Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation*

Yehuda Lindell

Dept. of Computer Science and Applied Math.
The Weizmann Institute of Science
Rehovot 76100, Israel.
lindell@wisdom.weizmann.ac.il

October 9, 2003

Abstract

In this paper we show that any *two-party* functionality can be securely computed in a *constant number of rounds*, where security is obtained against (polynomial-time) malicious adversaries that may arbitrarily deviate from the protocol specification. This is in contrast to Yao's constant-round protocol that ensures security only in the face of semi-honest adversaries, and to its malicious adversary version that requires a polynomial number of rounds.

In order to obtain our result, we present a constant-round protocol for secure coin-tossing of polynomially many coins (in parallel). We then show how this protocol can be used in conjunction with other existing constructions in order to obtain a constant-round protocol for securely computing any two-party functionality. On the subject of coin-tossing, we also present a constant-round *almost perfect* coin-tossing protocol, where by "almost perfect" we mean that the resulting coins are guaranteed to be statistically close to uniform (and not just pseudorandom).

Keywords: Secure computation, Constant-round protocols, Coin-tossing.

*An extended abstract of this work appeared in *CRYPTO 2001*; the full version appeared in the *Journal of Cryptology,* 16(3):143-184, 2003. The journal version contains an error in the proof of Lemma 3.1; the error is fixed here.

1 Introduction

1.1 Secure Two-Party Computation

In the setting of two-party computation, two parties with respective private inputs x and y, wish to jointly compute a functionality $f(x, y) = (f_1(x, y), f_2(x, y))$, such that the first party receives $f_1(x, y)$ and the second party receives $f_2(x, y)$. This functionality may be probabilistic, in which case $f(x, y)$ is a random variable. Loosely speaking, the security requirements are that nothing is learned from the protocol other than the output (*privacy*), and that the output is distributed according to the prescribed functionality (*correctness*). The actual definition [22, 27, 3, 11] blends these two conditions (see Section 2.2). The above security must be guaranteed even when one of the parties is adversarial. Such an adversary may be *semi-honest* (or passive), in which case it correctly follows the protocol specification, yet attempts to learn additional information by analyzing the transcript of messages received during the execution. On the other hand, an adversary may be *malicious* (or active), in which case it can arbitrarily deviate from the protocol specification.

The first general solutions for the problem of secure computation were presented by Yao [33] for the two-party case (with security against semi-honest adversaries) and Goldreich, Micali and Wigderson [25] for the multi-party case (with security even against malicious adversaries). Thus, despite the stringent security requirements placed on such protocols, wide-ranging completeness results were established, demonstrating that *any* probabilistic polynomial-time functionality can be securely computed (assuming the existence of trapdoor permutations).

Yao's protocol. In [33], Yao presented a *constant-round* protocol for securely computing any functionality, where the adversary may be semi-honest. Denote Party 1 and Party 2's respective inputs by x and y and let f be the functionality that they wish to compute (for simplicity, assume that both parties wish to receive $f(x, y)$). Loosely speaking, Yao's protocol works by having one of the parties (say Party 1) first generate an "encrypted" circuit computing $f(x, \cdot)$ and send it to Party 2. The circuit is such that it reveals nothing in its encrypted form and therefore Party 2 learns nothing from this stage. However, Party 2 can obtain the output $f(x, y)$ by "decrypting" the circuit. In order to ensure that Party 2 learns nothing more than the output itself, this decryption must be "partial" and must reveal $f(x, y)$ only. Without going into unnecessary details, this is accomplished by Party 2 obtaining a series of keys corresponding to its input y, such that given these keys and the circuit, the output value $f(x, y)$ (and only this value) may be obtained. Of course, Party 2 must obtain these keys without revealing anything about y and this can be done by running $|y|$ instances of a (semi-honest) secure 2-out-of-1 Oblivious Transfer protocol [14], which is constant-round. By running the Oblivious Transfer protocols in parallel, this protocol requires only a constant number of rounds.

Now consider what happens if Yao's protocol is run when the adversary may be malicious. Firstly, we have no guarantee that Party 1 constructed the circuit so that it correctly computes $f(x, \cdot)$. Thus, *correctness* may be violated. Secondly, the Oblivious Transfer protocol must satisfy the requirements for secure computation (in the face of malicious adversaries), and must maintain its security when run in parallel. We know of no such (highly secure) oblivious transfer protocol that runs in a constant number of rounds. Finally, if the functionality f is *probabilistic*, then Party 1 must be forced to input a truly random string into the circuit. Thus, some type of coin-tossing protocol is also required.

Secure protocol compilation. As we have mentioned, Goldreich, Micali and Wigderson [24, 25] showed that assuming the existence of trapdoor permutations, there exist protocols for securely computing any multi-party functionality, where the adversary may be malicious. They achieve this in two stages. First, they show a protocol for securely computing any functionality in the semi-honest adversarial model. Next, they construct a *protocol compiler* that takes any semi-honest protocol and "converts" it into a protocol that is secure in the malicious model. As this compiler is generic, it can be applied to *any* semi-honest protocol and in particular, to the constant-round two-party protocol of Yao. However, due to the nature of their compilation, the output protocol is no longer constant-round.

1.2 Our Results

The focus of this paper is to construct a protocol compiler such that the round-complexity of the compiled protocol is of the same order as that of the original protocol. We observe that the only component of the GMW compiler for which there does *not* exist a constant-round construction is that of coin-tossing in the well [8]. In particular, in the GMW compiler the coins are tossed sequentially, and thus polynomially many rounds are required. Our technical contribution is therefore in constructing a constant-round protocol for coin-tossing in the well *of polynomially many coins.* That is, we obtain the following theorem (informally stated):

Theorem 1 (constant-round coin-tossing): *Assuming the existence of one-way functions, there exists a* constant-round *protocol for the coin-tossing functionality* (*as required by the* GMW *compiler*)*.*

In order to construct such a constant-round protocol we introduce a technique relating to the use of commitment schemes, which we believe may be useful in other settings as well. Commitment schemes are a basic building block and are used in the construction of many protocols. Consider, for example, Blum's protocol for coin-tossing a single bit [8]. In this protocol, Party 1 sends a commitment to a random-bit; then, Party 2 replies with its own random bit and finally Party 1 decommits. Loosely speaking, the security of such a protocol is guaranteed by providing a simulator who receives a uniformly chosen bit and "interacts" with the adversarial party in order to generate a transcript of the protocol execution that is consistent with this bit. The difficulty in simulating this protocol with an adversarial Party 2 is that the simulator only knows the correct value to commit to *after* Party 2 sends its message. However, since the simulator is bound to its commitment, it must somehow guess the correct value *before* this message is sent. In case the messages are long (say n bits rather than a single bit or $\log n$ bits), this may be problematic. Thus, rather than decommitting, we propose to have the party reveal the committed value and then prove (in zero-knowledge) the validity of this revealed value. In a real execution, this is equivalent to decommitting, since the committing party is effectively bound to the committed value by the zero-knowledge proof. However, the simulator is able to provide a *simulated* zero-knowledge proof (rather than a real one). Furthermore, this proof remains indistinguishable from a real proof even if the revealed value is incorrect (and thus the statement is false). Therefore, the simulator can effectively "decommit" to any value it wishes and is not bound in any way by the original commitment that it sends.

Combining the constant-round protocol of Theorem 1 with other known constructions, we obtain the following theorem:

Theorem 2 *Assume the existence of one-way functions. Then, there exists a protocol compiler that given a two-party protocol* Π *for securely computing* f *in the semi-honest model produces a two-party protocol* Π' *that securely computes* f *in the malicious model, so that the number of rounds of communication in* Π' *is within a* constant multiplicative factor *of the number of rounds of communication in* Π*.*

We stress that, when ignoring the "round preservation" clause, the existence of a protocol compiler is not new and has been shown in [24, 25] (in fact, as we have mentioned, we use most of the components of their compiler). Our contribution is in reducing the overhead of the compiler, in terms of the round-complexity, to a constant factor. The main result, stated in the following theorem, is obtained by applying the compiler of Theorem 2 to the constant-round protocol of Yao.

Theorem 3 *Assuming the existence of trapdoor permutations, any two-party functionality can be securely computed in the malicious model in a* constant number of rounds.

While on the subject of coin-tossing, we also present a constant-round protocol for the "almost perfect" coin-tossing (of polynomially many coins) that guarantees that the output of the coin-tossing protocol is statistically close to uniform, and not just computationally indistinguishable.

1.3 Related Work

Although generic feasibility results for the problem of secure computation have been established, these protocols are not very efficient. For example, in the [25] protocol, both the number of rounds and communication complexity are polynomial in the size of the circuit computing the functionality. Thus some research has focused on finding efficient protocols for *specific* problems of secure computation. See [9, 12, 15, 31] for just a few examples. This direction is not the focus of our work.

Other research has considered the efficiency of generic solutions themselves and as such also addresses fundamental questions regarding efficiency considerations (e.g., the possibility of obtaining protocols with only a constant number of rounds or with sub-linear communication complexity). Specifically, in the setting of multi-party computation with an honest majority, Beaver, Micali and Rogaway [5] showed that any functionality can be securely computed in a constant number of rounds, where the adversary may be malicious. Unfortunately, their techniques rely heavily on the fact that a majority of the parties are honest and as such cannot be applied to the case of two-party protocols. The above addresses the issue of the round complexity of protocols. Another important question relates to the communication complexity (in terms of the bandwidth) of secure protocols. See [29] for a recent work in this direction.

As we have described, in this paper we establish the analogous result of [5] for the setting of *two*-party computation.

1.4 Organization

In Sections 2 and 3, definitions and the cryptographic tools used in our protocols are presented (most of these are standard, although Section 3.2 contains some important discussions and Section 3.3 contains a new technical lemma, see below). Then, in Section 4 we discuss the protocol compiler of Goldreich, Micali and Wigderson and observe that in order to achieve "round-preserving" compilation, one needs only to construct a constant-round coin-tossing protocol. Our technical contribution in this paper thus begins in Section 5 where we present such a constant-round coin-tossing protocol. Then, in Section 6 we present a secure protocol for "almost perfect" coin-tossing, for which the output is guaranteed to be statistically close to uniform (and not just pseudorandom).

In addition, we provide two useful technical results. In Section 3.3 we present a general lemma that can be used to simplify the analysis of protocols that use zero-knowledge proofs of knowledge as subprotocols (as indeed we do in our coin-tossing protocols). Also, in Appendix A we consider zero-knowledge arguments and arguments of knowledge in a setting where the adversarial party may run in *expected* polynomial-time (rather than being limited to strict polynomial-time). In particular, we show that the zero-knowledge arguments of knowledge of [17] remain zero-knowledge in this setting, whereas the zero-knowledge proof system of [20] seems not to.

2 Definitions

Most of the definitions presented below are standard. The only difference is that in the setting of secure computation, we allow the adversary to run in *expected* polynomial-time (rather than strict polynomial-time).

2.1 Preliminaries

In this section we present some basic definitions and notations. We begin by recalling the definitions of statistical closeness and computational indistinguishability.

Definition 4 (Statistical Closeness): *Let $S \subseteq \{0,1\}^*$ be a set of strings (S is called the index set). Let $\{X_s\}_{s\in S}$ and $\{Y_s\}_{s\in S}$ be probability ensembles, so that for any $s \in S$ the distribution X_s (resp., Y_s) ranges over strings of length polynomial in $|s|$. We say that the ensembles are* statistically close*, denoted $\{X_s\} \stackrel{\rm s}{\equiv} \{Y_s\}$, if for every polynomial $p(\cdot)$ and all sufficiently long $s \in S$*

$$\sum_{\alpha} |\Pr[X_s = \alpha] - \Pr[Y_s = \alpha]| < \frac{1}{p(|s|)}$$

Definition 5 (Computational Indistinguishability): *Let S, $\{X_s\}_{s\in S}$ and $\{Y_s\}_{s\in S}$ be as above. We say that the ensembles are* computationally indistinguishable*, denoted $\{X_s\} \stackrel{\rm c}{\equiv} \{Y_s\}$, if for every probabilistic polynomial time distinguisher D, every polynomial $p(\cdot)$, all sufficiently long $s \in S$ and all auxiliary information $z \in \{0,1\}^{\mathrm{poly}(n)}$*

$$|\Pr[D(X_s, s, z) = 1] - \Pr[D(Y_s, s, z) = 1]| < \frac{1}{p(|s|)}$$

We denote the uniform distribution over strings of length m by U_m. Thus a distribution ensemble $\{X_n\}$ ranging over strings of length m is said to be pseudorandom if it is computationally indistinguishable from $\{U_m\}$. (When considering ensembles indexed by N, we associate N and $\{1^n : n \in \mathsf{N}\}$.) For a set S, we denote $s \in_R S$ when s is chosen *uniformly* from S. Finally, the security parameter is denoted by n.

Uniform versus non-uniform presentation. For the sake of simplicity, the definition of secure computation is presented in the non-uniform model of computation (in line with the presentation of [18]). However, the proof of the coin-tossing protocol itself is in the uniform model of computation. That is, the protocol is shown to be secure when the adversary is any probabilistic (expected) polynomial-time machine. This enables us to assume a weaker assumption; namely that of one-way functions that are hard to invert for uniform machines. However, in such a case, security is only obtained for a weaker, uniform adversary. Nevertheless, our proof also implies security in the non-uniform model, and thus security can be formulated against non-uniform adversaries and assuming one-way functions that are hard to invert for non-uniform machines.

Expected polynomial-time machines. In this work, we consider adversaries that are modeled by expected polynomial-time interactive machines. An interactive machine M_1 is expected polynomial-time if there exists a (single) polynomial $p(\cdot)$ such that for *every* (possibly unbounded) machine M_2, the expected running time of M_1 (upon input of length n) when interacting with M_2 is bounded by $p(n)$.

