On the (Im-)Possibility of Extending Coin Toss[⋆]

Dennis Hofheinz[1], Jörn Müller-Quade[2], and Dominique Unruh[2]

[1] CWI, Cryptology and Information Security Group, Prof. Dr. R. Cramer, `Dennis.Hofheinz@cwi.nl`
[2] IAKS, Arbeitsgruppe Systemsicherheit, Prof. Dr. Th. Beth, Universität Karlsruhe, `{muellerq,unruh}@ira.uka.de`

Abstract. We consider the cryptographic two-party protocol task of extending a given coin toss. The goal is to generate n common random coins from a single use of an ideal functionality which gives $m < n$ common random coins to the 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 which 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 novel protocol for unconditionally secure coin toss extension. The new 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.

Table of Contents

[⋆] This is the full version of the paper presented at Eurocrypt 2006.

1 Introduction

Manuel Blum showed in [Blu81] how to flip a coin over the telephone line. His protocol guaranteed that even if one party does not follow the protocol, the other party still gets a uniformly distributed coin toss result. This general concept of generating common randomness in a way such that no dishonest party can dictate the result proved very useful in cryptography, e.g., in the construction of protocols for general secure multi-party computation.

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."

The first thing the extensibility of a given coin toss depends on is the required security type. One type of security requirement (which we call "stand-alone simulatability" here) can simply be that the protocol imitates an ideal coin toss functionality in the sense of [Gol04], 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 [Can01a, Can05] or the Reactive Simulatability model [PW01, BPW04].

Orthogonal to this, one can vary the level of fulfilment of each of these 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, e.g., 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 depends on the asymptotic behaviour of m.

Our results are summarized in the table below. 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 available by assumption), and **boldface** indicates novel results.

Security type ↓ / level →	Computational	Statistical	Perfect
stand-alone simulatability	yes	**depends**[3]	**no**
universal composability	**depends**[4]	**no**	**no**

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). By Kitaev, this result was extended even to protocols using quantum communication (cf. [ABDR04]). [BGR96] first investigated the problem of extending a coin-toss. They presented a statistically secure protocol for extending a given coin-toss (pre-shared using a VSS), if less than $\frac{1}{6}$ of the parties are corrupted. Note that their main attention was on the efficiency of the protocol, since in that scenario arbitrary multi-party computations and therefore in particular coin-toss from scratch are known to be possible. The result does not apply to the two-party case.

[3] Coin toss extension is possible if and only if the seed has superlogarithmic length.

[4] Coin toss extension is impossible if the seed does not have superlogarithmic length. The possibility result depends on the complexity assumption we use, cf. Section 3.1.

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 (Coro. 8) and for universal composability(Coro. 15). Now for the statistical case. When demanding only stand-alone simulatability, the situation depends on the number of the 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 (Th. 11). Otherwise, we show that there can even be no protocol that derives $m+1$ common random coins (Coro. 8). In the universal composability setting, the situation is more clear: we show that there simply is no protocol that derives from m common coins $m+1$ coins, no matter how large m is (Th. 14). (However, here we restrict to protocols that run in a polynomial number of rounds.)

Known results in the computational case. In [Blu81] Blum gave a computationally secure protocol. In [Gol04, Proposition 7.4.8], this protocol is shown to be stand-alone simulatable, and together with the sequential composition theorem [Gol04, 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 [CF01] that coin-toss cannot be achieved from scratch. However, they showed that given a sufficiently large common reference string (CRS), bit commitment is possible. From this it is easy to see that such a 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 [Cle86], the task of coin-toss is considered in a scenario slightly different from ours: in [Cle86], protocol participants may not abort protocol execution without generating output. In that setting, [Cle86] show that 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 has not been considered so far in the computational setting.

Our results in the computational case. We answer the question concerning the minimal size necessary for a coin-toss to be extensible: If an m-bit coin-toss functionality is given, and m is not superlogarithmic, then it is already impossible for the parties to derive $m+1$ common random coins (in a universally composable way) from it (Th. 6). However, we also show that under strengthened computational assumptions, there are protocols that extend m to any polynomial number (in the security parameter) of common random coins, if m is superlogarithmic (Th. 5). In that sense, we give the remaining parts for a complete characterization of the computational case.

Notation

- A function f is *negligible*, if for any $c > 0$, $f(k) \leq k^{-c}$ for sufficiently large k (i.e., $f \in k^{-\omega(1)}$).
- f is *overwhelming*, if $1 - f$ is negligible (i.e., $f \in 1 - k^{-\omega(1)}$).
- f is *noticeable*, if for some $c > 0$, $f(k) \geq k^{-c}$ for sufficiently large k (i.e., $f \in k^{-O(1)}$). Note that functions exists, which are neither negligible nor noticeable.
- f is *polynomially bounded*, if for some $c > 0$, $f(k) \leq k^{c}$ for sufficiently large k (i.e., $f \in k^{O(1)}$).

- f is *polynomially-large*, if there is a $c > 0$ s.t. $f(k)^c \geq k$ for sufficiently large k (i.e., $f \in k^{\Omega(1)}$).
- f is *superpolynomial*, if for any $c > 0$, $f(k) > k^c$ for sufficiently large k (i.e., $f \in k^{\omega(1)}$).
- 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 (i.e., $f \in (\log k)^{\omega(1)}$).
- f is *exponentially-small*, if there exists a $c > 1$, s.t. $f(k) \leq c^{-k}$ for sufficiently large k (i.e., $f \in \Omega(1)^{-k} = 2^{-\Omega(k)}$).
- f is *subexponential*, if for any $c > 1$, $f(k) < c^k$ for sufficiently large k (i.e., $f \in o(1)^k = 2^{o(k)}$).

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 [Gol04],[5] and Universal Composability (UC) as defined in [Can01a].

Stand-alone simulatability. In [Gol04] 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 [Gol04].

A protocol consists of two parties that alternatingly send messages to each other. The parties may also invoke an ideal functionality, which is given as an oracle (in our cases, they invoke a smaller coin-toss to realise a larger one).

We say the protocol π stand-alone simulatably realises a probabilistic function f, if for any efficient adversary A that may replace none or a single party, there is an efficient simulator S s.t. 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 learn 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, 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 unlimited ones. In the case of statistical stand-alone simulatability we require the real and ideal protocol execution to be statistically indistinguishable (and not only computationally), and in the perfect case we even require these distributions to be identical.

