by S, and hence R could only take $2^m \ll 2^n$ different values.

Furthermore, for a fixed value s of S, let α_s denote the maximum probability that a corrupted Alice can achieve $R_i = 0$. Similarly, β_s denotes the maximum probability that a corrupted Bob can achieve $R_i = 1$. For any *fixed* s, we are in the same situation as in a coin toss protocol that has to pick a random bit R_i without using any seed at all. In this case, either Alice can enforce outcome 0 or Bob can enforce outcome 1 (Theorem 10). Thus, for all s, $\alpha_s \approx 1$ or $\beta_s \approx 1$. Let $V_\alpha := \{s : \alpha_s \approx 1\}$, i.e., V_α is the set of all seeds for which Alice can enforce outcome 0. V_β is defined analogously.

Let Δ_α denote the probability that in an honest execution, $S \in V_\alpha$ and $R_i \neq 0$. Let Δ_β denote the probability that in an honest execution, $S \in V_\beta$ and $R_i \neq 1$. If both $\Delta_\alpha \approx 0$ and $\Delta_\beta \approx 0$, then the value of R_i would be determined by whether $S \in V_\alpha$ holds. But this contradicts the fact that R_i is not determined by S. Thus, $\Delta_\alpha \not\approx 0$ or $\Delta_\beta \not\approx 0$. Without loss of generality, assume $\Delta_\alpha \not\approx 0$, i.e., with noticeable probability, in

[13] We bound this probability from above only since we want to allow that an honest party gives no output with high probability.

an honest execution we have $R_i \neq 0$, but the seed S is such that a corrupted Alice could have enforced $R_i = 0$.

Thus, a corrupt Alice can increase the bias toward 0 by the noticeably amount Δ_α when compared to the honest case. But since in the honest case, the bias toward 0 is $\frac{1}{2}$, Alice can enforce $R_i = 0$ with probability $\frac{1}{2} + \Delta_\alpha$. This violates the security of the protocol π.

Thus, we have led our initial assumption to a contradiction; hence, Theorem 20 holds.

We now proceed with the full proof.

Proof. Assume for contradiction that a protocol π with negligible bounds μ and ν as in the statement of the theorem exists.

Without loss of generality, we may assume that honest parties always give output in $\{0,1\}^n$ or no output. We also assume that if both parties are honest (and all messages are delivered), with probability 1, both parties give the same output or both parties give no output. The latter can be achieved by adding two additional messages at the end of the protocol where the parties compare their outputs (and give no output in the case of disagreement).

For the remainder of the proof, we fix the security parameter k.

We first make a number of simple definitions. Denote by the random variable $S \in \{0,1\}^m$ the initial seed that is available to both protocol parties (the CRS). Let the random variable $R \in \{0,1\}^n \cup \{\perp\}$ denote the protocol outcome, i.e., the extended CRS, in an *honest* protocol execution. We write $R = \perp$ for the case that no output is given. Let R_i be the i-th bit of R, with $R_i := \perp$ if $R = \perp$.

Let $b_i(s) \in \{0,1\}$ with $b_i(s) := 1$ iff $\mathsf{Pr}[R_i = 1 \mid S = s] > \mathsf{Pr}[R_i = 0 \mid S = s]$. (Intuitively, $b_i(S)$ is the most probable value of R_i given S.) □

Claim 1. There exists an $i \in \{1, \ldots, n\}$ such that we have:

$$\mathsf{Pr}[R_i = b_i(S)] \leq 1 - \frac{1-2\mu}{2n}. \tag{10}$$

First, assume for contradiction that (10) does not hold for any i. Let $f(s) := b_1(s) \| \ldots \| b_n(s)$ for $s \in \{0,1\}^m$, i.e., $f(S)$ is the value of R resulting from predicting R bitwise.

Then, we have

$$\mathsf{Pr}[R = f(S)] = \mathsf{Pr}\big[\forall i : R_i = b_i(S)\big] \geq 1 - \sum_{i=1}^{n} \mathsf{Pr}\big[R_i \neq b_i(S)\big] \overset{(*)}{>} 1 - n\frac{1-2\mu}{2n} = \frac{1}{2} + \mu \tag{11}$$

where $(*)$ uses the fact that (10) is assumed not to hold. With $M := f(\{0,1\}^m)$, we get $\mathsf{Pr}[R \in M] \geq \mathsf{Pr}[R = f(S)] \overset{(11)}{>} \frac{1}{2} + \mu$.