Discussion - expected polynomial-time. An alternative definition to the above one would be to say that the adversary must run in expected polynomial-time when interacting with an honest party only, but is not restricted in the case that it interacts with any other machine. This is the definition of expected polynomial-time suggested by Feige [16, Section 3.3]. However, Feige shows that obtaining zero-knowledge against such expected polynomial-time verifiers is problematic. We therefore adopt the above definition, for which we *can* obtain zero-knowledge, as is shown in Appendix A.

We stress that we do not claim that a definition allowing for expected polynomial-time adversaries is "superior" to an analogous definition that restricts adversaries to strict polynomial-time. In fact, we personally adopt the view that expected polynomial-time is very problematic and it is far preferable to stay within the relm of strict polynomial-time (without adopting either of the above definitions). Nevertheless, we consider expected polynomial-time here for the following reasons:

1. At the time that this work was conducted, no constant-round zero-knowledge protocols with *strict* polynomial-time simulators were known to exist. Likewise, all extractors for constant-round zero-knowledge proofs of knowledge ran in expected polynomial-time. Subsequent to this work, constant-round protocols with strict polynomial-time simulators and extractors were demonstrated [1, 2]. Thus, relying on that work, the results of this paper can be modified to consider strict polynomial-time adversaries only. We note, however, that [1, 2] assume the existence of trapdoor clawfree permutations, whereas the protocols we use (with expected polynomial-time simulators and extractors) assume only the existence of one-way functions.

2. Despite the problematic nature of equating efficient computation with expected polynomial-time, we believe that it is important to have rigorous proofs of security also in this model. We feel that the results presented in Appendix A justify this view.

We conclude by noting that the standard way of bypassing this issue is to restrict the adversary to strict polynomial-time, and allow for its simulation to take expected polynomial-time. Thus, the security provided states that anything the adversary could see in a strict polynomial-time attack on the protocol, it could generate in expected polynomial-time. On the other hand, in this work we prove that anything that the adversary could see in an expected polynomial-time attack, it could generate in that same time. This seems to be a preferable security claim.

2.2 Secure Computation

In this section we present the definition for secure two-party computation. The following description and definition is based on [18], which in turn follows [22, 27, 3, 11].

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$ and the second party (with input y) wishes to obtain $f_2(x, y)$.[1] We often denote such a functionality by $(x, y) \mapsto (f_1(x, y), f_2(x, y))$. Thus, for example, the basic coin-tossing functionality is denoted by $(1^n, 1^n) \mapsto (U_n, U_n)$.

Adversarial behavior. Loosely speaking, the aim of a secure two-party protocol is to protect an honest party against dishonest behavior by the other party. This "dishonest behavior" can manifest itself in a number of ways; in particular, we focus on what are known as *semi-honest* and *malicious* adversaries. A semi-honest adversary follows the prescribed protocol, yet attempts to learn more information than "allowed" from the execution. Specifically, the adversary may record the entire message transcript of the execution and attempt to learn something beyond the protocol output. On the other hand, a malicious adversary may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol, may substitute its local input (and enter with a different input) and may abort the protocol prematurely. One ramification of the adversary's ability to abort, is that it is impossible to achieve "fairness". That is, the adversary may obtain its output while the honest party does not.[2]

As is standard for two-party computation, in this work we consider a static corruption model, where one of the parties is adversarial and the other is honest, and this is fixed before the execution begins. (This is in contrast to an adaptive corruption model where an adversary can corrupt one or both of the parties during the execution, based on what it sees. Such a model makes sense for two-party computation when one considers a large network of users in which pairs of parties run two-party protocols.)

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation.

Execution in the ideal model. The ideal model differs for semi-honest and malicious parties. First, for semi-honest parties, an ideal execution involves each party sending their respective input to the trusted party and receiving back their prescribed output. An honest party then outputs this output, whereas a semi-honest party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message it obtained from the trusted party. (See [18] for a formal definition of the ideal and real models for the case of semi-honest adversaries.)

We now turn to the ideal model for malicious parties. Since some malicious behavior cannot be prevented (for example, early aborting), the definition of the ideal model in this case is somewhat more involved. An ideal execution proceeds as follows:

Inputs: Each party obtains an input, denoted z.

[1] Another way of defining f is as a deterministic function $f : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where the third input is uniformly chosen. That is, first a uniformly distributed string r is chosen, and then the first and second parties receive $f_1(x, y, r)$ and $f_2(x, y, r)$, respectively.

[2] We note that although complete fairness is not possible to achieve, there are protocols that obtain "approximate" fairness [22, 4]. In this work we adopt the definition where fairness is not required.

Send inputs to trusted party: An honest party always sends z to the trusted party. A malicious party may, depending on z, either abort or sends some $z' \in \{0,1\}^{|z|}$ to the trusted party.

Trusted party answers first party: In case it has obtained an input pair, (x, y), the trusted party (for computing f), first replies to the first party with $f_1(x, y)$. Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol $\perp$.

Trusted party answers second party: In case the first party is malicious it may, depending on its input and the trusted party's answer, decide to *stop* the trusted party. In this case the trusted party sends $\perp$ to the second party. Otherwise (i.e., if not stopped), the trusted party sends $f_2(x, y)$ to the second party.

Outputs: An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Let $f : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1\}^* \times \{0,1\}^*$ be a functionality, where $f = (f_1, f_2)$, and let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic *expected polynomial-time* machines (representing parties in the ideal model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the honest party instructions in the above-described ideal execution). Then, the joint execution of f under $\overline{M}$ in the ideal model (on input pair (x, y)), denoted $\text{IDEAL}_{f,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 from the above ideal execution. For example, in the case that M_1 is malicious and always aborts at the outset, the joint execution is defined as $(M_1(x, \perp), \perp)$. Whereas, in case M_1 never aborts, the joint execution is defined as $(M_1(x, f_1(x', y)), f_2(x', y))$ where $x' = M_1(x)$ is the input that M_1 gives to the trusted party.

Execution in the real model. We next consider the real model in which a real (two-party) protocol is executed (and there exists no trusted third party). In this case, a malicious party may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform expected polynomial-time machines. In particular, the malicious party may abort the execution at any point in time (and when this happens prematurely, the other party is left with no output).

Let f be as above and let Π be a two-party protocol for computing f. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of non-uniform probabilistic *expected polynomial-time* machines (representing parties in the real model). Such a pair is admissible if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the strategy specified by Π). Then, the joint execution of Π under $\overline{M}$ in the real model (on input pair (x, y)), denoted $\text{REAL}_{\Pi,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

Definition 6 (security in the malicious model): *Let f and Π be as above. Protocol Π is said to* securely compute f (in the malicious model) *if for every pair of admissible non-uniform probabilistic expected polynomial-time machines $\overline{A} = (A_1, A_2)$ for the real model, there exists a pair of admissible* *non-uniform probabilistic expected polynomial-time machines* $\overline{B} = (B_1, B_2)$ *for the ideal model, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(x,y)\right\}_{x,y \text{ s.t. } |x|=|y|}$$

We note that the above definition assumes that the parties know the input lengths (this can be seen from the requirement that $|x| = |y|$). Some restriction on the input lengths is unavoidable, see [18, Section 2.1] for discussion.

Remark: The above definition is different from the standard definition in that the adversary (in both the ideal and real models) is allowed to run in *expected* polynomial-time (rather than *strict* polynomial-time). This seems to be inevitable given that currently known constant-round zero-knowledge proofs require *expected* polynomial-time simulation.[3] We note that the honest party strategies in our protocols run in strict polynomial-time only.

3 Cryptographic Tools

In this section we provide (informal) definitions of perfectly-binding and perfectly-hiding commitment schemes, and present the definition of zero-knowledge arguments of knowledge. All the above is standard. However, Section 3.2 also contains important discussions that we recommend not skipping (even for the familiar reader), and Section 3.3 contains a new technical lemma that we rely on in proving the security of our coin-tossing protocol.

3.1 String Commitment

3.1.1 Perfectly binding commitment schemes

Commitment schemes are a basic ingredient in many cryptographic protocols. They are used to enable a party, known as the *sender*, to commit itself to a value while keeping it secret from the *receiver* (this property is called hiding). Furthermore, the commitment is binding, and thus in a later stage when the commitment is opened, it is guaranteed that the "opening" can yield only a single value determined in the committing phase. In a *perfectly binding* commitment scheme, the binding property holds even for an all-powerful sender, while the hiding property is only guaranteed with respect to a polynomial-time bounded receiver.

For simplicity, we present the definition for a non-interactive, commitment scheme for a single bit. String commitment can be obtained by separately committing to each bit in the string. We denote by $C(b;r)$ the output of the commitment scheme C upon input $b \in \{0,1\}$ and using the random string $r \in_R \{0,1\}^n$ (for simplicity, we assume that C uses n random bits where n is the security parameter).

Definition 7 (perfectly-binding bit commitment): *A* perfectly binding commitment scheme *is a probabilistic algorithm* C *satisfying the following two conditions:*

1. Perfect Binding: $C(0;r) \neq C(1;s)$ *for every* $r, s \in \{0,1\}^n$ *and* $n \in \mathsf{N}$.
2. Computational Hiding: *The probability ensembles* $\{C(0;U_n)\}_{n\in\mathsf{N}}$ *and* $\{C(1;U_n)\}_{n\in\mathsf{N}}$ *are computationally indistinguishable.*

[3] As we have mentioned, subsequent to this work constant-round zero-knowledge arguments and arguments of knowledge with strict polynomial-time simulation and extraction were presented in [1, 2].

Non-interactive perfectly-binding commitment schemes can be constructed using any 1–1 one-way function (see [19, Section 4.4.1]). Allowing some minimal interaction (in which the receiver first sends a single message), almost perfectly-binding commitment schemes can be obtained from any one-way function [28].

3.1.2 Perfectly hiding commitment schemes

We now informally describe the requirements for a *perfectly hiding* commitment scheme. In such a scheme, the binding property is guaranteed to hold only with respect to a polynomial-time sender. On the other hand, the hiding property is unconditional. That is, the distributions of commitments to 0 and commitments to 1 are identical, and thus even an all-powerful receiver cannot know the value committed to by the sender. We stress that the binding property guarantees that a cheating probabilistic polynomial-time sender can find only one decommitment, even though decommitments to both 0 and 1 exist. See [19, Section 4.8.2] for a full definition.

Perfectly hiding commitment schemes can be constructed from any one-way permutation [30]. However, *constant-round* schemes are only known to exist under seemingly stronger assumptions; specifically that of collision-resistant hash functions [32, 13] or families of certified clawfree functions (see [19, Section 4.8.2.3]). (We note that certified clawfree functions exist under the discrete-log assumption.)

3.2 Constant Round Zero-Knowledge Arguments of Knowledge

Feige and Shamir [17] show that there exist constant-round zero-knowledge arguments (with negligible error probability) for membership in any $\mathcal{NP}$-language, assuming that there exist one-way functions. Furthermore their argument system is actually a system of *arguments of knowledge*.

We first recall the definition of zero-knowledge proof systems [23]. Below, we denote by $\langle P, V^* \rangle(x)$ the output of V^* after interacting with P on common input x.

Definition 8 (auxiliary-input zero-knowledge): *Let (P, V) be an interactive proof for a language L; denote by $P_L(x)$ the set of strings y satisfying the completeness condition with respect to $x \in L$ (i.e., $\Pr[\langle P(y_x), V(z)\rangle(x) = 1] = 1$ for every $z \in \{0,1\}^*$). We say that (P, V) is* auxiliary-input zero-knowledge *if for every probabilistic* expected *polynomial-time machine V^*, there exists a probabilistic algorithm M^*, running in expected-time that is polynomial in its first input, so that the following two probability ensembles are computationally indistinguishable:*

1. $\{\langle P(y_x), V^*(z)\rangle(x)\}_{x \in L, z \in \{0,1\}^*}$ *for an arbitrary $y_x \in P_L(x)$.*
2. $\{M^*(x, z)\}_{x \in L, z \in \{0,1\}^*}$

In the above definition, z represents the auxiliary information given to the verifier. Note that whereas P and V^* above are interactive strategies, M^* is a non-interactive machine. We now present the definition for arguments of knowledge (as defined in [7, 19]):

Definition 9 (system of arguments of knowledge): *Let R be a binary relation and $\kappa : \mathsf{N} \mapsto [0, 1]$. We say that a probabilistic polynomial-time V is a* knowledge verifier for the relation R *with* knowledge error κ *if the following two conditions hold:*

- Non-triviality: *There exists a probabilistic polynomial-time interactive machine P so that for every $(x, y) \in R$, all possible interactions of V with $P(y)$ on common-input x are accepting.*

- Validity (or knowledge soundness) with error $\kappa(\cdot)$: *There exists a polynomial $q(\cdot)$ and a probabilistic oracle machine K, called a* knowledge extractor*, such that for every probabilistic* expected *polynomial-time machine P^* and every $x, y, r \in \{0,1\}^*$, machine K satisfies the following condition:*

  Denote by $p(x,y,r)$ the probability that V accepts, on input x, when interacting with the prover specified by $P^*_{x,y,r}$ (where $P^*_{x,y,r}$ denotes the strategy of P^* on common-input x, auxiliary-input y and random-tape r). Then, if $p(x,y,r) > \kappa(|x|)$ then, $(x, K^{P^*_{x,y,r}}(x)) \in R$, where $K^{P^*_{x,y,r}}$ runs within an expected number of steps bounded by

  $$\frac{q(|x|)}{p(x,y,r) - \kappa(|x|)}$$

  The oracle machine K is called a knowledge extractor.