Universal Composability. In contrast to stand-alone simulatability, Universal Composability [Can01a] 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)

[5] In fact, [Gol04] does not use the name stand-alone simulatability but simply speaks about *security in the malicous model*. We adopt the name stand-alone simulatability for this paper to be able to better distinguish the different notions.

and try to distinguish between real and ideal protocol. This additional strictness brings the advantage of a versatile composition theorem (the Universal Composition Theorem [Can01a]).

We only sketch the model here and refer to [Can01a] for details.

A protocol consists of several machines that may (a) get input from the environment, (b) give output to the environment (both 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 a single bit of output. The adversary may choose to corrupt parties at any point in time.[6]

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 only learn and influence protocol data, if (a) the functionality explicitly allows this, or (b) it corrupts a party (note that the simulator may only corrupt the same parties as the adversary). In the latter case, the simulator can choose inputs into the functionality in the name of that party and gets the outputs appartaining to that party. In the case of uncorrupted parties, the environment is in control of the corresponding in- and output of the ideal functionality.

We say a protocol π universally composably (UC)-implements an ideal functionality $\mathcal{F}$ (or short π is universally composable if $\mathcal{F}$ is clear from the context), if for any efficient adversary $\mathcal{A}$, there is an efficient simulator $\mathcal{S}$, s.t. for all efficient environments $\mathcal{Z}$ and all auxiliary inputs z for $\mathcal{Z}$, the distributions of the output-bit of $\mathcal{Z}$ in the real and the ideal protocol execution are indistinguishable.

What has been sketched above we call *computational* UC. We further define *statistical* and *perfect* UC. In these notions, we allow adversary, simulator and environment to be unlimited machines. Further, in the case of perfect UC, we require the distributions of the output-bit of $\mathcal{Z}$ to be *identical* in real and ideal protocol execution.

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. As soon as both parties have this way signalled their willingness to start, the functionality selects n coins in 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.)

If the functionality now sent κ directly and without delay to the parties, this behaviour 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.

A more detailed description follows:

[6] It is then called an *adaptive* adversary. If the adversary can only corrupt parties before the start of the protocol, we speak of *static corruption*. All results in this paper hold for both variants of the security definition.

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.

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

Definition 1. *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 securely and non-trivially implements CT_n by having access only to CT_m. This security can be computational, statistical or perfect.*

By a "non-trivial" implementation we mean a protocol that, with overwhelming probability, guarantees outputs if no party is corrupted and all messages are delivered. (Alternatively, one may also consider protocols that provide output with overwhelming probability.) This requirement is useful since without it, a trivial protocol that does not generate any output formally implements every functionality. (Cf. [CLOS02] and [BHMQU05, Section 5.1] for more discussion and formal definitions of "non-triviality.")

On unlimited simulators. Following [BPW04], we have modelled statistical and perfect stand-alone and UC security using unlimited simulators. Another approach is to require the simulators to be polynomial in the running-time of the adversary. All our results apply also to that case: 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 11. 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 The Computational Case

3.1 Universal Composability

In the following, we need the assumption of enhanced trapdoor permutations with dense public descriptions (called ETD henceforth). Roughly, these are trapdoor permutations with the additional properties that (i) one can choose the public key in an oblivious fashion, i.e., even given the coin tosses we used it is infeasible to invert the function, and (ii) the public keys are computationally indistinguishable from random strings.

Definition 2 (Enhanced trapdoor permutations with dense public descriptions). *A system of* enhanced trapdoor permutations with dense public descriptions (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, satisfying the following:*

For any non-uniform probabilistic polynomial-time algorithm A there is a negligible function μ s.t. the following conditions are satisfied:

– Permutations. $\mathsf{Pr}\left[(pk, td) \leftarrow I(1^k) : pk \textit{ is a valid public key and } td = pk^{-1}\right] \geq 1 - \mu(k)$, *and any valid public key is a permutation.*
– Almost uniform sampling. *For any valid public key pk that can be output by $I(1^k)$, 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.
– Dense public descriptions. *There is a polynomially bounded, efficiently computable function s (not depending on A) s.t.*

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

Exponentially-hard *ETD are defined analogously, except that we require the conditions above to hold for all* subexponential-time *A and an* exponentially-small *μ.*

Lemma 3. *There is a constant $d \in \mathbb{N}$ s.t. the following holds:*

Assume that ETD exist, s.t. the size of the circuits describing the ETD is bounded by $s(k)$ for security parameter k.[7]

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

A protocol for realising bit commitment using a CRS has been given in [CLOS02]. To show this lemma, we only need to review their construction to see, that a CRS of length s^d is indeed sufficient. For details, see Appendix A.2.

Lemma 4. *Let $s(k)$ be a polynomially bounded function, that is computable in time polynomial in k.*

Assume one of the following holds:
– *ETD exist and s is a polynomially-large function.*
– *Exponentially-hard ETD exist and s is a superlogarithmic function.*

Then there also exist a constant $e \in \mathbb{N}$ independent of s and ETD, s.t. the size of the circuits describing the ETD is bounded by $s(k)^e$ for security parameter k.

This is shown by scaling the security parameter of the original ETD. The proof is given in Appendix A.3.

Theorem 5. *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 exist, or*
– *m is super*poly*logarithmic and exponentially-hard ETD exist.*

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

[7] 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 $s(k)$.

Proof. Let d be as in Lemma 3. Let further e be as in Lemma 4. If m is polynomially-large or superpolylogarithmic, then $s := m^{1/(de)}$ is polynomially-large or superlogarithmic, resp. So, by Lemma 4 there are ETD, s.t. the size of the circuits describing the ETD is bounded by $s^e = m^{1/e}$. Then, by Lemma 3 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-realises an n-bit coin-toss (based on the protocol of [Blu81]): 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. Then P_1 unveils the bits r_1. The output of the parties is the n-bit string $r = r_1 \oplus r_2$, where $\oplus$ denotes the bit-wise exclusive or.

It is easy to see, that this protocol UC-realises an n-bit coin-toss. We only roughly 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 $\mathcal{S}$ unveil 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 chose 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 Composition Theorem we can put the above constructions together and get a protocol that UC-realises 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 5 can be decreased. If we assume that for any superlogarithmic m, there are ETD s.t. the size of their circuits is bounded by $m^{1/d}$ (where d is the constant from Lemma 3), we get coin-toss extension even for superlogarithmic m (using the same proof as for Theorem 5, except that instead of Lemma 4 we use the stronger assumption).

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

Theorem 6. *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.*

Proof (sketch). Assume for contradiction that protocol π, with parties P_1 and P_2 using CT_m, implements CT_n (with m, n as in the theorem statement). Let $\mathcal{A}_1$ be an adversary on π that, taking the role of a corrupted party P_1, simply reroutes all communication of P_1 (with either P_2 or CT_m) to the protocol environment $\mathcal{Z}_1$ and thus lets $\mathcal{Z}_1$ take part as P_1 in the real protocol.