But by the security of π, we have that $\Pr[R \in M] \leq 2^{-n}|M| + \mu \leq 2^{-n}2^m + \mu\frac{1}{2} + \mu$. Thus, we have a contradiction, and hence (10) holds and Claim 1 is shown.

Claim 2. For the value i from Claim 1, we have

$$\mathsf{Pr}[R_i = b(S)] \leq 1 - \frac{1-2\mu}{2n}, \tag{12}$$

for *any* predicate $b : \{0,1\}^m \to \{0,1\}$.

Since $\Pr[R_i = b_i(s)|S = s] \geq \Pr[R_i = 1 - b_i(s)|S = s]$ by construction of b_i, we have $\Pr[R_i = b_i(s)|S = s] \geq \Pr[R_i = b(s)|S = s]$ both in the case $b_i(s) \neq b(s)$ and trivially in the case $b_i(s) = b(s)$. Thus,

$$\begin{aligned}\mathsf{Pr}[R_i = b(S)] &= \sum_{s\in S} 2^{-m} \Pr[R_i = b(s)|S = s] \\ &\leq \sum_{s\in S} 2^{-m} \Pr[R_i = b_i(s)|S = s] = \Pr[R_i = b_i(S)] \overset{(10)}{\leq} 1 - \frac{1-2\mu}{2n}.\end{aligned}$$

This shows Claim 2.

Note the implications of Claim 2: Intuitively it states that (if μ is small) there is a bit of the protocol output that is (to a certain extent) undetermined at the start of the protocol, even when knowing the seed S.

In the following, let i be as in Claims 1 and 2.

Claim 3. It is

$$\nu + \sqrt{\nu} \geq \frac{1}{4n} - \frac{\mu}{2n} - 2\mu. \tag{13}$$

To see this, for any seed $s \in \{0,1\}^m$, we denote by α_s the maximal probability for an adversary A^* that corrupts the first party Alice to achieve $R_i^* = 0$. Analogously, by β_s, we denote the maximal probability for an adversary that corrupts the second party Bob to achieve $R_i^* = 1$. Here, $R^* \in \{0,1\}^n \cup \{\perp\}$ denotes the honest party's output and R_i^* denotes the i-th bit of R^*. These definitions are similar to the definitions α and β from the proof of Theorem 10, except that here we consider the probabilities given a specific value s of the seed. Further, let $\gamma_s := \mathsf{Pr}[R = \perp \mid S = s]$ (in the uncorrupted case). An analogous calculation to the one in the proof of Theorem 10 shows that for all $s \in \{0,1\}^m$,

$$(1 - \alpha_s)(1 - \beta_s) \leq \gamma_s. \tag{14}$$

Now we define

$$\begin{aligned} V_\alpha &:= \{s \in \{0,1\}^m \mid \alpha_s > \beta_s\} & V_\beta &:= \{s \in \{0,1\}^m \mid \alpha_s \leq \beta_s\} \\ \Gamma_\alpha &:= \mathsf{Pr}[R_i = 0 \text{ and } S \in V_\alpha] & \Gamma_\beta &:= \mathsf{Pr}\big[R_i = 1 \text{ and } S \in V_\beta\big] \\ \Delta_\alpha &:= 2^{-m}|V_\alpha| - \Gamma_\alpha & \Delta_\beta &:= 2^{-m}|V_\beta| - \Gamma_\beta. \end{aligned}$$

First, we will compute a lower bound on $\Delta_\alpha + \Delta_\beta$. For this, we define the predicate $b : \{0,1\}^m \to \{0,1\}$ such that $b(s) =$ iff $s \in V_\alpha$. Then,

$$1 - (\Delta_\alpha + \Delta_\beta) = \Gamma_\alpha + \Gamma_\beta = \mathsf{Pr}[R_i = b(S)] \overset{(12)}{\leq} 1 - \frac{1-2\mu}{2n}.$$

Thus, $\Delta_\alpha + \Delta_\beta \geq \frac{1-2\mu}{2n}$.