An interactive pair (P, V) so that V is a knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a system of arguments of knowledge *for the relation R. The argument system (P, V) is a system of* zero-knowledge arguments of knowledge *if it is also zero-knowledge.*

A remark on alternative definitions of proofs of knowledge. The definition of a proof of knowledge used by Feige and Shamir in [17] is different from the above (specifically, it states that the extractor runs in expected polynomial-time and the probability of successful extraction is at most negligibly smaller than the probability that the verifier accepts). However, as shown in [7], these definitions are equivalent for $\mathcal{NP}$ relations.

Soundness. We note that the above validity requirement implies *soundness* with error of at most κ. That is, if there does not exist a y such that $(x, y) \in R$, then the probability that the verifier accepts is at most κ. This is because no witness can be extracted (as none exists), yet the validity requirement demands that extraction succeeds in the case that $p(x) > \kappa(|x|)$. As mentioned above, the system of arguments of Feige and Shamir [17] is such that the error function κ is negligible.

Formally, the soundness property of interactive proofs is stated as follows: there exists a *single* negligible function $\kappa(\cdot)$, such that for every probabilistic polynomial-time (malicious) prover P^*, the probability that V accepts after interacting with P^* upon input x and $x \notin L$, is at most $\kappa(|x|)$. However, the definition of soundness used in [17] states that for every prover P^* there exists a (possibly different) negligible function κ_{P^*}, such that the probability that the verifier accepts upon common input $x \notin L$ is at most $\kappa_{P^*}(|x|)$. The fact that these soundness requirements are equivalent was shown in [6].

On expected polynomial-time adversaries. The standard definition of zero-knowledge arguments refers to strict polynomial-time verifiers. Thus, a simulator is required to generate a transcript that is indistinguishable to a real transcript for such a verifier. On the other hand, our definition above refers to *expected* polynomial-time verifiers. This extension is needed since in our definition of secure computation, the real-model adversary in a two-party protocol is allowed to run in expected polynomial-time. In the protocol simulation, this adversary plays the verifier and we must therefore be able to simulate the zero-knowledge protocol when the verifier runs in expected (rather than strict) polynomial-time. This is not immediate. In fact, in Appendix A, we show that there exist expected polynomial-time verifiers for which the simulator for the protocol of [20] runs for an exponential number of steps. Thus, the zero-knowledge property of the [20] protocol seems *not* to hold for expected polynomial-time verifiers (we do not know how to "fix" the simulator of [20] so that it does succeed for such verifiers). A similar issue arises with respect to extraction from an expected polynomial-time prover.

In Appendix A, we show that the simulator (resp., extractor) of the zero-knowledge arguments of knowledge of [17] *does* run in expected polynomial-time for any expected polynomial-time verifier (resp., prover). Thus, we can use the [17] argument system in our protocols, where security must be guaranteed against expected polynomial-time adversaries. See Appendix A for a detailed treatment of this issue.

Perfect zero-knowledge arguments. A perfect zero-knowledge argument is one for which there exists a simulator that generates a transcript that is *identically distributed* to the transcript of a real proof (and not just computationally indistinguishable). Constant-round perfect zero-knowledge arguments and arguments of knowledge can be constructed using families of clawfree functions or any perfectly hiding string commitment [10, 17].

3.3 Proofs of Knowledge as Subprotocols: A Useful Lemma

Zero-knowledge proofs (or arguments) of knowledge are often used as subprotocols within larger protocols. The property desired by the use of a proof of knowledge in such a scenario is that a simulator can run the extractor for the proof of knowledge and thus obtain some secret information. This secret information is then used in order to simulate the rest of the larger protocol.

However, a technical problem arises when using a proof of knowledge in such a way. In order to illustrate this problem, consider a two-party protocol consisting of a number of stages. In one of the stages, party A provides a proof of knowledge of some witness w to party B. Then, in the proof of security, a simulator S needs to simulate A's view and as part of this simulation, it must extract the witness w. The natural way to construct such a simulator is to simply have it run the knowledge extractor at the appropriate point, thus obtaining w. However, S needs to simulate A's *entire* view and this includes A's view *within* the proof of knowledge. (Recall that by the definition of proofs of knowledge, the extractor only outputs a witness and not a view of the execution.[4]) Therefore, in order to obtain both A's view within the proof of knowledge *and* the witness, S works as follows.

First, S verifies the proof given by A, by playing the honest verifier. Then, if the proof is rejected, S halts (exactly as party B in a real interaction with A would). On the other hand, if S accepts the proof, it proceeds by running the extractor for the proof of knowledge in order to obtain w. In this way, S obtains both A's view in the proof (from the initial verification), and the witness (from the extraction). Now, let p denote the probability that A convinces the verifier in the proof of knowledge. Then, by the definition of proofs of knowledge (Definition 9), there exists an extractor K and a negligible function κ (this is the knowledge-error function) such that if $p > \kappa$, then K successfully obtains the witness w within an expected number of steps bounded by $poly/(p-\kappa)$. Since the simulator runs the extractor with probability p (i.e., only when the proof is accepted), we have that the overall expected running-time of the simulator is $poly \cdot p/(p-\kappa)$. This brings us to the problem: the function $p/(p-\kappa)$ may not be polynomial (for example, if $\kappa = 2^{-n/2}$ and $p = 2^{-n/2} + 2^{-n}$, then $p/(p-\kappa) \approx 2^{n/2}$).

This technical difficulty was solved by Goldreich and Kahan in [20] and therefore there is no inherent problem here. However, their techniques are complex and thus applying them every time

[4]This technicality is especially annoying since all known extraction procedures do indeed generate such a view. However, since it is not required by the definition, they do not output it.

we wish to use a proof of knowledge as a subprotocol is cumbersome. Our goal here is therefore to present a general lemma that will enable the use of proofs of knowledge as subprotocols *without* requiring any complicated analysis. Rather, the analysis (indeed using the techniques of [20]) is needed only once in proving the lemma (established below).

Witness-extended emulation. Loosely speaking, the lemma below states that if there exists a proof of knowledge with a knowledge extractor K (by Def. 9), then there exists an expected polynomial-time *witness-extended emulator*[5] E who outputs the information required by the simulator to continue the simulation of the larger protocol. That is, E outputs the following two items:

1. The honest verifier's view of an execution of the proof of knowledge with the specified prover. (Recall that a party's view contains the contents of its input, auxiliary input and random tape, along with the transcript of messages that it sees during the execution.)

2. The witness for the statement being proved in the proof of knowledge (in the case that the above verifier view is accepting).

We note that given such a lemma, simulation in the context of secure computation as described above is simple. The simulator S runs the witness-extended emulator E at the appropriate point and obtains the verifier's view v and (possibly) the secret witness w. Simulator S can then compute from v whether or not the proof was accepting: if not, then S halts. Otherwise, S has the witness w and can continue the simulation. In order to do this, it computes the transcript of messages t_{pok} that the prover would receive in this execution (such a transcript can easily be computed from v). Then, it feeds the prover with these messages and continues the simulation of the rest of the protocol, as required. We note that since E runs in expected polynomial-time, the overall running-time of S is also expected polynomial-time.

Notation: Recall that $P^*_{x,y,r}$ denotes the strategy of P^* upon common-input x, auxiliary-input y and random-tape r. We denote by $\mathsf{view}_V^{P^*_{x,y,r}}(x)$ a random variable describing the view of the honest verifier V in an execution of the proof of knowledge with $P^*_{x,y,r}$ (this random variable depends only on the coins of V). Furthermore, let $\mathsf{accept}_V(\cdot)$ be a (deterministic) function that takes a specific verifier view, and outputs 1 if and only if V accepts in the proof execution in which this is its view.

We are now ready to present the definition:

Definition 10 (witness-extended emulator): *Let R be a binary relation and let (P, V) be an interactive proof system. Consider a probabilistic expected polynomial-time machine E that is given input x and access to the oracle $P^*_{x,y,r}$. Let $E_1^{P^*_{x,y,r}}(x)$ and $E_2^{P^*_{x,y,r}}(x)$ denote the random variables representing the first and second elements of the output of E, respectively. We say that E is a* witness-extended emulator for (P, V) and R *if for every interactive function P^*, every $x, y, r \in \{0,1\}^*$, every polynomial $p(\cdot)$ and all sufficiently large x's,*

1. *E_1 outputs the distribution of V's view in a real execution with $P^*_{x,y,r}$. That is,*

$$\left\{E_1^{P^*_{x,y,r}}(x)\right\}_{x,y,r} \equiv \left\{\mathsf{view}_V^{P^*_{x,y,r}}(x)\right\}_{x,y,r}$$

[5] We chose to call this machine a "witness-extended emulator" due to the fact that the main goal is *emulation* and the witness extraction is a tool used to accomplish this goal.

2. *The probability that V's view (as output by E_1) is accepting, and yet E_2 does not output a correct witness, is negligible. That is,*

$$\Pr[\mathsf{accept}_V(E_1^{P^*_{x,y,r}}(x)) = 1 \ \& \ (x, E_2^{P^*_{x,y,r}}(x)) \notin R] < \frac{1}{p(|x|)}$$

As we have mentioned above, the aim of this section is to prove the existence of a witness-extended emulator for *any* proof of knowledge. That is,

Lemma 3.1 (witness-extended emulation lemma): *Let R be a binary relation and let (P, V) be a proof of knowledge for R with negligible knowledge error. Then, there exists a witness-extended emulator E for (P, V) and R.*

Proof: Let $P^*_{x,y,r}$ be an arbitrary prover strategy and let x be the common-input (i.e., P^* is proving that it knows w such that $(x, w) \in R$). We construct a witness-extended emulator E as follows: E works in a similar manner as described in the motivating discussion above. That is, E first verifies the proof from $P^*_{x,y,r}$ by playing the role of the honest verifier. Following this step, E can already define its first output. That is, let r_V be the coins used by E in the verification of the proof from $P^*_{x,y,r}$. Furthermore, let t equal the transcript of messages received by V in the proof execution between $P^*_{x,y,r}$ and V (who has random-tape r_V). Then, E defines V's view as $E_1 = (x, r_V, t)$. It is clear that the output of E_1 is distributed exactly like that of the honest verifier's view (when interacting with $P^*_{x,y,r}$). Therefore, at this point, item (1) of Definition 10 is fulfilled.

Now, if the above proof is rejected by E (i.e., if $\mathsf{accept}_V(E_1) = 0$), then E outputs the pair $(E_1, \perp)$ and halts. In this case, there is no requirement that E outputs a witness and this is therefore sufficient. Otherwise, if the proof is accepting (i.e., if $\mathsf{accept}_V(E_1) = 1$), then E proceeds to attempt extraction of the witness. However, as described above, the naive approach of simply running the extractor K for the proof of knowledge (P, V) may result in E not being expected polynomial-time.

This problem is solved by ensuring that the extractor never runs "too long". Specifically, as in [20], E first estimates the value of $p(x, y, r)$, where $p(x, y, r)$ denotes the probability that $P^*_{x,y,r}$ successfully proves that it knows a witness w such that $(x, w) \in R$. This is done by repeating the verification until a fixed polynomial number, denoted $q'(|x|)$, of successful verifications occur. By choosing $q'(\cdot)$ to be large enough, it is possible to obtain an estimate $\tilde{p}$ such that with overwhelming probability $\tilde{p}$ is within a constant factor of $p(x, y, r)$. We set q' so that the probability that $\tilde{p}$ is not within a constant factor of p is at most $2^{-|x|^2}$.

Next, E repeats the following up to $|x|$ times: E runs the extractor K and answers all of K's oracle queries with the oracle $P^*_{x,y,r}$. However, E limits the number of steps taken by K to $|x| \cdot q(|x|)/\tilde{p}$ steps, where $q(\cdot)$ is the polynomial from the *validity* condition in the definition of proofs of knowledge (every extractor K has a single polynomial $q(\cdot)$ associated with it and so E can use the appropriate q). Note that a call to the oracle $P^*_{x,y,r}$ is counted by E as a single step. Now, if within this time K outputs a witness w, then E outputs the pair (E_1, w) (where E_1 is the value generated above). (We note that E does not need to check if w is a valid witness because by the definition of K, it only outputs valid witnesses.) If K does not output a witness within this time, then E aborts this attempt and tries again. As mentioned above, this is repeated up to $|x|$ times; we stress that in each attempt, K is provided independent coins by E. If the extractor K did not output a witness in any of the $|x|$ attempts, then E halts and outputs the pair $(E_1, \mathsf{time\text{-}out})$. We will show that this strategy ensures that the probability that E outputs $\mathsf{time\text{-}out}$ is negligible.

Therefore, the probability that the initial verification of the proof succeeded, yet E does not output a valid witness, is negligible.

In addition to the above, E keeps a count of the overall running time of K and if it reaches $2^{|x|}$ steps, then it halts outputting the pair $(E_1, \textsf{time-out})$. (This additional time-out is needed to ensure that E does not run too long in the case that the estimate $\tilde{p}$ is not within a constant factor of $p(x, y, r)$. Recall that this "bad event" can only happen with probability $2^{-|x|^2}$.)

We first claim that E runs in expected polynomial-time.

Claim 3.2 *The emulator E runs in expected-time that is polynomial in $|x|$.*

Proof: Recall that E initially verifies the proof provided by $P^*_{x,y,r}$. Since E merely plays an honest verifier, this takes a strict polynomial number of steps. Next, E obtains an estimate $\tilde{p}$ of $p(x, y, r)$. This involves repeating the verification until $q'(|x|)$ successes are obtained. Therefore, the expected number of repetitions equals exactly $q'(|x|)/p(x, y, r)$ (since the expected number of trials for a single success is $1/p(x, y, r)$). After the estimation $\tilde{p}$ has been obtained, E runs the extractor K for a maximum of $|x|$ times, each time for at most $|x| \cdot q(|x|)/\tilde{p}$ steps.