Imagine a protocol environment $\mathcal{Z}_1$, running with π and $\mathcal{A}_1$ as above, that keeps and internal simulation $\overline{P_1}$ of P_1 and lets this simulation take part in the protocol (through $\mathcal{A}_1$). After a protocol run, $\mathcal{Z}_1$ inspects the output $\overline{\kappa_1}$ of $\overline{P_1}$ and compares it to the output κ_2 of the uncorrupted P_2.

In a real protocol run with π, $\mathcal{A}_1$, and $\mathcal{Z}_1$, we will have $\overline{\kappa_1} = \kappa_2$ with overwhelming probability since π non-trivially implements CT_n, and CT_n guarantees common outputs. So a simulator $\mathcal{S}_1$, running in the ideal model with CT_n and $\mathcal{Z}_1$, must be able to achieve that the ideal output κ_2 (that is ideally chosen by CT_n and cannot be influenced by $\mathcal{S}_1$) is identical to what the simulation $\overline{P_1}$ of P_1 inside $\mathcal{Z}_1$ outputs. In that sense, $\mathcal{S}_1$ must be able to "convince" $\overline{P_1}$ to also output κ_2. To this end, $\mathcal{S}_1$ may—and must—fake a complete real protocol communication as $\mathcal{A}_1$ would deliver it to $\mathcal{Z}_1$ (and thus, to $\overline{P_1}$).

However, then we can construct another protocol environment $\mathcal{Z}_2$ that expects to take the role of party P_2 in a real protocol run (just like $\mathcal{Z}_1$ expected to take the role of P_1). To this end, an adversary $\mathcal{A}_2$ on π with corrupted P_2 is employed that forwards all communication of P_2 with either P_1 or CT_n to $\mathcal{Z}_2$. Internally, $\mathcal{Z}_2$ now simulates $\mathcal{S}_1$ (and not P_2!) from above and an instance $\overline{\mathsf{CT}_n}$ of the trusted host CT_n. Recall that $\mathcal{S}_1$, given a target string κ by CT_n, mimics an uncorrupted P_2 along with an instance of CT_m. In that situation, $\mathcal{S}_1$ can convince an honest P_1 with overwhelming probability to eventually output κ.

Chances are 2^{-m} that the CT_m-instance made up by $\mathcal{S}_1$ outputs the same seed as the real CT_m in a run of $\mathcal{Z}_2$ with π and $\mathcal{A}_2$. So with probability at least $2^{-m} - \mu$ for negligible μ, in such a run, $\mathcal{Z}_2$ observes a P_1-output κ that is identical to the output of the internally simulated $\overline{\mathsf{CT}_n}$. But then, by assumption about the security of π, there is also a simulator $\mathcal{S}_2$ for $\mathcal{A}_2$ and $\mathcal{Z}_2$ that provides $\mathcal{Z}_2$ with an indistinguishable view. In particular, in an ideal run with $\mathcal{S}_2$ and CT_n, $\mathcal{Z}_2$ observes equal outputs from CT_n and $\overline{\mathsf{CT}_n}$ with probability at least $2^{-m} - \mu'$ for negligible μ'. This is a contradiction, as both outputs are uniformly and independently chosen n-bit strings, and $n \geq m + 1$. □

4 Statistical and Perfect Cases

4.1 Stand-alone simulatability

We start off with a negative result:

Theorem 7. *Let $m < n$ be functions in the security parameter k. If m is not superlogarithmic, there is no two-party n-bit coin-toss protocol π (not even an inefficient one) that uses an m-bit coin-toss and has the following properties:*

- Non-triviality. *If no party is corrupted, the probability that the parties give different, invalid or no output is negligible (by invalid output we mean output not in $\{\mathtt{0},\mathtt{1}\}^n$).*
- Security. *For any (possibly unbounded) adversary corrupting one of the parties there is a negligible function μ, s.t. for every security parameter k and every $c \in \{\mathtt{0},\mathtt{1}\}^n$, the probability for protocol output c is at most $2^{-n} + \mu(k)$.*

If we require perfect non-triviality (the probability for different or no outputs is 0) and perfect security (the probability for a given output c is at most 2^{-n}), such a protocol π does not exist, even if m is superlogarithmic.

Proof (sketch). It is sufficient to consider the case $n = m + 1$.

Without loss of generality, we can assume that the available m-bit coin toss is only used at the end of the protocol. Similarly, we can assume that in the honest case, the parties never output distinct values. A detailed proof for these statements can be found in the full proof.

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, excluding the value of the m-bit coin-toss. We distinguish three sets of complete transcripts: the set $\mathfrak{A}$ of transcripts having non-zero probability for the protocol output $\mathtt{0}^n$, the set $\mathfrak{B}$ of transcripts having zero probability of output $\mathtt{0}^n$ and zero probability that the protocol gives no output, and the set $\mathfrak{C}$ of transcripts having non-zero probability of giving no output. Note that, since for a complete transcript, the protocol output only depends on the m-bit coin-toss, any of the above non-zero probabilities is at least 2^{-m}.

For any partial transcript p (i.e., a situation *during* the run of the protocol), we define three values α, β, 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, $(1-\alpha)(1-\beta) \leq \gamma$. 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 $\max\{1-\alpha, 1-\beta\}$ 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, 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 consider the maximum probability a corrupted Bob can achieve that the protocol output is not 0^n. By the security property, this probability should be at most $(2^n-1)2^{-n}$ plus a negligible amount, which is not overwhelming. However, since every transcript in $\mathfrak{B}$ gives such an output with probability 1, the probability of such is β, which is overwhelming, in contradiction of the security property.

The perfect case is proven similarly. □

The full proof is given in Appendix A.5.

Corollary 8. *By a* non-trivial *coin-toss protocol we mean a protocol s.t. (in the uncorrupted case) the probability that the parties give no or different output is negligible. By a* perfectly non-trivial *coin-toss protocol where this probability is zero.*

Let m be not superlogarithmic *and $n > m$. Then there is no non-trivial protocol realising 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 realising 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 7 and thus, if non-trivial, contradict Theorem 7. Analogously for perfect security. □

However, not all is lost:

Now we will prove that there exists a protocol for coin toss extension from m to n bit which 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 $\min_x - \log \Pr[X = x]$). The randomness from this string is then extracted using a randomness extractor. Interestingly 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.

To obtain the coin toss extension we need a result about randomness extractors able to extract one bit of randomness while leaving the seed reusable like a catalyst.