Hence, $\Delta_\alpha \geq \frac{1-2\mu}{4n}$ or $\Delta_\beta \geq \frac{1-2\mu}{4n}$. First consider the case that $\Delta_\alpha \geq \frac{1-2\mu}{4n}$. For $s \in V_\alpha$, we have $(1-\alpha_s)^2 < (1-\alpha_s)(1-\beta_s) \leq \gamma_s$ from (14), so

$$\begin{aligned} 2^{-m}|V_\alpha| - 2^{-m}\sum_{s\in V_\alpha}\alpha_s = 2^{-m}\sum_{s\in V_\alpha}(1-\alpha_s) &\leq 2^{-m}\sum_{s\in\{0,1\}^m}\sqrt{\gamma_s} \\ &\overset{(*)}{\leq} \sqrt{\sum_{s\in\{0,1\}^m} 2^{-m}\gamma_s} = \sqrt{\Pr[R=\bot]} \leq \sqrt{\nu} \end{aligned} \tag{15}$$

where $(*)$ is an application of Jensen's inequality.

Let $\varepsilon > 0$ be arbitrary. Then, we construct an adversary A^* that corrupts the first party Alice and proceeds as follows. If $s \in V_\alpha$, it executes a strategy that achieves $\mathsf{Pr}\left[R_i^* = 0 \mid S = s\right] \geq \alpha_s - \varepsilon$. (We have to include the error ε since α_s might be a proper supremum.) If $s \notin V_\alpha$, the adversary A^* behaves honestly. We get that

$$\begin{aligned} &\mathsf{Pr}\left[R_i^* = 0\right] - \mathsf{Pr}[R_i = 0] \\ &= \mathsf{Pr}\left[R_i^* = 0 \text{ and } S \in V_\alpha\right] - \mathsf{Pr}[R_i = 0 \text{ and } S \in V_\alpha] \\ &\quad + \mathsf{Pr}\left[R_i^* = 0 \text{ and } S \notin V_\alpha\right] - \mathsf{Pr}[R_i = 0 \text{ and } S \notin V_\alpha] \\ &\overset{(*)}{=} \mathsf{Pr}\left[R_i^* = 0 \text{ and } S \in V_\alpha\right] - \mathsf{Pr}[R_i = 0 \text{ and } S \in V_\alpha] \\ &= \sum_{s\in V_\alpha} \mathsf{Pr}\left[R_i^* = 0 \text{ and } S = s\right] - \Gamma_\alpha \\ &\geq \left(2^{-m}\sum_{s\in V_\alpha}(\alpha_s - \varepsilon)\right) - \Gamma_\alpha \overset{(15)}{\geq} 2^{-m}|V_\alpha| - \sqrt{\nu} - \varepsilon - \Gamma_\alpha = \Delta_\alpha - \sqrt{\nu} - \varepsilon \end{aligned} \tag{16}$$

Here, $(*)$ uses the fact that A^* behaves honestly if $s \notin V_\alpha$, so R and R^* have the same distribution in this case.

On the other hand, we have by the security and non-triviality of π that

$$\Pr[R_i = 0] = 1 - \Pr[R_i = 1] - \Pr[R_i = \bot] \geq 1 - \left(\tfrac{1}{2} + \mu\right) - \nu = \tfrac{1}{2} - \mu - \nu$$

and

$$\Pr[R_i^* = 0] \leq \tfrac{1}{2} + \mu,$$

so

$$\frac{1-2\mu}{4n} \quad -\sqrt{\nu}-\varepsilon \le \Delta_\alpha - \sqrt{\nu} - \varepsilon \overset{(16)}{\le} \Pr[R_i^* = 0] - \Pr[R_i = 0]$$
$$\le \tfrac{1}{2} + \mu - \left(\tfrac{1}{2} - \mu - \nu\right) = 2\mu + \nu.$$

Since this holds for any $\varepsilon > 0$, we have $\frac{1-2\mu}{4n} - \sqrt{\nu} \le 2\mu + \nu$, from which (13) follows immediately.