Given the above, we are ready to compute the expected running-time of E. In order to do this, we differentiate between two cases. In the first case, we consider what happens if $\tilde{p}$ is *not* within a constant factor of $p(x, y, r)$. The only thing we can say about E's running-time in this case is that it is bound by $2^{|x|}$ (since this is an overall bound on its running-time). However, since this event happens with probability at most $2^{-|x|^2}$, this case adds only a negligible factor to the overall expected running-time. We now consider the second case, where $\tilde{p}$ *is* within a constant factor of $p(x, y, r)$. In this case, we can bound the expected running-time of E by

$$p(x, y, r) \cdot \left(\frac{q'(|x|)}{p(x, y, r)} + \frac{|x|^2 q(|x|)}{\tilde{p}} \right) = \text{poly}(|x|) \cdot \frac{p(x, y, r)}{\tilde{p}} = \text{poly}(|x|)$$

and this concludes the analysis. ■

As we have mentioned above, E's first output E_1 is distributed exactly as required by Definition 10. Therefore, it remains to show that the probability that P^*'s proof is accepting and yet E does not output a valid witness, is negligible. Notice that whenever P^*'s proof is accepting, E runs the extractor K and either obtains a proper witness w or it outputs $\textsf{time-out}$. That is, in the case of accepting proofs, if E does not output $\textsf{time-out}$, then it outputs a proper witness. Therefore, it suffices to show that the probability that E outputs $\textsf{time-out}$ is negligible.

Claim 3.3 *The probability that E outputs the* $\textsf{time-out}$ *symbol is a negligible function in $|x|$.*

Proof: Notice that the probability that E outputs $\textsf{time-out}$ is less than or equal to the probability that the extractor K does not succeed in outputting a witness w in any of the $|x|$ extraction attempts *plus* the probability that E runs for $2^{|x|}$ steps.

We first claim that the probability that E runs for $2^{|x|}$ steps is negligible. We have already shown in Claim 3.2, that E runs in expected polynomial-time. Therefore, the probability that an execution will deviate so far from its expectation and run for $2^{|x|}$ steps is negligible. (It is enough to use Markov's inequality to establish this fact.)

We now continue by considering the probability that in all $|x|$ extraction attempts, the extractor K does not output a witness within $|x|q(|x|)/\tilde{p}$ steps. Consider the following two possible cases (recall that $p(x, y, r)$ equals the probability that $P^*_{x,y,r}$ succeeds in proving the proof, and that κ is the negligible knowledge-error function of the proof system):

1. *Case 1:* $p(x,y,r) \le 2\kappa(|x|)$: In this case, P^* succeeds in proving the proof with only negligible probability. This means that the probability that E even reaches the stage that it runs K is negligible (and thus E outputs time-out with negligible probability only).

2. *Case 2:* $p(x,y,r) > 2\kappa(|x|)$: Recall that by the definition of proofs of knowledge, the polynomial $q(\cdot)$ is such that the expected number of steps taken by K to output a witness is at most $q(|x|)/(p(x,y,r) - \kappa(|x|))$. Now, since in this case $p(x,y,r) > 2\kappa(|x|)$, it holds that the expected number of steps required by K is less than $2q(|x|)/p(x,y,r)$. Assuming that $\tilde{p}$ is within a constant factor of $p(x,y,r)$, we have that the expected number of steps is bound by $O(q(|x|)/\tilde{p})$. Therefore, by Markov's inequality, the probability that K runs longer than $|x|q(|x|)/\tilde{p}$ steps in any given extraction attempt is at most $O(1/|x|)$. It follows that the probability that K runs longer than this time in $|x|$ independent attempts is negligible in $|x|$ (specifically, it is bound by $O(1/|x|)^{|x|}$). This covers the case that $\tilde{p}$ is within a constant factor of $p(x,y,r)$. However, the probability that $\tilde{p}$ is *not* within a constant factor of $p(x,y,r)$ is also negligible. Putting this together, we have that E outputs time-out with negligible probability only.

■

This completes the proof of the lemma. ■

3.3.1 On Expected Polynomial-Time Provers

We proved the witness-extended emulation lemma for E who is given oracle access to an arbitrary prover strategy P^*. However, in some contexts (like in secure computation), P^* runs in polynomial-time (or expected polynomial-time) and the running-time of P^* must be included in the analysis of the overall running-time of E. This is a non-issue in the case that P^* runs in strict polynomial-time, because we can simply multiply E's running-time by a single polynomial. However, in the case that P^* runs in expected polynomial-time, we must ensure that E remains expected polynomial-time. This extension must therefore be dealt with when considering expected polynomial-time adversaries (as is the case in this work). The only place that this arises in the proof of Lemma 3.1 is in Claim 3.2 (i.e., when analyzing the running-time of E).

We therefore reprove Claim 3.2 here for the case that P^* is an expected polynomial-time machine. Before proceeding, we note that in this context, it does not make sense to consider a prover $P^*_{x,y,r}$ with fixed randomness. This is because the expectation with respect to P^*'s running-time is taken over its random-tape r as well.[6] Thus, we modify E so that it first chooses a random-tape r for $P^*_{x,y}$. This then defines the prover $P^*_{x,y,r}$, and E continues as described above. We consider extractors who receive $P^*_{x,y,r}$ and whose expected running-time is polynomial, when the expectation is both over its own coins *and* the choice of r. Furthermore, we include the running-time of $P^*_{x,y,r}$ in the running-time of the extractor. We say that such an extractor "remains expected polynomial-time when including the running-time of an expected polynomial-time prover P^*". We note that in Appendix A, we show that such extractors exist (in particular, the extractor of the [17] zero-knowledge arguments of knowledge).

[6] We stress that for a fixed r, the machine $P^*_{x,y,r}$ may *not* be polynomial-time. For example, consider a prover which runs in strict polynomial-time except when its random-tape equals a single random string r, in which case it runs for $2^{|r|}$ steps. Clearly, $P^*_{x,y}$ runs in expected polynomial-time. However, the machine $P^*_{x,y,r}$ with fixed random-tape r, runs for an exponential number of steps.

Claim 3.4 (Claim 3.2 – restated): *If K remains expected polynomial-time when including the running-time of an expected polynomial-time prover P^*, then E also remains expected polynomial-time when including the running-time of an expected polynomial-time prover P^*.*

Proof: Recall that E first verifies the proof from $P^*_{x,y,r}$ and if it is accepting then repeats the verification until $q'(|x|)$ successes (i.e., accepting executions) are obtained. Following this, E runs the knowledge extractor K for exactly $|x|$ extraction attempts (while limiting its running-time). By the assumption that K remains expected polynomial-time in this setting, it is enough to analyze the running-time of the first stage of E. We stress that in each verification of the first stage of the emulation, the emulator E (playing the verifier) uses new and independent randomness, whereas $P^*_{x,y}$'s randomness is fixed to r. In Claim A.1, we show that the expected running-time for obtaining a *single* success in such a scenario, is $1/p(x,y,r)$ times the expected running-time of $P^*_{x,y,r}$ (where the expectation is taken over K's coins). Therefore, by the linearity of expectations, the expected running-time for obtaining $q'(|x|)$ successes equals $q'(|x|)/p(x,y,r)$ times the expected running-time of $P^*_{x,y,r}$. By continuing as in the analysis of Claim 3.2, we obtain that E concludes the first stage in an expected number of steps bounded by $\mathrm{poly}(|x|)$ times the expected running-time of $P^*_{x,y,r}$. We conclude by noticing that since $P^*_{x,y}$'s random-tape r is chosen uniformly, the overall expected running-time of E (where the expectation is also over the choice of r) is bounded by $\mathrm{poly}(|x|)$ times the expected running-time of $P^*_{x,y}$, as required. ■

4 Two-Party Computation Secure Against Malicious Adversaries

4.1 The Compiler of Goldreich, Micali and Wigderson [25]

Goldreich, Micali and Wigderson [25] showed that assuming the existence of trapdoor permutations, there are secure protocols (in the malicious model) for any multi-party functionality. Their methodology works by first presenting a protocol secure against semi-honest adversaries. Next, a *compiler* is applied that transforms *any* protocol secure against semi-honest adversaries into a protocol secure against malicious adversaries. Thus, their compiler can also be applied to the constant-round two-party protocol of Yao [33] (as it is secure against semi-honest adversaries). However, as we shall see, the output protocol has a polynomial number of rounds, and in particular is *not* constant-round. In this section, we describe the [25] compiler and show what should be modified in order to obtain a *constant-round* compiler instead.

Enforcing semi-honest behavior. The GMW compiler takes for input a protocol secure against semi-honest adversaries; from here on we refer to this as the "basic protocol". Recall that this protocol is secure in the case that each party follows the protocol specification exactly, using its input and uniformly chosen random tape. Thus, in order to obtain a protocol secure against malicious adversaries, we need to enforce potentially malicious parties to behave in a semi-honest manner. First and foremost, this involves forcing the parties to follow the prescribed protocol. However, this only makes sense relative to a *given* input and random tape. Furthermore, a malicious party must be forced into using a *uniformly chosen* random tape. This is because the security of the basic protocol may depend on the fact that the party has no freedom in setting its own randomness.[7]

[7] A clear example of this is the semi-honest 1-out-of-k Oblivious Transfer protocol of [14] (see Construction 2.2.5 in [18]). The oblivious transfer functionality is defined by $((b_1,\ldots,b_k),i) \mapsto (\lambda, b_i)$. In the [14] protocol, the receiver gives the sender k images of a trapdoor permutation, where the receiver knows only the i'th preimage. The protocol

An informal description of the GMW compiler. In light of the above discussion, the compiler begins by having each party commit to its input. Next, the parties run a coin-tossing protocol in order to fix their random tapes (clearly, this protocol must be secure against malicious adversaries). A regular coin-tossing protocol in which both parties receive the same uniformly distributed string is not sufficient here. This is because the parties' random tapes must remain secret. This is solved by augmenting the coin-tossing protocol so that one party receives a uniformly distributed string (to be used as its random tape) and the other party receives a commitment to that string. Now, following these two steps, each party holds its own uniformly distributed random-tape and a commitment to the other party's input and random-tape. Therefore, each party can be "forced" into working consistently with the committed input and random-tape.

We now describe how this behavior is enforced. A protocol specification is a deterministic function of a party's view consisting of its input, random tape and messages received so far. As we have seen, each party holds a commitment to the input and random tape of the other party. Furthermore, the messages sent so far are public. Therefore, the assertion that a new message is computed according to the protocol is of the $\mathcal{NP}$ type (and the party sending the message knows an adequate NP-witness to it). Thus, the parties can use zero-knowledge proofs to show that their steps are indeed according to the protocol specification. As the proofs used are zero-knowledge, they reveal nothing. On the other hand, due to the soundness of the proofs, even a malicious adversary cannot deviate from the protocol specification without being detected. We thus obtain a reduction of the security in the malicious case to the given security of the basic protocol against semi-honest adversaries.

In summary, the components of the compiler are as follows (from here on "secure" refers to security against malicious adversaries):

1. **Input Commitment:** In this phase the parties execute a secure protocol for the following functionality:
$$((x, r), 1^n) \mapsto (\lambda, C(x; r))$$
where x is the party's input string (and r is the randomness chosen by the committing party).

   A secure protocol for this functionality involves the committing party sending $C(x; r)$ to the other party followed by a zero-knowledge proof of knowledge of (x, r). Informally, this functionality ensures that the committing party "knows" the value being committed to.

2. **Coin Generation:** The parties generate t-bit long random tapes (and corresponding commitments) by executing a secure protocol in which one party receives a commitment to a uniform string of length t and the other party receives the string itself (to be used as its

works so that the receiver obtains b_j if it knows the j'th preimage (and otherwise he learns nothing of the value of b_j). Thus, were the receiver to know more than one of the k preimages, it would learn more than a single b_i, in contradiction to the security of the protocol. Now, if the receiver can "alter" its random tape, then it can influence the choice of the images of the permutation so that it knows more than one preimage. Thus, the fact that the receiver uses a truly random tape is crucial to the security.

We mention that the above weakness is not a peculiarity of the above specific protocol, and in fact, any protocol secure in the face of semi-honest adversaries can be modified so that it is still "semi-honest" secure, yet is completely breakable in the case that the adversary can fix its own randomness. In order to see this, consider the following addition to the beginning of some semi-honest protocol. One of the parties (say Party 1) randomly chooses an n-bit string. Then, if the string is all zeros, it asks Party 2 for its input (and Party 2 complies). Otherwise, the parties continue with the original protocol. Since a semi-honest adversary cannot fix its own randomness (and cannot request Party 2's input if the random string is not all zeros), the modified protocol is still secure. On the other hand, if Party 1 is adversarial *and* can fix its own randomness, then it can obtain Party 2's input, in contradiction to the security requirements.

random tape) and the decommitment (to be used later for proving "proper behavior"). That is, the parties compute the functionality:

$$(1^n, 1^n) \mapsto ((U_t, U_{t\cdot n}), C(U_t; U_{t\cdot n}))$$

(where we assume that to commit to a t-bit string, C requires $t \cdot n$ random bits).

3. **Protocol Emulation:** In this phase, the parties run the basic protocol whilst proving (in zero-knowledge) that their steps are consistent with their input string, random tape and prior messages received.

A detailed description of each phase of the compiler and a full proof that the resulting protocol is indeed secure against malicious adversaries can be found in [18].

4.2 Achieving Round-Preserving Compilation

As we have mentioned, our aim in this work is to show that the GMW compiler can be implemented so that the number of rounds in the resulting compiled protocol is within a *constant factor* of the number of rounds in the original semi-honest protocol. We begin by noting that using currently known constructions, Phases 1 and 3 of the GMW compiler can be implemented in a constant number of rounds. That is,

Proposition 4.1 *Assuming the existence of one-way functions, both the* input-commitment *and* protocol-emulation *phases can be securely implemented in a* constant number of rounds.