Lemma 9. *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 a min-entropy of at least t the statistical distance of $s \| h_m(s,x)$ and the uniform distribution on $\{0,1\}^{m+1}$ is at most $2^{-t/2}/\sqrt{2}$.*

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 [CW79], where s is the index selecting from that family. Therefore the Leftover Hash Lemma [ILL89, Sti02] 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 function h_m a simple protocol is possible which extends $m(k)$ coin tosses to $m(k)+1$ if the function $m(k)$ is superlogarithmic.

Theorem 10. *Let $m(k)$ be a superlogarithmic function, then there exists a constant round statistically stand-alone simulatable protocol that realises an $(m+1)$-bit coin-toss using an m-bit coin-toss.*

Proof. Let h_m be as in Lemma 9. Then the following protocol realises a coin toss extension by one bit. Assume $m := m(k)$ where k is the security parameter.

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.

Similar to construction 7.4.7 in [Gol04] the protocol is constructed in a way that the adversary is not able to abort the protocol (not even by not terminating). Hence we can safely assume that the adversary will send some message of the correct length and will invoke the coin toss functionality. We assume the adversary to corrupt P_2, corruption of P_1 is handled analogously. Further we assume the random tape of $\mathcal{A}$ to be fixed in the following. 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 $f_{\mathcal{A}}(a)$. There is no loss in generality if we assume the view of the parties to consists of just a, b, s and the protocol output to be $s\|h_m(s, a\|b)$.

Now for a specific adversary $\mathcal{A}$ with fixed random tape the output distribution of the real protocol (i.e., view and output) is completely described by the following experiment: 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 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 $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) = \tilde{r}_{m+1}$ then 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 to 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 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 9 that the statistical distance between $r = s\|h_m(s, a\|b)$ is bounded by $2^{-m/4-1/2}/\sqrt{2} = (2^{-m})^{1/4}/2$. Since for superlogarithmic m it is 2^{-m} 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 invokations of the adversary are independent, the probability that $h_m(\tilde{s}, \tilde{a}\|\tilde{b}) \neq \tilde{r}_m$ after each activation is neglibigly far from 2^{-k}. So the simulator fails only with negligible probability.

It follows that the real and the ideal protocol execution 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 which extracts a larger amount of randomness (while not necessarily treating the seed like a catalyst). 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 fulfils both the property of computational stand-alone simulatability and of 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 $m(k)$ which is statistically secure *and* computationaly secure, i.e., the simulator for polynomial adversaries is polynomially bounded, too. The basic idea here is to extract one bit at a time in polynomially many rounds.

Theorem 11. *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 that realises an $(m+p)$-bit coin-toss using an m-bit coin-toss.*

Proof. Let h_m be as in Lemma 9. Then the following protocol realises 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 $s\|h_m(s, a_1\|b_1)\| \dots \|h_m(s, a_{p(k)}\|b_{p(k)})$ and output this string.

We only roughly sketch the differences to the proof of Theorem 10. For each protocol round the simulator follows the strategy described in the proof of Theorem 10 (i.e., the simulator rewinds the adversary by *one* round, if the coin-toss produced is not the correct one.) Then using standard hybrid techniques it can be shown that this simulator indeed gives an indistinguishable ideal protocol run. Here it is only noteworthy that we use the fact that $s\|h_m(s, a_1\|b_1)\| \dots \|h_m(s, a_{p(k)}\|b_{p(k)})$ is statistically indistinguishable from the uniform distribution on $m+p$ bits. However, this follows directly from Lemma 9 and the fact that each $a_i\|b_i$ has min-entropy at least $\lfloor \frac{m-1}{2} \rfloor$ even given the values of all $a_\mu\|b_\mu$ for $\mu < i$. □

4.2 Universal Composability (statistical/perfect case)

In the case of statistical security, adversary and protocol environment are allowed to be computationally unbounded. In that case, we show that there is no simulatably secure coin toss extension protocol that runs in a polynomial number of rounds. This is forced by requiring the parties to halt after a polynomial number of activations. However, note that we do not impose any restrictions on the amount of computational work these parties perform in one of those activations.

The proof of this statement is done by contradiction. Furthermore, the proof is split up into an auxiliary lemma and the actual proof. In the auxiliary lemma, we show that without loss of generality, 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.

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 which depends only on the protocol. (As stated above, the parties are still unbounded in each activation.) We start with a helping lemma whose proof is available in Appendix A.6.

Lemma 12. *If there is a non-trivial statistically universally composable protocol for $(m \to n)$-coin toss extension, then there is also one in which each party*

– *has only one connection to the other party and one connection to CT_m,*
– *in each activation sends either an "`init`" message to CT_m or some message to the other 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 view that this party has experienced so far, 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).*

We proceed with

Lemma 13. *There is no non-trivial statistically universally composable protocol for $(m \to n)$-coin toss extension which meets the requirements from Lemma 12.*

Proof. Assume for contradiction that π, using CT_m, is a statistically universally composable implementation of CT_n, and also satisfies the requirements from Lemma 12.

Assume a fixed environment $\mathcal{Z}_0$ that gives both parties "`init`" input and then waits for both parties to output a coin toss result. Consider an adversary $\mathcal{A}_0$ that delivers all messages between the parties immediately. The resulting setting D_0 is depicted in Figure 1.

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.,

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

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 $\kappa_1 = \kappa_2 = f(\mathit{com}, \omega)$ with overwhelming probability.

For a fixed protocol communication $\mathit{com} = c$, consider the set

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

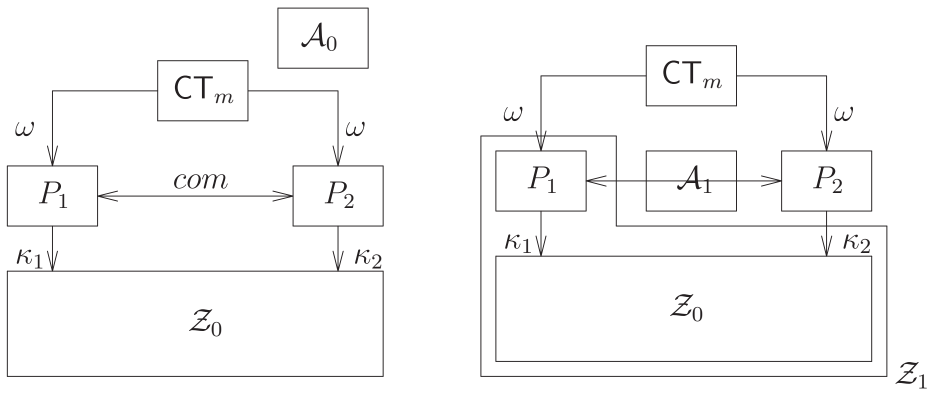

Fig. 1. *Left:* The initial setting D_0 for the statistical case. (Some connections which are not important for our proof have been omitted.) *Right:* Setting D_1 with a corrupted P_1. Setting D_2 (with P_2 corrupted instead of P_1) is defined analogously.