The case that $\Delta_\beta \ge \frac{1-2\mu}{4n}$ is handled analogously, except that now A^* corrupts Bob and achieves $\mathsf{Pr}\left[R_i^* = 1 \mid S = s\right] \ge \beta_s + \varepsilon$ for all $s \in V_\beta$. This shows Claim 3.

From Claim 3, we immediately get Claim 20, since no pair of negligible functions μ, ν can fulfill (13). □

From Theorem 20, we can directly derive the impossibility of statistical and perfect coin toss extension for both stand-alone simulatability and UC.

Acknowledgements

This work was partially supported by the projects ProSecCo (IST-2001-39227) and SECOQC of the European Commission and by the Cluster of Excellence "Multimodal Computing and Interaction" and by the institutional research funding IUT2-1 of the Estonian Ministry of Education and Research. Part of this work was done while the first and third authors were with the IAKS, University of Karlsruhe. Further, we thank the anonymous referees for valuable comments.

A. An Auxiliary Lemma

Lemma 21. *For any interactive machine M, there is a (not necessarily efficient) interactive machine M' that has the same behavior as M,*[14] *but M' additionally fulfills the following property: In each activation, the output of M' depends only on the input and output M' received so far and on fresh randomness, but not on any internal state.*

Proof. We transform the machine M into M' as follows: In an activation of M', let com denote the communication so far. Let I_{com} denote the inputs of M' in com and O_{com} the outputs. Then, for any possible output x, M' calculates the conditional probability p_x that M gives output x when receiving I_{com} under the condition that it gave outputs O_{com} so far. Then, M' outputs x with probability p_x. By construction, the probability

[14] By having the same behavior, we mean that given a fixed sequence of inputs, the outputs of M and M' have the same probability distribution.

that M' outputs a sequence O_{com} given inputs I_{com} is the same as the probability that P outputs O_{com} given inputs I_{com}. It follows that M and M' behave identically. □

Note The interactive machine M' constructed above can be very inefficient. Hence, removing state may cost efficiency. Alternatively, a machine M can be transformed into a stateless machine M' that has access to an NP oracle [5,21]. (These papers show that an NP oracle allows to uniformly choose a witness among all witnesses of a given NP statement. If we choose the witnesses to be the random coins of M, and the statement to be "M's history is consistent with its inputs and outputs so far," we obtain a stateless machine M' that requires access to an NP oracle.) We thank one anonymous reviewer for pointing out this connection.

References

[1] A. Ambainis, H. Buhrman, Y. Dodis, H. Röhrig, Multiparty quantum coin flipping, in *19th Annual IEEE Conference on Computational Complexity, Proceedings of CCC'04*, (IEEE Computer Society, 2004), Online available at arXiv:quant-ph/0304112. pp. 250–259

[2] M. Backes, D. Hofheinz, J. Müller-Quade, D. Unruh, On fairness in simulatability-based cryptographic systems, in R. Küsters, J. Mitchell, editors, *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, (ACM Press, 2005). Full version as IACR ePrint 2005/294. pp. 13–22

[3] M. Backes, B. Pfitzmann, M. Waidner, Secure asynchronous reactive systems. IACR ePrint Archive, (2004). Online available at http://eprint.iacr.org/2004/082

[4] M. Bellare, J.A. Garay, T. Rabin, Distributed pseudo-random bit generators—a new way to speed-up shared coin tossing, in *Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Proceedings of PODC 2003*, (ACM Press, 1996). Online available at http://www-cse.ucsd.edu/users/mihir/papers/dprg.pdf. pp. 191–200

[5] M. Bellare, O. Goldreich, E. Petrank, Uniform generation of NP-witnesses using an NP-oracle. *Inf. Comput.*, **163**(2), 510–526 (2000).