First consider the input-commitment phase. As mentioned above, this phase can be securely implemented by having the committing party send a perfectly binding commitment of its input to the other party, followed by a zero-knowledge proof of knowledge of the committed value. Both constant-round commitment schemes and constant-round zero-knowledge arguments of knowledge are known to exist by the works of Naor [28] and Feige and Shamir [17], respectively (these constructions can also be based on any one-way function). Thus the input-commitment phase can be implemented as required for Proposition 4.1.[8] Next, we recall that a secure implementation of the protocol emulation phase requires zero-knowledge proofs for $\mathcal{NP}$ only. Thus, once again, using the argument system of [17], this can be implemented in a constant number of rounds (using any one-way function).

Constant-round coin tossing. In contrast to the input-commitment and protocol-emulation phases of the GMW compiler, known protocols for tossing polynomially many coins do not run in a constant number of rounds. Rather, single coins are tossed sequentially (and thus $poly(n)$ rounds are needed). In particular, the proof of [18] does *not* extend to the case that many coins are tossed in parallel. Thus, in order to obtain a round-preserving compiler, it remains to present a secure protocol for the coin-generation functionality that works in a constant number of rounds. Furthermore, it is preferable to base this protocol on the existence of one-way functions only (so that this seemingly minimal assumption is all that is needed for the entire compiler). In the next section we present such a coin-tossing protocol, thereby obtaining Theorem 2 (as stated in the introduction).

[8] We note that the protocol for the commit-functionality, as described in [18], is for a single-bit only (and thus the compiler there runs this protocol sequentially for each bit of the input). However, the proof for the commit-functionality remains almost identical when the functionality is extended to commitments of poly(n)-bit strings (rather than for just a single-bit). Alternatively, using the witness-extended emulation lemma of Section 3.3, a simple proof of the security of this protocol can be obtained.

4.3 Constant-Round Secure Computation

Recall that by Yao [33], assuming the existence of trapdoor permutations, any two-party functionality can be securely computed in the *semi-honest* model in a constant-number of rounds. Thus, applying the constant-round compiler of Theorem 2 to Yao's protocol, we obtain a constant-round protocol that is secure in the *malicious* model, and prove Theorem 3. That is, assuming the existence of trapdoor permutations, any two-party functionality can be securely computed in the *malicious* model in a *constant-number of rounds.*

5 The Augmented Coin-Tossing Protocol

In this section we present our coin-tossing protocol, thus proving Theorem 1.

5.1 The Augmented Coin-Tossing Functionality

In a basic coin-tossing functionality, both parties receive identical uniformly distributed strings. That is, the functionality is defined as: $(1^n, 1^n) \mapsto (U_m, U_m)$ for some $m = \text{poly}(n)$. This basic coin-tossing is augmented as follows. Let F be any deterministic function. Then, define the augmented coin-tossing functionality by:

$$(1^n, 1^n) \mapsto (U_m, F(U_m))$$

That is, the first party indeed receives a uniformly distributed string. However, the second party receives F applied to that string (rather than the string itself). Setting F to the *identity function*, we obtain basic coin-tossing. However, recall that the coin-generation component of the GMW compiler requires the following functionality:

$$(1^n, 1^n) \mapsto ((U_t, U_{t\cdot n}), C(U_t; U_{t\cdot n}))$$

where C is a commitment scheme (and we assume that C requires n random bits for every bit committed to).[9] Then, this functionality can be realized with our augmentation by setting $m = t + t \cdot n$ and $F(U_m) = C(U_t; U_{t\cdot n})$. Thus, the second party receives a commitment to a uniformly distributed string of length t and the first party receives the string and its decommitment. Recall that in the compiler, the party uses the t-bit string as its random tape and the decommitment in order to prove in zero-knowledge that it is acting consistently with this random tape (and its input).

5.2 Motivating Discussion

In order to motivate our construction of a constant-round coin-tossing protocol, we consider the special case of basic coin-tossing (i.e., where F is the identity function). A natural attempt at a coin-tossing protocol follows:

Protocol 1 (Attempt at Basic Coin-Tossing):

1. *Party* 1 *chooses a random string* $s_1 \in_R \{0,1\}^m$ *and sends* $c = \text{Commit}(s_1) = C(s_1; r)$ *for a random* r*.*

[9]There are other ways of defining this functionality that suffice for the GMW compiler. However, we chose the exact functionality defined by Goldreich in [18] because we rely on his proof of security of the GMW protocol.

2. Party 2 chooses a random string $s_2 \in_R \{0,1\}^m$ and sends it to Party 1.

3. Party 1 decommits to s_1 sending the pair (s_1, r).

Party 1 always outputs $s \stackrel{\text{def}}{=} s_1 \oplus s_2$, whereas Party 2 outputs $s_1 \oplus s_2$ if Party 1's decommitment is correct and $\perp$ otherwise.

We note that when $m = 1$ (i.e., a single bit), the above protocol is the basic coin-tossing protocol of Blum [8] (a rigorous proof of the security of this protocol can be found in [18]). However, here we are interested in a parallelized version where the parties attempt to simultaneously generate an m-bit random string (for any $m = \text{poly}(n)$). Intuitively, due to the secrecy of the commitment scheme, the string s_2 chosen by (a possibly malicious) Party 2 cannot be dependent on the value of s_1. Thus if s_1 is chosen uniformly, the resulting string $s = s_1 \oplus s_2$ is close to uniform. On the other hand, consider the case that Party 1 may be malicious. Then, by the protocol, Party 1 is committed to s_1 before Party 2 sends s_2. Thus, if s_2 is chosen uniformly, the string $s = s_1 \oplus s_2$ is uniformly distributed. We note that due to the binding property of the commitment scheme, Party 1 cannot alter the initial string committed to. We conclude that neither party is able to bias the output string.

However, the infeasibility of either side to bias the resulting string is not enough to show that the protocol is secure. This is because the definition of secure computation requires that the protocol simulate an *ideal execution* in which a trusted third party chooses a random string s and gives it to both parties. Loosely speaking, this means that there exists a simulator that works in the ideal model and simulates an execution with a (possibly malicious) party such that the joint output distribution (in this ideal scenario) is indistinguishable from when the parties execute the real protocol.

Protocol 1 seems not to fulfill this more stringent requirement. That is, our problem in proving the security of Protocol 1 is with constructing the required simulator. The main problem that occurs is regarding the simulation of Party 2.

Simulating a malicious Party 2: The simulator receives a uniformly distributed string s from the trusted party and must generate an execution consistent with s. That is, the commitment $c = C(s_1)$ given by the simulator to Party 2 must be such that $s_1 \oplus s_2 = s$ (where s_2 is the string sent by Party 2 in Step 2 of the protocol). However, s_1 is chosen and fixed (via a perfectly binding commitment) *before* s_2 is chosen by Party 2. Since the commitment is perfectly binding, even an all-powerful simulator cannot "cheat" and decommit to a different value. This problem is compounded by the fact that Party 2 may choose s_2 based on the commitment received to s_1 (by say invoking a pseudorandom function on c). Therefore, rewinding Party 2 and setting s_1 to equal $s \oplus s_2$ will not help (as s_2 will change and thus once again $s_1 \oplus s_2$ will equal s with only negligible probability). We note that this problem does not arise in the single-bit case as there are only two possible values for s_2 and thus the simulator succeeds with probability 1/2 each time.

A problem relating to abort: The above problem arises even when the parties *never* abort. However, another problem in simulation arises due to the ability of the parties to abort. In particular, simulation of Party 1 in Protocol 1 is easy assuming that Party 1 never aborts. On the other hand, when Party 1's abort probability is unknown (and specifically when it is neither negligible nor noticeable), we do not know how to construct a simulator that does not skew the real probability of abort in the simulated execution. Once again, this problem is considerably easier in the single-bit case since Party 1's decision of whether or not to abort is based on only a single bit sent by Party 2 in Step 2 of the protocol.

We note that basic coin-tossing is a special case of the augmented coin-tossing functionality. Thus, the same problems (and possibly others) must be solved in order to obtain an augmented coin-tossing protocol. As we will show, our solutions for these problems are enough for the augmented case as well.

5.3 The Actual Protocol

Before presenting the protocol itself, we discuss how we solve the problems described in the above motivating discussion.

- *Party* 1 *is malicious:* As described, when Party 1 is malicious, the problem that arises is that of aborting. In particular, Party 1 may decide to abort depending on the string s_2 sent to it by Party 2. This causes a problem in ensuring that the probability of abort in the simulation is negligibly close to that in a real execution. This is solved by having Party 1 send a proof of knowledge of s_1 after sending the commitment. Then, the simulator can extract s_1 from the proof of knowledge and can send $s_2 = s_1 \oplus s$ (where s is the string chosen by the trusted party) without waiting for Party 1 to decommit in a later step.

- *Party* 2 *is malicious:* As described, the central problem here is that Party 1 must commit itself to s_1 before s_2 is known (yet $s_1 \oplus s_2$ must equal s). This cannot be solved by rewinding because Party 2 may choose s_2 based on the commitment to s_1 that it receives (and thus changing the commitment changes the value of s_2). We solve this problem by not having Party 1 decommit at all; rather, it sends $s = s_1 \oplus s_2$ (or $F(s_1 \oplus s_2)$ in the augmented case) and proves in zero-knowledge that the value sent is consistent with its commitment and s_2. Thus, the simulator (who can generate proofs to false statements of this type) is able to "cheat" and send s (or $F(s)$) irrespective of the *real* value committed to in Step 1.[10]

  This technique of not decommitting, but rather revealing the committed value and proving (in zero-knowledge) that this value is correct, is central to our simulation strategy. Specifically, it enables us to "decommit" to a value that is unknown at the time of the commitment. (As we have mentioned, in order for the simulation to succeed, Party 2 must be convinced that the commitment of Step 1 is to s_1, where $s_1 \oplus s_2 = s$. However, the correct value of s_1 is only known to the simulator after Step 2.)

We now present our constant-round protocol for the augmented secure coin-tossing functionality: $(1^n, 1^n) \mapsto (U_m, F(U_m))$, for $m = \text{poly}(n)$. For the sake of simplicity, our presentation uses a non-interactive commitment scheme (which is easily constructed given any 1–1 one-way function). However, the protocol can easily be modified so that an interactive commitment scheme is used instead (in particular, the two-round scheme of Naor [28], which is based on any one-way function).

Protocol 2 (Augmented Parallel Coin-Tossing):

1. *Party* 1 *chooses* $s_1 \in_R \{0,1\}^m$ *and sends* $c = C(s_1; r)$ *for a random* r *to Party* 2 (*using a perfectly binding commitment scheme*).

[10] In general, nothing can be said about a simulated proof of a false statement. However, in the specific case of statements regarding commitment values, proofs of false statements are indistinguishable from proofs of valid statements. This is due to the hiding property of the commitment scheme.

2. *Party* 1 *proves knowledge of* (s_1, r) *with a (constant round) zero-knowledge argument of knowledge with negligible error. If the proof fails, then Party* 2 *aborts with output* $\perp$.

3. *Party* 2 *chooses* $s_2 \in_R \{0,1\}^m$ *and sends* s_2 *to Party* 1.

4. *If until this point Party* 1 *received an invalid message from Party* 2*, then Party* 1 *aborts, outputting* $\perp$.

   Otherwise, Party 1 *sends* $y = F(s_1 \oplus s_2)$.

5. *Party* 1 *proves to Party* 2 *using a (constant round) zero-knowledge argument that there exists a pair* (s_1, r) *such that* $c = C(s_1; r)$ *and* $y = F(s_1 \oplus s_2)$ *(that is, Party* 1 *proves that* y *is consistent with* c *and* s_2*).*[11] *If the proof fails, then Party* 2 *aborts with output* $\perp$.

6. Output:

   - *Party* 1 *outputs* $s_1 \oplus s_2$ *(even if Party* 2 *fails to correctly complete the verification of the proof in Step 5).*[12]
   - *Party* 2 *outputs* y.

Round complexity: Using the constant-round zero-knowledge argument system of Feige and Shamir [17] and the constant-round commitment scheme of Naor [28], Protocol 2 requires a constant number of rounds only. We stress that the argument system of [17] is also an argument of knowledge.

Sufficient assumptions: All the components of Protocol 2 can be implemented using one-way functions. In particular the string commitment of Naor [28] can be used (this requires an additional pre-step in which Party 2 sends a random string to Party 1; however this step is of no consequence to the proof). Furthermore, the zero-knowledge argument of knowledge of [17] can be used in both Steps 2 and 5. Since both the [28] and [17] protocols only assume the existence of one-way functions, this is the only assumption required for the protocol.

Theorem 11 *Assuming the existence of one-way functions, Protocol* 2 *is a secure protocol for augmented parallel coin-tossing.*

Proof: We need to show how to efficiently transform any admissible pair of machines (A_1, A_2) for the real model into an admissible pair of machines (B_1, B_2) for the ideal model. We denote the trusted third party by T, the coin-tossing functionality by f and Protocol 2 by Π. We first consider the case that A_1 is adversarial.

[11] It may appear that the reason that Party 1 does not decommit to c is due to the fact that Party 2 should only learn $F(s)$, and not s itself (if Party 1 decommits, then s is clearly revealed). Following this line of thinking, if F was the identity function, then Steps 4 and 5 could be replaced by Party 1 sending the actual decommitment. However, we stress that we do not know how to prove the security of such a modified protocol. The fact that Party 1 does not decommit, even when F is the identity function, is crucial to our proof of security.