of "improbable outputs" after communication c. Then obviously $|M_c| \geq 2^n - 2^m \geq 2^{n-1}$. By definition of the ideal output (i.e., the output of CT_n in the ideal model), this implies that 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.) Otherwise, an environment could distinguish real and ideal model by testing for $\kappa_1 = \kappa_2 \in M_c$. Since $\mathsf{E}(M_c, \varepsilon)$ is exactly that probability, we have $\mathsf{E}(M_c, \varepsilon) \geq 2/5$ for sufficiently large k. Also, $\mathsf{E}(M_c, c)$ is negligible by definition, so M_c satisfies

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

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 is 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

$$\mathsf{E}(M_c, \overline{c}) - \mathsf{E}(M_c, \overline{c}m) \geq \frac{1}{3p(k)} \tag{2}$$

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.

Note that such an m must be either sent by P_1 or P_2. So there is a $j \in \{1, 2\}$, such that for infinitely many k, party P_j sends such an 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 uncorrupted party P_{3-j}. 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 Figure 1.

Since D_j is basically only a re-grouping of D_0, the random variables com, ω, and κ_i are distributed exactly as in D_0, so we simply identify them. In particular, in D_j, for infinitely many k, there is with probability at least $1/2$ a prefix $\overline{c}$ and a message m *sent by* P_j of com that satisfy (2).

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 for *all* subsets M of $\{0,1\}^n$ whether

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

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

If (3) 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, $\overline{c}$, m, and M refer to these values for which (3) holds.

In any case, after having tossed the coin b once, $\mathcal{Z}'_j$ remembers the set M from (3), and does not check (3) again. After the protocol finishes, $\mathcal{Z}_j$ outputs either $(\perp, \perp)$ (if (3) was never fulfilled), or (b, β) for the evaluation β of the predicate $[\kappa_1 = \kappa_2 \in M]$ (i.e., $\beta = 1$ iff the protocol gives output, the protocol outputs match and lie in M).

Now by our choice of j, $\mathsf{Pr}[b \neq \perp] \geq 1/2$ for infinitely many k.

Also, Lemma 12 guarantees that the internal state of the parties at the time of tossing b consists only of $\overline{c}$. So, when $\mathcal{Z}'_j$ has chosen $b = 1$, and rewound the simulated P_j, the probability that at the end of the protocol $\kappa_1 = \kappa_2 \in M$ is the same as the probability of that event in the setting D_j under the condition that the communication *com* begins with $\bar{c}$. This probability again is exactly $\mathsf{E}(M, \overline{c})$ 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 *com* 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 $\frac{1}{2}\mathsf{E}(M, \overline{c})$ 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 $\frac{1}{2}\big(\mathsf{E}(M, \overline{c}) - \mathsf{E}(M, \bar{c}m)\big) \geq \frac{1}{3p(k)}$.

Since this bound on the difference of the probabilities always holds when $b \neq \perp$, by averaging we get

$$\mathsf{Pr}[\beta = 1 \wedge b = 1 \mid b \neq \perp] - \mathsf{Pr}[\beta = 1 \wedge b = 0 \mid b \neq \perp] \geq \frac{1}{3p(k)}$$

and using the fact that $\mathsf{Pr}[b \neq \perp] \geq \frac{1}{2}$ for infinitely many k we then have that

$$\mathsf{Pr}[\beta = 1 \wedge b = 1] - \mathsf{Pr}[\beta = 1 \wedge b = 0] \geq \frac{1}{6p(k)} \tag{4}$$

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

We show that no simulator $\mathcal{S}_j$ can achieve property (4) 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, e.g., $\tilde{b}$, $\tilde{\beta}$.

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 outputs.

By construction of $\tilde{b}$ and κ, the variable $\tilde{b}$ and the tuple $(\tilde{M}, \kappa)$ are independent given $\tilde{b} \neq \perp$. Hence, since $\tilde{\beta}$ is a function of $\tilde{M}$ and κ,

$$\mathsf{Pr}\left[(\tilde{b}, \tilde{\beta}) = (0, 1)\right] = \mathsf{Pr}\left[(\tilde{b}, \tilde{\beta}) = (1, 1)\right]. \tag{5}$$

So comparing (4) and and (5), $\mathcal{Z}'_j$'s output distribution 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 contradication. □

Combining the above Lemmas 12 and 13 we therefore get:

Theorem 14. *There is no non-trivial statistically universally composable protocol for $(m \to n)$-coin toss extension that proceeds in a polynomial number of rounds.*

In the proof of Lemma 13, we have used that the protocol has only polynomially many rounds only in one place. Namely, we obtained in (2) 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 *non-zero* 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 (4) being non-zero instead of non-negligible. This suffices to show that the considered protocol is not perfectly secure, and thus:

Corollary 15. *There is no non-trivial perfectly universally composable non-trivial 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 unlimited adversaries, simulators and environments, i.e., machines that can implement any probabilistic function, even e.g., the halting problem or similar. Often however, statistical security is instead defined with respect to unlimited 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 [Unr06].

Corollary 16. *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 of rounds and that is statistically universally composable with respect to adversaries / environments / simulators that are unlimited Turing machines.

Proof. [Unr06] show that a bounded-time protocol is universally-composably implements a bounded-time functionality with respect to unlimited adversaries / environments / simulators if and only if it universally-composably implements that functionality with respect to unlimited 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 14. □

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

Acknowledgements. This work was partially supported by the projects ProSecCo (IST-2001-39227) and SECOQC of the European Commission. Part of this work was done while the first author was with the IAKS, Universität Karlsruhe. Further, we thank the anonymous referees for valuable comments.

A Detailed Proofs

A.1 An auxiliary lemma

Lemma 17. *For any interactive machine M, there is a (not necessarily efficient) interactive machine M' that has the same behaviour as M,*[8] *but M' additionally fulfils 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 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. □

A.2 Proof of Lemma 3

Proof (of Lemma 3). The main work (i.e., finding the protocol and proving its security) has been done in [CLOS02]. 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 which is independent of s and the chosen ETD. (In [CLOS02] 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 [CLOS02], there is a protocol UAHC that, assuming a uniform CRS and the existence of ETD, implements multiple commitments. The CRS is assumed to contain the following: (i) 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. (iii) a public key for a CCA2-secure cryptosystem E_{cca}.