[6] M. Blum, Coin flipping by telephone, in A. Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, U.C. Santa Barbara Department of Electrical and Computer Engineering, (1981). Online available at http://www-2.cs.cmu.edu/~mblum/research/pdf/coin/. pp. 11–15

[7] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, (IEEE Computer Society, 2001). Full version online available at http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps. pp. 136–145

[8] R. Canetti, M. Fischlin, Universally composable commitments, in J. Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO '01*, vol. 2139 *Lecture Notes in Computer Science*, (Springer-Verlag, 2001). Full version online available at http://eprint.iacr.org/2001/055.ps. pp. 19–40

[9] R. Canetti, Y. Lindell, R. Ostrovsky, A. Sahai, Universally composable two-party and multi-party secure computation, in *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, (ACM Press, 2002). Extended abstract, full version online available at http://eprint.iacr.org/2002/140.ps. pp. 494–503

[10] J.L. Carter, M.N. Wegman, Universal classes of hash functions. *J. Comput. Syst. Sci.*, **18**(2), 143–154 (1979).

[11] R. Cleve, Limits on the security of coin flips when half the processors are faulty, in *Eighteenth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1986*, (ACM Press, 1986). Online available at https://doi.org/10.1145/12130.12168. pp. 364–369

[12] D. Dolev, C. Dwork, M. Naor, Non-malleable cryptography, in *Twenty-Third Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1991*, (ACM Press, 1991). Extended abstract, full version online available at http://www.wisdom.weizmann.ac.il/~naor/PAPERS/nmc.ps. pp. 542–552

[13] O. Goldreich, *Foundations of cryptography—volume 1 (basic tools)*. (Cambridge University Press, 2001). Previous version online available at http://www.wisdom.weizmann.ac.il/~oded/frag.html
[14] 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.
[15] O. Goldreich, Basing non-interactive zero-knowledge on (enhanced) trapdoor permutations: The state of the art. Online available at http://www.wisdom.weizmann.ac.il/~oded/PSBookFrag/nizk-tdp.ps, (Oct. 2009).
[16] O. Goldreich, L.A. Levin, A hard-core predicate for all one-way functions, in *STOC '89: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, (New York, NY, USA, 1989. ACM Press). pp. 25–32
[17] S. Goldwasser, S. Micali, Probabilistic encryption. *J. Comput. Syst. Sci.*, **28**(2), 270–299 (1984).
[18] I. Haitner, T. Holenstein, O. Reingold, S.P. Vadhan, H. Wee, Universal one-way hash functions via inaccessible entropy, in *Advances in Cryptology, Proceedings of EUROCRYPT 2010*, (2010), pp. 616–637
[19] D. Hofheinz, V. Shoup, GNUC: a new universal composability framework. *J. Cryptology*, **28**(3), 423–508 (2015).
[20] R. Impagliazzo, L.A. Levin, M. Luby, Pseudo-random generation from one-way functions, in *Twenty-First Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1989*, (ACM Press, 1989). Online available at https://doi.org/10.1145/73007.73009. pp. 12–24
[21] M. Jerrum, L.G. Valiant, V.V. Vazirani, Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, **43**, 169–188 (1986).
[22] J. Katz, C. Koo, On constructing universal one-way hash functions from arbitrary one-way functions. *IACR Cryptology ePrint Archive*, **2005**, 328 (2005)
[23] T. Moran, M. Naor, G. Segev, An optimally fair coin toss. *J. Cryptology*, **29**(3), 491–513 (2016).
[24] B. Pfitzmann, M. Waidner, A model for asynchronous reactive systems and its application to secure message transmission, in *IEEE Symposium on Security and Privacy, Proceedings of SSP '01*, (IEEE Computer Society, 2001). Full version online available at http://eprint.iacr.org/2000/066.ps. pp. 184–200
[25] J. Rompel, One-way functions are necessary and sufficient for secure signatures, in *Twenty-Second Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1990*, (ACM Press, 1990), pp. 387–394
[26] D.R. Stinson, Universal hash families and the leftover hash lemma, and applications to cryptography and computing. *J. Combin. Math. Combin. Comput.*, **42**, 3–31 (2002). Online available at http://www.cacr.math.uwaterloo.ca/~dstinson/papers/leftoverhash.ps
[27] D. Unruh, Relations among statistical security notions or why exponential adversaries are unlimited, (2006). Available as IACR ePrint 2005/406