[12] Recall that the definition of secure computation enables either party to abort before any output is received. However, only Party 1 is able to abort once it receives its output. We therefore prevent Party 2 from aborting after Step 4 (where it receives its output) by instructing Party 1 to output $s_1 \oplus s_2$ irrespective of Party 2's actions from this point on.

Lemma 5.1 *Let (A_1, A_2) be a pair of probabilistic expected polynomial-time machines for the real model in which A_2 is* honest. *Then there exists a pair of probabilistic expected polynomial-time machines (B_1, B_2) for the ideal model in which B_2 is honest, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

Proof: In this case the second party is honest and thus B_2 is determined. We now transform the real-model adversary A_1 into an ideal-model adversary B_1, where the transformation is such that B_1 uses *black-box* access to A_1. Specifically, B_1 chooses a uniform random tape, denoted R, for A_1 and invokes A_1 on input 1^n and random tape R. Once the input and random tape are fixed, A_1 is a deterministic function of messages received during a protocol execution. Thus $A_1(1^n, R, \overline{m})$ denotes the message sent by A_1 with input 1^n, random-tape R and sequence $\overline{m}$ of incoming messages to A_1.

The transformation works by having B_1 emulate an execution of A_1, while playing A_2's role. Machine B_1 does this when interacting with the trusted third party T and its aim is to obtain an execution with A_1 that is consistent with the output received from T. Therefore, B_1 has both external communication with T and "internal" emulated communication with A_1. Machine B_1 works as follows:

1. The ideal adversary B_1 chooses a uniformly distributed random tape R for the real adversary A_1, invokes the function $A_1(1^n, R)$, and obtains c (where c is supposed to equal $C(s_1; r)$ for some $s_1 \in \{0,1\}^m$, and C is the perfectly binding commitment scheme specified in the protocol).

2. B_1 runs the witness-extended emulator E guaranteed by Lemma 3.1 for the proof of knowledge of Step 2, with the prover function determined by $A_1(1^n, R)$. See Section 3.3 for the definition of a witness-extended emulator and its proof of existence for every proof of knowledge. Loosely speaking, such an emulator receives for input the prover strategy and statement being proved, and outputs (in expected polynomial-time) the verifier's view of a proof system execution along with a witness in the case that the proof is "accepting". That is, the emulator E outputs a pair (v, w), where v denotes the verifier's view of a proof given by A_1 to an honest verifier, and w is (possibly) a witness for the statement being proved (in this case, the witness is the decommitment of c).

   B_1 derives from v the series of incoming messages to A_1 in the proof, denote this series by t_{pok}. Then, B_1 checks whether or not $\mathsf{accept}_V(v) = 1$ (where $\mathsf{accept}_V(v) = 1$ if and only if the verifier would accept the proof associated with the view v).

   (a) If $\mathsf{accept}_V(v) = 0$, then B_1 aborts and outputs $A_1(1^n, R, t_{pok})$.

   (b) If $\mathsf{accept}_V(v) = 1$, then B_1 checks that w is a correct witness; that is, B_1 checks that $w = (s_1, r)$ and $c = C(s_1; r)$. If this is not the case, then B_1 outputs a special $\mathsf{failure}$ symbol.

3. Otherwise (if the proof from A_1 is accepting and w is a proper decommitment to c), B_1 sends 1^n to the (external) trusted third party T and receives the output s. (Note that B_2 does not as of yet receive $F(s)$.)

4. B_1 sets $s_2 = s \oplus s_1$, passes s_2 to A_1, and receives some string y from A_1 (y "should" equal $F(s)$ by the protocol, but this may not be the case). Formally, B_1 obtains y by computing the function $A_1(1^n, R, t_{pok}, s_2)$.

5. B_1 (internally) interacts with $A_1(1^n, R, t_{pok}, s_2)$ in the (zero-knowledge) argument of Step 5 of the protocol, in which A_1 plays the prover and B_1 plays the part of the verifier. (Denote the incoming messages to A_1 from the zero-knowledge argument by t_{pf}.)

   (a) If the verification fails, then B_1 instructs T to send $\perp$ (abort) to B_2.

   (b) If the verification succeeds, then B_1 instructs T to send $F(s)$ to B_2.

6. B_1 outputs $A_1(1^n, R, t_{pok}, s_2, t_{pf})$.

We need to show that

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

Clearly, if A_1 follows the instructions of Protocol 2, then the output distributions in the ideal and real models are identical. This is because A_1's view in the ideal-model emulation with B_1 is identical to that of a real execution with A_2. Furthermore, A_2 would output $F(s_1 \oplus s_2)$ in such a real execution and this equals $F(s)$, exactly as output by B_2 in the ideal model. However, A_1 may not follow the instructions of the protocol and we must show that nevertheless, the real and ideal output distributions are computationally indistinguishable (in fact, we will show that they are *statistically close*).

Loosely speaking, differences between the ideal and real distributions can occur either if B_1 outputs failure (which occurs when $\mathsf{accept}_V(v) = 1$ and w is not a proper decommitment), or if $y \neq F(s)$ and yet the verification of the second proof succeeds.

In the proof, we separate A_1's actions into two distinct stages. In the first stage A_1 sends c and proves knowledge of the decommitment. In the second stage, A_1 sends y and proves that it is the "correct" value (i.e., it is consistent with c and s_2). The proof proceeds by analyzing all possible scenarios for these stages.

Stage 1 – the commitment and proof of knowledge: By the witness-extended emulation lemma (Lemma 3.1), the verifier view output by E is identically distributed to A_2's view of a real execution of this proof provided by A_1 (and verified by A_2). Therefore, the series of messages t_{pok} received by A_1 from B_1 is identically distributed to the messages it would receive from A_2 in a real execution. This means that the view (in Stage 1) of A_1 in a real execution is identical to its view in the emulation from B_1. Thus, this stage can contribute a difference between the distributions only in the case that B_1 outputs $\mathsf{failure}$. However, by Lemma 3.1, the probability that $\mathsf{accept}_V(v) = 1$ and yet $w \neq (s_1, r)$ where $c = C(s_1; r)$, is negligible. Thus, the probability that B_1 outputs $\mathsf{failure}$ is negligible and the real and ideal executions (until this point) are statistically close.

By the above analysis, we have that if B_1 reaches Step 3 of its specification, then B_1 has obtained the (unique) pair (s_1, r) such that $c = C(s_1; r)$. Recall that in Step 4, B_1 passes $s_2 = s \oplus s_1$ to A_1, where s is the output received from T. We continue by analyzing the emulation of the second stage.

Stage 2 – y and the proof of consistency: First consider the case that A_1 sends $y = F(s)$ in Step 4 of the emulation (i.e., A_1 sends the "correct" value). There are two possible sub-cases here – either B_1 accepts the proof from A_1 in Step 5 or it rejects the proof. If the proof is accepted then B_2

outputs $F(s)$ (as B_1 instructs T to give $F(s)$ to B_2), otherwise B_2 outputs $\perp$ (again, as instructed by B_1). In a real execution, A_2 would also output $y = F(s)$ in the case that the proof is accepted and $\perp$ otherwise. The fact that the resulting distributions are identical is derived from the fact that the probability that B_1 accepts the (internally generated) proof from A_1 in the emulation, equals the probability that A_2 accepts the proof in a real execution (with A_1).

Finally, we consider the case that A_1 sends $y \neq F(s)$ in Step 4 of the emulation (i.e., A_1 attempts to "cheat"). If the verification of the proof in Step 5 fails, then both A_2 and B_2 output $\perp$ (as in the previous case). On the other hand, if the verification succeeds then the real-party A_2 outputs $y \neq F(s_1 \oplus s_2)$, whereas the ideal-party B_2 outputs $F(s_1 \oplus s_2)$ (and thus the real and ideal output distributions differ). However, by the soundness property of the proof system, a verifier can be convinced of a false assertion with at most negligible probability.

We thus conclude that unless the extraction fails or A_1 successfully "cheats" in the last proof, the output distributions are identical. Since these events can occur with at most negligible probability, we have that the distributions are statistically close. That is,

$$\left\{\mathrm{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \overset{\mathrm{s}}{\equiv} \left\{\mathrm{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

We conclude by noting that B_1 runs in expected polynomial-time (the "expected" part coming from the execution of the witness-extended emulator E that runs in expected polynomial-time). ■

We now consider the case that A_2 is adversarial.

Lemma 5.2 *Let (A_1, A_2) be a pair of probabilistic expected polynomial-time machines for the real model in which A_1 is* honest. *Then there exists a pair of probabilistic expected polynomial-time machines (B_1, B_2) for the ideal model in which B_1 is honest, such that*

$$\left\{\mathrm{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \overset{\mathrm{c}}{\equiv} \left\{\mathrm{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

Proof: In this case the first party is honest and thus B_1 is determined. We now transform the real-model adversary A_2 into an ideal-model adversary B_2. As before, B_2 works by using black-box access to A_2. The notation used here is the same as in the previous lemma. Machine B_2 works as follows:

1. The ideal adversary B_2 chooses a uniformly distributed random tape R for the real adversary A_2, invokes $A_2(1^n, R)$ and (internally) passes to A_2 the commitment $c = C(0^m; r)$ for a random r (recall that in a real execution, A_2 expects to receive $C(s_1; r)$ for $s_1 \in_R \{0,1\}^m$).

2. B_2 invokes the *simulator* for the zero-knowledge argument of knowledge of the decommitment of c, using $A_2(1^n, R, c)$ as the verifier.[13] (That is, this is a simulation of the proof of knowledge that A_1 is supposed to prove to A_2 in a real execution.)

3. B_2 obtains s_2 from A_2. (Recall that this is formally stated by having B_2 compute the function $A_2(1^n, R, c, t_{pok})$, where t_{pok} is the resulting transcript from the simulation of the zero-knowledge proof of knowledge in the previous step).

   If at any point until here A_2 sent an invalid message, then B_2 aborts and outputs $A_2(1^n, R, c, t_{pok})$.

[13] There is a significant difference between the transformation (of A_2) here and the transformation (of A_1) described in Lemma 5.1. There, B_1 *emulated* a true execution of the zero-knowledge proofs between A_1 and A_2 by playing A_2's role as an honest verifier. However, here B_2 runs the zero-knowledge *simulator* and does not attempt to really prove the statement.

4. The ideal adversary B_2 sends 1^n to the (external) trusted third party T and receives the output $F(s)$. (Note that this means that B_1 also receives its output s from T at this point.)

   Next, B_2 (internally) passes to A_2 the string $y = F(s)$.

5. B_2 invokes the simulator for the zero-knowledge proof of Step 5 of the Protocol with the verifier role being played by $A_2(1^n, R, c, t_{pok}, y)$. Denote the transcript from the simulation of the zero-knowledge proof by t_{pf}.

   (Recall that the statement being proved is that there exists a pair (s_1, r) such that $c = C(s_1; r)$ and $y = F(s_1 \oplus s_2)$, where s_2 is the string obtained from A_2 in Step 3. Note that with overwhelming probability this statement is false; nevertheless as we shall see the simulation generated is still indistinguishable from a real proof.)

6. B_2 outputs $A_2(1^n, R, c, t_{pok}, y, t_{pf})$.

We need to show that

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \tag{1}$$

The following differences are evident between the ideal and real executions:

- The commitment received by A_2 (in the internal emulation by B_2) is to 0^m, rather than to a random string consistent with $y = F(s)$ and s_2 (as is the case in a real execution). However, by the indistinguishability of commitments, this should not make a difference.
- In the internal emulation by B_2, the zero-knowledge proofs are simulated and not real proofs. However, by the indistinguishability of simulated proofs, this should also not make a difference. As mentioned above, this holds even though one of the statements is false with overwhelming probability; details follow in the proof.

The natural way to proceed at this point would be to define a hybrid experiment in which the commitment given by B_2 to A_2 is to s_1 and yet the zero-knowledge proofs are simulated. (In this hybrid experiment, s_1 must be such that $y = F(s_1 \oplus s_2)$.) However, such a hybrid experiment is problematic because the value of s_1 that is consistent with both y (from T) and s_2 is *unknown* at the point that B_2 generates the commitment. We must therefore bypass this problem before defining the hybrid experiment. We do this by defining the following mental experiment with a modified party B_2' (replacing Step 4 only of B_2 above):

4$'$. B_2' chooses $s_1 \in_R \{0,1\}^m$ (independently of what it has previously seen) and computes $y = F(s_1 \oplus s_2)$ (rather than obtaining $y = F(s)$ from T).

   Next, B_2' (internally) passes A_2 the string y.

We also modify the output of the honest party so that the output of (a virtual party) B_1' is defined as follows: if the execution of B_2' halts at Step 3 (i.e., if A_2 sent an invalid message), then $\text{output}(B_1') = \perp$, otherwise $\text{output}(B_1') = s_1 \oplus s_2$ (rather than the value s handed by T to B_1).

Notice that B_2' *does not* interact with any trusted third party at all. Rather, it chooses a uniformly distributed s, and computes $F(s)$ itself (choosing s_1 uniformly and setting $s = s_1 \oplus s_2$ is equivalent to uniformly choosing s). We stress that B_2' does not work in the ideal model, but is rather a mental experiment. Despite this, we will show that (in some sense) B_2' simulates the ideal model perfectly. To this end, define a mental experiment MENTAL with parties B_1' and B_2' by

$$\text{MENTAL}_{\overline{B'}}(1^n, 1^n) \stackrel{\text{def}}{=} (\text{output}(B_1'), \text{output}(B_2'))$$

We first claim that for the above $\overline{B}$ and $\overline{B}'$, we have that

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n,1^n)\right\}_{n\in\mathsf{N}} \equiv \left\{\text{MENTAL}_{\overline{B}'}(1^n,1^n)\right\}_{n\in\mathsf{N}} \tag{2}$$

This can be seen as follows. Since B_2' chooses s_1 uniformly and independently of s_2, we have that $s = s_1 \oplus s_2$ is uniformly distributed. Therefore, this is exactly the same as when the trusted third party T uniformly chooses s and gives it to B_2. This means that the outputs of B_2 and B_2' are identically distributed. Now, in a non-aborting execution, the outputs of B_1 and B_1' are defined to be s and $s_1 \oplus s_2$, respectively. Since $s = s_1 \oplus s_2$, and these are the same strings viewed by B_2 and B_2', we have that in such a case, the joint distributions of $\{B_1, B_2\}$ and $\{B_1', B_2'\}$ are identical. On the other hand, in aborting executions, B_1 and B_1' both output $\perp$. Therefore, this case does not change the above joint distributions. We conclude that $\{B_1, B_2\}$ and $\{B_1', B_2'\}$ are identically distributed.