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 [GM84, GL89]. 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} (from [DDN91]) is more involved. For this, we first need a non-interactive zero knowledge proof system (NIZK). [Gol01, Constr. 4.10.4 and 4.10.7] together with the additional remarks in [Gol04, C.4.1] present such a scheme, based on 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, $poly(s_3) \cdot m$-parallel executions of the scheme are necessary (using the same trapdoor permutation, see [Gol01, Constr. 4.10.4]) where m is a superlogarithmic

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

function in the security parameter. So if we choose $m := s$, the length of the CRS used by the NIZK scheme is bounded by $poly(s(k) + s_2(k))$.

Another ingredient we need is a universal family of one-way hash functions. In [Rom90] a scheme is presented, that converts a one-way-function f into a universal family of one-way hash functions. Here both description and image of the hash function have a length $s_3 \in poly(s_4)$, where s_4 is the length of the images of f. If we use the f constructed above, $s_4 \leq s$.

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 trapdoor permutation scheme ($2s_4 s$ 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 (which is bounded by s). So the CRS has a length of at 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 $s_3 + 2s_4 s + poly(s + s_4) \subseteq poly(s)$.

Finally, since the protocol UAHC from [CLOS02] 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 $poly(s)$. □

A.3 Proof of Lemma 4

Proof (of Lemma 4). Let I be the key generation algorithm and S be the sampling algorithm of the system for (exponentially-hard) ETD. We now construct a new scheme (I', S') as follows: $I'(k') := I(s(k'))$ and $S' := S$. Since I and S can be described by polynomial-size circuits, (I', S') satisfies the restriction of the circuit size to s^c for some $c \in \mathbb{N}$. It is left to show, that (I', S') is system for ETD.

We will use the following notation: When talking about the original ETD (I, S), we will use the names from Def. 2 (e.g., A, k, μ). When talking about (I', S'), we will add a prime (e.g., A', k', μ').

Let a polynomial-time A' be given. W.l.o.g., we can assume that A' behaves identically for $k' := k'_1$ and $k' := k'_2$ with $s(k_1) = s(k_2)$.

We then construct a machine A as follows: Upon input 1^k, A chooses k' to be the smallest k' with $s(k') = k$ (i.e., $k' := \min s^{-1}(\{k\})$). Then it 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 negligible (or exponentially-small) μ s.t., all conditions in Def. 2 hold. Let $\mu'(k') := \mu(s(k'))$. Then by construction, all the conditions in Def. 2 also hold for A', μ' and the modified system (I', S') (to see this, simply substitute $\tilde{s}(k')$ for k). Since μ' is negligible (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.

Since A runs in time polynomial in $\tilde{k} := \min s^{-1}(\{k\})$ (note that calculating $\tilde{k}$ takes time polynomial in $\tilde{k}$), it is sufficient to show that $\tilde{k}$ is polynomially-bounded (or subexponential, resp.) in k. We distinguish two cases. Case 1: If s is polynomially-large, then there is a c s.t. $s(k')^c \geq k'$ for almost all k. Then we have $s(k') \geq k'^{1/c}$ and then (since $k^{1/c}$ is increasing and invertible) $\tilde{k} = \min s^{-1}(\{k\}) \leq k^c$ for almost all k.

Case 2, s is superlogarithmic: Let $c \in \mathbb{N}$ be arbitrary. Then $2^{s(k')} \geq k'^c$ for sufficiently large k'. It follows $s(k') \geq c \log k'$, and (since $c \log k'$ is increasing and invertible) $\tilde{k} = \min s^{-1}(\{k\}) \leq 2^{k'/c}$ for sufficiently large k. Since this holds for every $c \in \mathbb{N}$, $\tilde{k} \in 2^{o(k)}$, i.e., $\tilde{k}$ is subexponential in k.

It remains to show that μ' is negligible. In the first case (where s is polynomially-large), μ is negligible. We can assume w.l.o.g., that μ is also strictly increasing. Since s is polynomially-large, there is a $c \in \mathbb{N}$, s.t. for sufficiently large k' it is $s(k') \geq k'^{1/c}$. Then $\mu'(k') = \mu(s(k')) \geq \mu(k'^{1/c})$

is negligible. In the second case s is superlogarithmic and μ is exponentially-small. So we can w.l.o.g. assume that $\mu(k) = c^{-k}$ for some $c > 0$ and sufficiently large k. Then we have that $\mu'(k') = (2^{-s(k')})^{\log c}$ for almost all k'. Since $2^{-s(k')}$ is negligible, so is μ'. □

A.4 Proof of Theorem 6

Proof (of Theorem 6). 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_0 of machines as in a real protocol run with corrupted P_1. More specifically, C_0 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 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 $\mathcal{A}_1$. The 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 Figure 2.

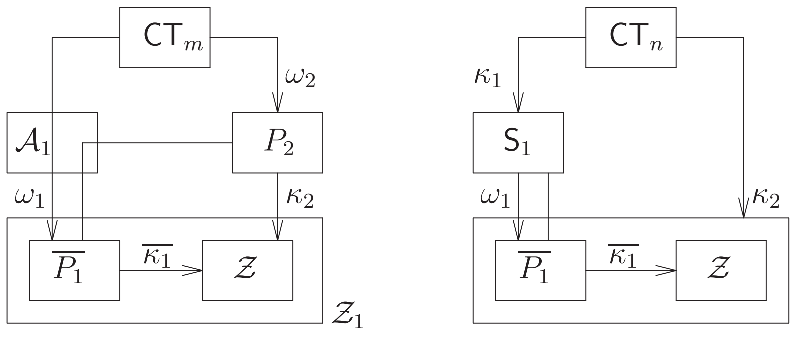

Fig. 2. *Left:* The real protocol with corrupted P_1 and relaying $\mathcal{A}_1$. *Right:* The real protocol with corrupted P_1 and simulator $\mathcal{S}_1$.

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. By simulatability, 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 re-grouping machines. So in C_1, $\mathcal{Z}_1$ gives output 1 with overwhelming probability.

Now by simulatability, 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 Figure 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) to communication from $\overline{P_1}$. 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.

Now we modify network C_2. First, we re-group 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 $\overline{\mathsf{CT}_m}$ that ideally selects and delivers an $\overline{\omega_1}$ to $\overline{P_1}$. Both the $\overline{\omega_2}$ also output by $\overline{\mathsf{CT}_m}$ and the ω_1 still output by $\mathcal{S}_1$ are simply collected by a dummy machine $*$ that does nothing. The resulting collection is called C_3 and depicted in Figure 3.