In essence, the above states that the mental experiment is exactly the same as the ideal model (with B_2). However, as we have mentioned, this transition to the mental experiment is needed before defining an appropriate hybrid experiment.

By the above, it is enough to show that

$$\left\{\text{MENTAL}_{\overline{B}'}(1^n,1^n)\right\}_{n\in\mathsf{N}} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(1^n,1^n)\right\}_{n\in\mathsf{N}} \tag{3}$$

We begin by defining a hybrid setting (with a party B_2'') in which the initial commitment is to s_1 (rather than to 0^m as in the mental experiment), yet the zero-knowledge proofs are simulated (rather than being actual proofs as in the real model). First, define B_2'' as follows:

0. B_2'' chooses $s_1 \in_R \{0,1\}^m$.

1. B_2'' invokes $A_2(1^n, R)$ for a uniformly chosen R and (internally) passes A_2 the commitment $c = C(s_1; r)$ for a random r. (Recall that in contrast, in Step 1 of B_2', the commitment was to 0^m.)

2. B_2'' invokes the simulator for the zero-knowledge argument of knowledge of the decommitment of c, using $A_2(1^n, R, c)$ as the verifier.

   This is like Step 2 of B_2 (and B_2') except that here the commitment has a different distribution.

3. B_2'' obtains s_2 from A_2. (This, as well as the rest of this step, is like in B_2.)

4. B_2'' computes $y = F(s_1 \oplus s_2)$.

   Next, B_2'' (internally) passes A_2 the string y.

   (This is like Step $4'$ in the mental experiment, except that the first part of the step involving choosing s_1 has been moved to Step 0.)

The remaining Steps (5 and 6) are exactly the same as in the specification of B_2 and B_2'. Finally, the output of (a virtual party) B_1'' is defined as for B_1'. Now define:

$$\text{HYBRID}_{\overline{B}''}(1^n,1^n) \stackrel{\text{def}}{=} (\text{output}(B_1''), \text{output}(B_2''))$$

Note that in HYBRID, B_2'' chooses s_1 at the outset rather than after receiving s_2. However, this makes no difference to the distribution of s_1 because in the mental experiment B_2' chooses s_1 independently of any messages seen. Thus, the only difference between the MENTAL and HYBRID settings is with respect to the initial commitment (which in HYBRID is to s_1 and in MENTAL is to 0^m). On the other hand, the only difference between the HYBRID and REAL settings is with respect to the zero-knowledge proofs that are simulated in HYBRID rather than being actual proofs as in REAL. Below we shall show that both differences are indistinguishable and thus Eq. (3) follows.

We first show that

$$\left\{\text{MENTAL}_{\overline{B}'}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{HYBRID}_{\overline{B}''}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \tag{4}$$

As we have mentioned, the only difference between these two settings is with respect to the initial commitment (i.e., the commitment given to B_2'' is either to 0^m or to a uniformly chosen string s_1). Any distinguishable difference in the above two distributions can thus be used to distinguish such commitments. In particular, let D' be a probabilistic polynomial-time distinguisher that attempts to distinguish between the MENTAL and HYBRID experiments. We now construct a probabilistic polynomial-time machine D that uses D' in order to "break" the commitment scheme, specifically by distinguishing a commitment to 0^m from a commitment to $s_1 \in_R \{0,1\}^m$. Given c (that is either a commitment to 0^m or s_1), D internally invokes $A_2(1^n, R)$ and gives A_2 the commitment c. Next, D continues by simulating the rest of the MENTAL experiment, which is identical to the rest of HYBRID (i.e., D follows the specification of B_2' or equivalently B_2'' for Steps 2 to 6 of these experiments). The key point is that D need not know the value committed to in c in order to carry out this simulation. That is, D runs the zero-knowledge simulators and computes $y = F(s_1 \oplus s_2)$ exactly as defined for B_2' and B_2''. Then, upon concluding the simulation, D generates the pair $(\text{output}(B_1'), A_2(1^n, R, c, t_{pok}, y, t_{pf}))$ and outputs $D'(\text{output}(B_1'), A_2(1^n, R, c, t_{pok}, y, t_{pf}))$.

Now, if c is a commitment to 0^m, then the pair generated by D is distributed *exactly* according to $\text{MENTAL}_{\overline{B}'}(1^n, 1^n)$. On the other hand, if c is a commitment to s_1, then the pair is distributed *exactly* according to $\text{HYBRID}_{\overline{B}''}(1^n, 1^n)$. Thus, D can distinguish commitments to 0^m from commitments to s_1 with the same success as D' can distinguish the MENTAL and HYBRID distributions. By the indistinguishability of commitments, Eq. (4) follows.[14]

We now prove that

$$\left\{\text{HYBRID}_{\overline{B}''}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\Pi,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \tag{5}$$

The only difference between the two settings is with respect to the zero-knowledge proofs (in HYBRID the proofs are simulated and in REAL they are actual proofs). The fact that the experiments are indistinguishable is due to the formulation of zero-knowledge with respect to auxiliary inputs. In particular, it follows that for every s_1 and r, the transcript generated by the simulator for the proof of knowledge is indistinguishable from a real proof. Furthermore, for every s_1, r, t_{pok} and s_2 (where t_{pok} represents a transcript from the proof of knowledge), the transcript for the second proof is indistinguishable from a real proof. We therefore have that the output of B_2'' (in the hybrid model) is indistinguishable from the output of A_2 (in the real model).[15]

On the other hand, the probability that the (hybrid model) party B_1'' outputs $\perp$ is negligibly close to the probability that the real party A_1, in a real execution with A_2, aborts and outputs $\perp$. (This is because B_1'' and A_1 output $\perp$ only if A_2 sends an invalid message before Step 4. Since the only difference with respect to A_2's view in the settings is whether the proof is simulated or real, the probability that it sends an invalid message in both cases must be negligibly close.) Furthermore, in non-aborting executions, both B_1'' and A_1 output $s_1 \oplus s_2$. Thus, Eq. (5) holds.

[14] We note that this proof shows that the simulator generates an indistinguishable proof even for a false statement regarding the value committed to in c. Otherwise, the simulator could be used to distinguish commitments.

[15] Formally, one should consider a hybrid argument in which the first proof is real and the second is simulated. Then, by considering the indistinguishability of each simulated proof separately, the overall claim is obtained.

Combining Equations (4) and (5) we obtain Eq. (3). Finally, Eq. (1) follows from Equations (2) and (3), completing the proof of Lemma 5.2. ■

This completes the proof of Theorem 11. ■

5.4 Comparing Protocol 2 to the Protocol of [18]

The protocol for augmented coin-tossing presented by Goldreich [18] is for tossing a single bit only (i.e., where $m = 1$). Thus, in order to toss polynomially many coins, Goldreich suggests running the single-bit protocol many times sequentially. However, the only difference between Protocol 2 and the protocol of [18] is that here m can be any value polynomial in n and there m is fixed at 1 (i.e., by setting $m = 1$ in our protocol, we obtain the exact protocol of [18]). Despite this, our *proof* is different and works for any $m = \mathrm{poly}(n)$ whereas the *proof* of [18] relies heavily on $m = 1$ (or at the most $m = O(\log n)$).[16] Furthermore, there is a conceptual difference in the role of the two zero-knowledge proofs in the protocol. In [18], these proofs are used in order to obtain *augmented* coin-tossing (and are not needed for the case that F is the identity function). However, here these proofs are used for obtaining coin-tossing of $m = \mathrm{poly}(n)$ coins in parallel, even when F is the identity function.

6 Almost Perfect Coin-Tossing

In this section we present a constant-round protocol for *almost perfect* coin tossing. In such a protocol, the output distribution of a real execution is guaranteed to be *statistically close* to the output distribution of the ideal process (rather than the distributions being only computationally indistinguishable as required by secure computation); see Theorem 12. As in the previous section, the functionality we consider is that of augmented coin tossing:

$$(1^n, 1^n) \mapsto (U_m, F(U_m))$$

The protocol is almost identical to Protocol 2 except that the commitment scheme used is perfectly hiding and the zero-knowledge arguments are perfect.[17] These primitives are known to exist assuming the existence of families of clawfree functions or collision-resistant hash functions. Thus we rely here on a (seemingly) stronger assumption than merely the existence of one-way functions. We note that Protocol 1 is a protocol for the *almost perfect coin tossing* of a single bit and thus almost perfect coin tossing of m coins can be achieved in $O(m)$ rounds (see the proof in [18] which actually demonstrates statistical closeness). In this section we show that the almost perfect coin tossing of polynomially many coins can also be achieved in a *constant number of rounds*.

Protocol 3 (Augmented Almost Perfect Coin-Tossing):
An augmented almost perfect coin-tossing protocol is constructed by taking Protocol 2 and making the following modifications:

- *The commitment sent by Party* 1 *in Step* 1 *is* perfectly hiding.
- *The argument of knowledge provided by Party* 1 *in Step* 2 *is* perfect *zero-knowledge.*
- *The proof provided by Party* 1 *in Step* 5 *is a* perfect *zero-knowledge* argument of knowledge. (*Recall that in Protocol* 2*, this proof need not be a proof of knowledge.*)

[16] In private communication, Goldreich stated that he did not know whether or not his protocol [18] can be parallelized.

[17] We note that it actually suffices to use statistical zero-knowledge and statistically hiding commitments.

Constant-round perfect zero-knowledge arguments of knowledge are known to exist assuming the existence of constant-round perfectly hiding commitment schemes [10, 17]. Furthermore, constant-round perfectly-hiding commitment schemes can be constructed using families of clawfree [19] or collision-resistant hash functions [32, 13]. These commitment schemes work by having the receiver first uniformly choose a function f from the family designated in the protocol. The receiver then sends f to the sender who uses it to commit to a string by sending a single message. Thus, using such a scheme, Protocol 3 begins by Party 2 choosing a function f from a clawfree or collision-resistant family and sending it to Party 1. Then, Party 1 commits using f.

We stress the use of arguments of knowledge for *both* proofs here, whereas in Protocol 2 the proof of Step 5 need not be a proof of knowledge. The reason for this is that since the commitment is perfectly hiding, c is essentially a valid commitment to *every* $s_1 \in \{0,1\}^m$. Thus, every y is "consistent" with c and s_2. Therefore, what we need to ensure is that y is consistent with s_2 and the decommitment of c that are *known* to Party 1. This can be accomplished using a proof of knowledge.

Theorem 12 *Assuming the existence of perfectly-hiding commitment schemes, Protocol* 3 *is a secure protocol for augmented* almost perfect *coin-tossing. That is, for every admissible pair of probabilistic expected polynomial-time machines for the real model* (A_1, A_2) *there exists an admissible pair of probabilistic expected polynomial-time machines for the ideal model* (B_1, B_2)*, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\} \stackrel{\text{s}}{\equiv} \left\{\text{REAL}_{\Pi_2,\overline{A}}(1^n, 1^n)\right\}$$

where f is the augmented coin-tossing functionality and Π_2 *denotes Protocol* 3*.*

Proof Sketch: As in the proof of Theorem 11, we separately consider the cases that A_1 and A_2 are adversarial. We first consider the case that A_1 is adversarial.

Lemma 6.1 *Let* (A_1, A_2) *be a pair of probabilistic expected polynomial-time machines for the real model in which A_2 is* honest*. Then there exists a pair of probabilistic expected polynomial-time machines* (B_1, B_2) *for the ideal model in which B_2 is honest, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{s}}{\equiv} \left\{\text{REAL}_{\Pi_2,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

Proof Sketch: Loosely speaking, the only difference between here and the proof of Lemma 5.1 is that the commitment is only computationally binding. Furthermore, recall that in the proof of Lemma 5.1 we actually showed that the ideal and real distributions are statistically close, as required here. Thus, it is enough to show that this remains the case even though the commitment is only computationally binding. In the proof of Lemma 5.1, the fact that the commitment is perfectly-binding is used to show that if A_1 successfully proves both (zero-knowledge) proofs, then $y = F(s_1 \oplus s_2)$, where s_1 is the value A_1 committed to in the first step. Thus, the analogous argument here is that if A_1 successfully proves both (zero-knowledge) proofs of knowledge, then $y = F(s_1 \oplus s_2)$, where s_1 is the value A_1 *used* for the commitment in the first step. This is proven by showing that otherwise, the extractor for the proofs of knowledge can be used to extract two different decommitments from A_1. This would contradict the computational binding of the commitment scheme. ■

Lemma 6.2 *Let (A_1, A_2) be a pair of probabilistic expected polynomial-time machines for the real model in which A_1 is* honest*. Then there exists a pair of probabilistic expected polynomial-time machines (B_1, B_2) for the ideal model in which B_1 is honest, such that*

$$\left\{\text{IDEAL}_{f,\overline{B}}(1^n, 1^n)\right\}_{n\in\mathsf{N}} \stackrel{\text{s}}{\equiv} \left\{\text{REAL}_{\Pi_2,\overline{A}}(1^n, 1^n)\right\}_{n\in\mathsf{N}}$$

Proof Sketch: The proof of this lemma is very similar to the proof of Lemma 5.2. That is, we define analogous mental and hybrid experiments. Recall that the mental experiment is identically distributed to the ideal execution. Furthermore, the only difference between the mental and hybrid experiments is with respect to the value of the commitment. Since the commitments here are perfectly hiding, the experiments are identically distributed. Likewise, the only difference between the hybrid experiment and the real setting is that the hybrid setting uses a *simulated* proof rather than a real one. Then, since we use perfect zero-knowledge here, these distributions are also identically distributed. We therefore have that the ideal and real distributions are identical (and not even just statistically close). ■

This completes the proof of Theorem 12. ■

7 Acknowledgements

We would like to thank Oded Goldreich for his invaluable contribution to all aspects of this work. We would also like to thank Moni Naor for his suggestion that we look at the question of almost perfect coin-tossing as well, and Jonathan Katz for pointing out an error in the proof of Lemma 3.1 that appeared in earlier versions of this paper. Finally, we are grateful to the anonymous referees for their many helpful comments.