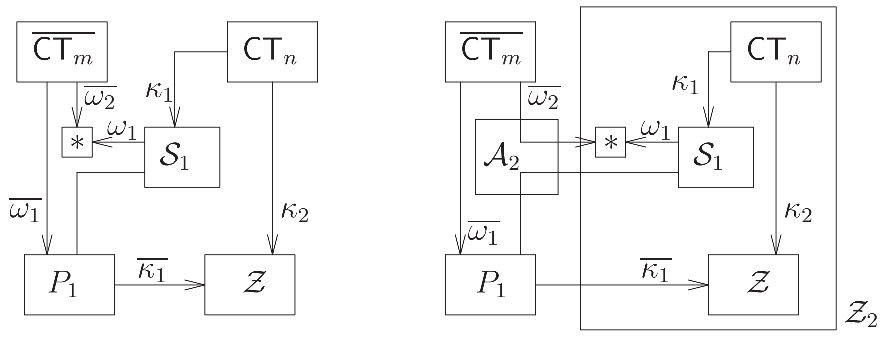

Fig. 3. *Left:* The modification C_3 of C_2. *Right:* The re-grouping C_4 of C_3.

The collections 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} be definition. Since in C_2 $\mathcal{Z}$ gave output 1 with overwhelming probability, it follows that in C_2 the probability is at least $2^{-m} - \mu$ 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 CT_m) into a large protocol environment $\mathcal{Z}_2$. A new real adversary is added that only relays the connection between $\mathcal{S}_1$ to P_1 and the connection between $*$ and $\overline{\mathsf{CT}_m}$. (The connection between $\mathcal{Z}$ and P_1 need not be relayed since this constitutes an input-output connection and no connection for intra-protocol communication.)

This re-grouping of machines gives a new network C_4 (cf. Figure 3). Note that C_4 is only a re-grouping 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 a configuration of the real protocol with environment $\mathcal{Z}_2$ and real adversary $\mathcal{A}_2$; in this case however, this is a configuration in which the adversary $\mathcal{A}_2$ takes the place of a corrupted party P_2.

Now by simulatability, there must be a simulator $\mathcal{S}_2$ that in an ideal setting with $\overline{\mathsf{CT}_n}$ (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.

In particular, this simulator achieves that a 1-output of $\mathcal{Z}$ is at most negligibly less probable than in the real setting C_4. However, in the real setting 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 trusted host 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 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. □

A.5 Proof of Theorem 7

Proof (of Theorem 7). We first consider the statistical (i.e., non-perfect) case. Let us assume that such a π exists.

W.l.o.g., we can assume the following facts about π: (i) If no party is corrupted, both always give the same output or no output (the latter we write as $\perp$). Further, the outcome of the protocol (for the honest parties) is a deterministic function of the messages sent and the value s of the m-bit coin-toss. If this is not the case, we can modify π to contain confirmation messages at the end, where both parties tell each other what they are going to output. If these values do not match, both output $\perp$. (ii) No messages are sent after invoking the m-bit coin-toss. If π sends messages after the m-bit coin-toss, we can transform π s.t. when the m-bit coin-toss s would have been invoked, both parties perform the remainder of the protocol in parallel for each possible value of s (i.e., each message sent consists of 2^m different messages, one for each value of s). At the end, when s is finally chosen, the protocol execution corresponding to s is chosen and the coin-toss. Note that this transformation does not invalidate assumption (i). The resulting protocol is inefficient (unless m is logarithmic), but this does not matter, since we prove the theorem even for inefficient protocols π. (iii) The honest parties maintain no internal state except for the list of the messages sent so far. Since we do not require the parties to be efficient, a machine can be transformed into such a machine without change of behaviour (by Lemma 17). Further, we can w.l.o.g. assume that $n = m + 1$.

We call the parties Alice and Bob.

In the following, by a complete transcript t we mean 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 $f(t, s) \in \{\mathtt{0}, \mathtt{1}\}^n \cup \{\perp\}$ for some deterministic function f. By a partial transcript we mean a prefix of a complete transcript.

We can now distinguish three sets of complete transcripts t: The set $\mathfrak{A}$ of transcripts, where the probability is non-zero that the output $\mathtt{0}^n$ is generated, the set $\mathfrak{B}$ of transcripts, where the probability of output $\mathtt{0}^n$ and of output $\perp$ is zero, and the set $\mathfrak{C}$ of transcripts, where the probability of output $\perp$ is non-zero. Formally:

$$\begin{aligned}
\mathfrak{A} &:= \big\{t : \exists s \in \{\mathtt{0}, \mathtt{1}\}^m : f(t, s) = \mathtt{0}^n\big\} \\
\mathfrak{B} &:= \big\{t : \forall s \in \{\mathtt{0}, \mathtt{1}\}^m : f(t, s) \neq \mathtt{0}^n, f(t, s) \neq \perp\big\} \\
\mathfrak{C} &:= \big\{t : \exists s \in \{\mathtt{0}, \mathtt{1}\}^m : f(t, s) = \perp\big\}
\end{aligned}$$

We now associate to each 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 denote, with what probability 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.

Let now 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 sent a message. Then there is a set M of partial transcripts that can immediately succeed p (one for each message that Alice can send). Furthermore, for each partial transcript $i \in M$, there is a well-defined probability r_i that given an uncorrupted Alice the next partial transcript will indeed be i. It is $\sum_{i \in M} r_i = 1$. Then we have

$$\alpha_p = \max_{i \in M} \alpha_i, \qquad \beta_p = \sum_{i \in M} r_i \beta_i \qquad \gamma_p = \sum_{i \in M} r_i \gamma_i, \tag{6}$$

since a corrupted Alice may choose the partial transcript i that maximises α, while if only Bob or no-one is corrupted, the next partial transcript is chosen accordingly 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 $i \in M$. 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. Then

$$\bar{\alpha}_p \bar{\beta}_p = \sum_i r_i \bar{\alpha}_p \bar{\beta}_i \leq \sum_i r_i \bar{\alpha}_i \bar{\beta}_i \leq \sum_i r_i \gamma_i = \gamma_p$$

(note that since $\sum_{i \in M} r_i = 1$, (6) also holds for $\bar{\beta}_{\dots}$ and $\bar{\gamma}_{\dots}$ instead of $\beta_{\dots}$ and $\gamma_{\dots}$).

Analogous reasoning can be applied when Bob is corrupted. By induction we therefore get $(1-\alpha_p)(1-\beta_p) \leq \gamma_p$ for any partial transcript p. Let $\emptyset$ denote the empty partial transcript, i.e., the beginning of the protocol. Then for $\alpha := \alpha_\emptyset, \beta := \beta_\emptyset, \gamma := \gamma_\emptyset$ it also holds that $(1-\alpha)(1-\beta) \leq \gamma$. Since π was assumed to be non-trivial, the probability that the protocol gives output $\perp$ in the uncorrupted case is negligible. 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 output $\perp$ is at least $2^{-m}\gamma$, so $2^{-m}\gamma$ is negligible, too. Since 2^{-m} is non-negligible, there exists an infinite set K, s.t. 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 $K' \subseteq K$, s.t. $1-\alpha$ *or* $1-\beta$ is negligible on K'.

Let us consider the first case, i.e., α is overwhelming on K'. By assumption, the probability P for protocol output $\mathtt{0}^n$ (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 $\mathtt{0}^n$, 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 $\{\mathtt{0},\mathtt{1}\}^n \setminus \{\mathtt{0}^n\}$ (with corrupted Bob) is at least $2^{-n}(2^n-1)+\mu(2^n-1)$ for some negligible μ. On the other hand, since a complete transcript in $\mathfrak{B}$ has probability 1 of giving output in $\{\mathtt{0},\mathtt{1}\}^n \setminus \{\mathtt{0}^n\}$, 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 $\mathtt{0}^n$ is (for suitable adversary) $\geq 2^{-m} > 2^{-n}$. If $\beta = 1$, the probability for an output in $\{\mathtt{0},\mathtt{1}\}^n \setminus \{\mathtt{0}^n\}$ is $1 > (2^n-1)2^{-n}$. Both cases contradict the security property. □

A.6 Proof of Lemma 12

Proof (of Lemma 12). We split the proof up in several lemmas:

Lemma 18. *If there is a statistically universally composable protocol for $(m \to n)$-coin toss extension, then there is also one in which each party*

- *has only one connection to the other party and one connection to CT_m,*
- *in each activation sends either an "*`init`*" message to CT_m or some message to the other party,*
- *sends in each protocol run at most one message to CT_m, and this is always an "*`init`*" message.*

Proof. This is proven by a straightforward conversion of a statistically universally composable $(m \to n)$-coin toss extension protocol into one that satisfies the lemma requirements. We omit the details. □

Lemma 19. *If there is a statistically universally composable protocol for $(m \to n)$-coin toss extension, then there is also one in which*

- *the parties satisfy the requirements from Lemma 18,*
- *the internal state of each of the two parties consists only of the view that this party has experienced so far.*

Proof. This is a direct consequence of Lemmas 18 and 17. □

Lemma 20. *If there is a statistically universally composable protocol for $(m \to n)$-coin toss extension, then there is also one in which*

- *the parties satisfy the requirements from Lemmas 18 and 19,*
- *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).*

Proof (sketch). First, using Lemmas 18 and 19, we can transform any statistically universally composable $(m \to n)$-coin toss extension protocol into one satisfying the requirements from these lemmas. Call the transformed protocol π (with parties P_1 and P_2). The remaining transformation modifies π such that all requirements from Lemma 20 are met.

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}. This slightly modified protocol π_1 realizes the original protocol π since a simulator (running with π) is informed whenever a party P_i has sent an "`init`" message to CT_m.

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 signalled 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 realizes π_1 (and by transitivity also π) since the modified behaviour 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 $(0 \le j < 2^m)$, 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.[9] Call the protocol with these modified parties π_3.

This protocol obviously fulfills the requirements in the theorem statement, and it only remains to show that π_3 realizes π_2 (and thus π), and hence is a universally composable protocol for coin toss extension. An attack on π_3 can be simulated in the setting of π_2 as follows: up to the point where the first π_2-party queries CT_m, the parties from π_2 behave exactly as those from π_3. But after the first "`init`" query, a simulator running with π_2 needs to simulate the messages of $2^m - 1$ virtual, parallel executions of π_3-parties. This is possible, since by Lemma 19, the internal state of the parties consists only of the received communication so far and is known to the simulator. $\square$

References

[ABDR04] Andris Ambainis, Harry Buhrman, Yevgeniy Dodis, and Heinz Röhrig. Multiparty quantum coin flipping. In *19th Annual IEEE Conference on Computational Complexity, Proceedings of CCC'04*, pages 250–259. IEEE Computer Society, 2004. Online available at `http://arxiv.org/abs/quant-ph/0304112`.

[BGR96] Mihir Bellare, Juan A. Garay, and Tal 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*, pages 191–200. ACM Press, 1996. Online available at `http://www-cse.ucsd.edu/users/mihir/papers/dprg.pdf`.

[BHMQU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On fairness in simulatability-based cryptographic systems. In Ralf Küsters and John Mitchell, editors, *Proceedings of the 2005 ACM Workshop on Formal Methods in Security Engineering*, pages 13–22. ACM Press, 2005. Full version as IACR ePrint 2005/294.

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, pages 11–15. U.C. Santa Barbara Dept. of Elec. and Computer Eng., 1981. Online available at `http://www-2.cs.cmu.edu/~mblum/research/pdf/coin/`.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure asynchronous reactive systems. IACR ePrint Archive, March 2004. Online available at `http://eprint.iacr.org/2004/082.ps`.

[Can01a] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[Can01b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[Can05] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. IACR ePrint Archive, December 2005. Full and revised version of [Can01b], online available at `http://eprint.iacr.org/2000/067.ps`.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

[Cle86] Richard 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*, pages 364–369. ACM Press, 1986. Online available at `http://doi.acm.org/10.1145/12130.12168`.

[9] 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.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[CW79] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Twenty-Third Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1991*, pages 542–552. ACM Press, 1991. Extended abstract, full version online available at `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/nmc.ps`.

[GL89] O. Goldreich and 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*, pages 25–32, New York, NY, USA, 1989. ACM Press.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Gol01] Oded Goldreich. *Foundations of Cryptography – Volume 1 (Basic Tools)*. Cambridge University Press, August 2001. Previous version online available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[Gol04] Oded 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`.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Twenty-First Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1989*, pages 12–24. ACM Press, 1989. Online available at `http://doi.acm.org/10.1145/73007.73009`.

[PW01] Birgit Pfitzmann and Michael 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*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

[Rom90] John 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*, pages 387–394. ACM Press, 1990.

[Sti02] Douglas Robert 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`.

[Unr06] Dominique Unruh. Relations among statistical security notions or why exponential adversaries are unlimited, 2006. Available as IACR ePrint 2005/406.