References

[1] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[2] B. Barak and Y. Lindell. Strict Polynomial-time in Simulation and Extraction. In *34th STOC*, pages 484–493, 2002.

[3] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[4] D. Beaver and S. Goldwasser. Multiparty Computation with Fault Majority. In *CRYPTO'89*, Springer-Verlag (LNCS 435), 1989.

[5] D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In *22nd STOC*, pages 503–513, 1990.

[6] M. Bellare. A Note on Negligible Functions. *Journal of Cryptology*, 15(4):271–284, 2002.

[7] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. In *CRYPTO'92*, Springer-Verlag (LNCS 740), pages 390–420, 1992.

[8] M. Blum. Coin Flipping by Phone. *IEEE Spring COMPCOM*, pages 133–137, February 1982.

[9] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In *CRYPTO'97*, Springer-Verlag (LNCS 1233), pages 425–439, 1997.

[10] G. Brassard, C. Crepeau and M. Yung. Constant-round perfect zero-knowledge computationally convincing protocols. In *Theoretical Computer Science*, 84(1):23–52, 1991.

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

[12] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private Information Retrieval. In *36th FOCS*, pp. 41–50, 1995.

[13] I. Damgard, T. Pederson and B. Pfitzmann. On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures. In *CRYPTO'93*, Springer-Verlag (LNCS 773), pages 250–265, 1993.

[14] S. Even, O. Goldreich and A. Lempel. A Randomized Protocol for Signing Contracts. *Communications of the ACM*, **28**, pages 637–647, 1985.

[15] R. Fagin, M. Naor and P. Winkler. Comparing Information Without Leaking It. *Communications of the ACM,* **39**, pages 77–85, 1996.

[16] U. Feige. *Alternative Models for Zero Knowledge Interactive Proofs*. Ph.D. Thesis, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1990. Available from `http://www.wisdom.weizmann.ac.il/~feige`.

[17] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 526–544, 1989.

[18] O. Goldreich. *Secure Multi-Party Computation*. Manuscript. Preliminary version, 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[19] O. Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools*. Cambridge University Press, 2001.

[20] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–189, 1996.

[21] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[22] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90*, Spring-Verlag (LNCS 537), pages 77–93, 1990.

[23] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SICOMP*, 18(1):186–208, 1989.

[24] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *JACM*, 38(1):691–729, 1991.

[25] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987. For details see [18].

[26] S. Halevi and S. Micali. More on Proofs of Knowledge. Cryptology ePrint Archive, 1998/015, 1998.

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

[28] M. Naor. Bit Commitment using Pseudorandom Generators. *Journal of Cryptology*, 4(2):151–158, 1991.

[29] M. Naor and K. Nissim. Communication Preserving Protocols for Secure Function Evaluation. In *33rd STOC*, pages 590–599, 2001.

[30] M. Naor, R. Ostrovsky, R. Venkatesan and M. Yung. Zero-Knowledge Arguments for NP can be Based on General Assumptions. *Journal of Cryptology*, 11(2):87–108, 1998.

[31] M. Naor and B. Pinkas. Oblivious Transfer and Polynomial Evaluation. In *31st STOC*, pages 245–254, 1999.

[32] M. Naor and M. Yung. Universal One-Way Hash Functions and their Cryptographic Applications. In *21st STOC*, pages 33–43, 1989.

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

A Expected Polynomial-Time Adversaries and Zero-Knowledge Arguments of Knowledge

In this work we considered a slightly non-standard definition of secure computation in which the real-model adversary is allowed to run in expected polynomial-time (rather than being limited to strict polynomial-time). This means that the simulator for the zero-knowledge argument system must remain expected polynomial-time also in the case that the verifier runs in expected polynomial-time; likewise for the extractor working with an expected polynomial-time prover of knowledge. (When computing the running-time of the simulator, we include the running-time of the verifier; likewise for the extractor and prover.) However, the standard definitions of zero-knowledge arguments of knowledge (and specifically the one used by Feige and Shamir [17]) refer to strict polynomial-time adversaries only.

In this appendix, we show that the [17] argument system remains a zero-knowledge argument of knowledge even when the verifier and prover may run in expected polynomial-time. However, it seems that this does *not* hold for all zero-knowledge arguments of knowledge. In particular, as we will see, the zero-knowledge proof of [20] seems *not* to remain zero-knowledge if the verifier may run in expected polynomial-time.

Failure of the naive approach. A naive approach to solving the above problem would be to simply truncate the execution of the verifier after it exceeds its expected running-time by "too much". The intuition behind such a suggestion is that the output of the truncated verifier is very close to that of the original one. Furthermore, this truncated verifier runs in strict polynomial-time. Thus, we can simply apply the existing simulator to the truncated verifier and we are done. The conclusion would be that any zero-knowledge argument remains zero-knowledge, even when the verifier runs in expected polynomial-time. However, the above intuition is not correct in our context here. For example, consider a very simple (cheating) verifier V^* who with probability $1/p(n)$, for some polynomial $p(\cdot)$, plays the honest verifier strategy, and otherwise it aborts (not sending any message). In addition, when V^* does not abort, it runs for $p(n)$ steps before sending any messages. Now, the expected running-time of V^* equals that of the honest verifier. Thus, according to the above strategy, we would truncate V^*'s execution after it exceeds, say, n^2 times the running-time of the honest verifier. The point is that if $p(n)$ is larger than this value, then the truncated V^* *never replies*. Thus, the simulation of the truncated V^* can be distinguished from real executions of V^* with probability $1/p(n)$. (Of course, this specific verifier can be simulated because $p(n)$ is a polynomial. Nevertheless, the example suffices for ruling out the above strategy as presented. For more discussion and examples, see [16, Section 3] and [2].)

A.1 The Zero-Knowledge Arguments of Knowledge of [17]

We assume that the reader is familiar with the [17] system of zero-knowledge arguments of knowledge.

A.1.1 Extraction from an expected polynomial-time prover

In this section, we consider a generic extractor K that works in the following way. Let P^* be a prover with input x and auxiliary input y. Then, the extractor K chooses a uniformly distributed random-tape r for P^*, defining a prover $P^*_{x,y,r}$, and works as follows:[18]

1. Interact with $P^*_{x,y,r}$ and play the honest verifier V. If V rejects, then halt and output $\perp$. If V accepts, then continue to the next stage.

2. Let $q(\cdot)$ be a fixed, predetermined polynomial. Then, interact with $P^*_{x,y,r}$ and play the honest verifier V in many interactions, until $q(|x|)$ accepting executions are achieved. In each execution, use fresh randomness for V (whereas P^*'s randomness is fixed for all executions).

3. Given the transcripts of $q(|x|)$ accepting executions with a prover who uses the same randomness in each one, K computes a witness for x. (With overwhelming probability this task can be done in polynomial-time.)

The extractor for the system of arguments of knowledge of [17] (which is actually n parallel executions of Blum's proof of knowledge of Hamiltonicity) works in exactly this way. For this scheme, the polynomial $q(\cdot)$ of Stage 2 is $q(n) \equiv 1$ (that is, one additional successful execution is enough, see [16]). Other proofs of knowledge can also be cast in this setting (e.g., the extractor for the [24] proof of knowledge based on coloring can also work in this way). For simplicity, we show the expected running-time of K when $q(n) \equiv 1$. The general case is easily derived.

We begin by analyzing the running-time of Stage 2 of the extraction procedure of K (Stage 3 can be ignored since it merely involves a fixed polynomial-time computation). First, denote by $t_{\mathrm{P}}(r, s)$, the running-time of P^* in an execution with the honest verifier V, where P^*'s random-tape equals r and V's random-tape equals s (we note that by P^* we really mean $P^*_{x,y}$). Now, notice

[18] We note that a very similar type of extractor was explicitly considered by [26], who called it a "conservative knowledge extractor". However, their motivation was very different.

that when P^* interacts with the honest verifier, its messages are a deterministic function of r and s. Therefore, $t_{\rm P}(r,s)$ is a *fixed* running-time. Next, let $\chi_{\rm P}(r,s) = 1$ if and only if V accepts when interacting with P^*, when P^* and V's respective random tapes are r and s. Then, the running time of Stage 2 of the extraction procedure exactly corresponds to the random variable T_r in the following game, where r is a fixed value (and the probability is taken over the extractor's coins). Recall that r is fixed for all of the executions, whereas s is different in each execution. Assume, for simplicity, that the length of V's random-tape s is n.

- *Initialize* $T_r = 0$.
- *Iterate:*
    1. *Choose* $s \in_R \{0,1\}^n$
    2. *Let* $T_r = T_r + t_{\rm P}(r,s)$
    3. *If* $\chi_{\rm P}(r,s) = 1$*, then* halt.

We now compute the expected value of T_r. (In the claim below, p corresponds to the probability that P^*, with randomness r, successfully completes the proof when the random tape s is uniformly chosen.)

Claim A.1 *For a given* r*, denote by* p *the probability* (*over a uniform choice of* s) *that* $\chi_{\rm P}(r,s) = 1$. *Then,*

$$\mathrm{Exp}[T_r] = \frac{1}{p} \cdot \frac{1}{2^n} \sum_{s \in \{0,1\}^n} t_{\rm P}(r,s)$$

Proof: Denote the set of "good" random-tapes by $G \stackrel{\rm def}{=} \{s \mid \chi_{\rm P}(r,s) = 1\}$. Then, we have that $|G| = p \cdot 2^n$, $|\overline{G}| = (1-p) \cdot 2^n$ and

$$\mathrm{Exp}[t_{\rm P}(r,G)] = \frac{1}{|G|} \sum_{s \in G} t_{\rm P}(r,s)$$

and

$$\mathrm{Exp}[t_{\rm P}(r,\overline{G})] = \frac{1}{|\overline{G}|} \sum_{s \notin G} t_{\rm P}(r,s)$$

Now, the expected number of times that the process of choosing a random-tape s is repeated until (but not including) success equals $\frac{1}{p} - 1$. Thus, we first show the following intuitive equality:

$$\mathrm{Exp}[T_r] = \left(\frac{1}{p} - 1\right) \cdot \mathrm{Exp}[t_{\rm P}(r,\overline{G})] + \mathrm{Exp}[t_{\rm P}(r,G)] \qquad (6)$$

Eq. (6) is shown as follows. First, notice that we can divide the expectation of T_r into events relating to how many iterations of the game occur until halt is reached. That is,

$$\mathrm{Exp}[T_r] = \sum_k \Pr[k \text{ iterations until } \mathsf{halt}] \cdot \mathrm{Exp}[T_r \mid k \text{ iterations until } \mathsf{halt}]$$

Noticing further that,

$$\mathrm{Exp}[T_r \mid k \text{ iterations until } \mathsf{halt}] = (k-1) \cdot \mathrm{Exp}[t_{\rm P}(r,\overline{G})] + \mathrm{Exp}[t_{\rm P}(r,G)]$$

we have:

$$\begin{aligned}
\mathrm{Exp}[T_r] &= \sum_k (1-p)^{k-1} p \cdot \Big((k-1)\cdot \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] + \mathrm{Exp}[t_{\mathrm{P}}(r,G)]\Big) \\
&= \sum_k (1-p)^{k-1} p \cdot k \cdot \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] - \sum_k (1-p)^{k-1} p \cdot \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] \\
&\quad + \sum_k (1-p)^{k-1} p \cdot \mathrm{Exp}[t_{\mathrm{P}}(r,G)] \\
&= \left(\frac{1}{p}-1\right) \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] + \mathrm{Exp}[t_{\mathrm{P}}(r,G)]
\end{aligned}$$

proving Eq. (6). Then,

$$\begin{aligned}
\mathrm{Exp}[T_r] &= \left(\frac{1}{p}-1\right) \cdot \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] + \mathrm{Exp}[t_{\mathrm{P}}(r,G)] \\
&= \frac{1}{p}\Big((1-p)\cdot \mathrm{Exp}[t_{\mathrm{P}}(r,\overline{G})] + p \cdot \mathrm{Exp}[t_{\mathrm{P}}(r,G)]\Big) \\
&= \frac{1}{p}\left(\frac{1}{2^n}\sum_{s\notin G} t_{\mathrm{P}}(r,s) + \frac{1}{2^n}\sum_{s\in G} t_{\mathrm{P}}(r,s)\right) \\
&= \frac{1}{p}\cdot\frac{1}{2^n}\sum_{s\in\{0,1\}^n} t_{\mathrm{P}}(r,s)
\end{aligned}$$

completing the proof. ■

Before continuing, we restate Claim A.1 in terms of the running-time of P^*. That is, let $t_{\mathrm{P}}(r)$ denote the random variable of the running-time of P^*, with randomness r, when interacting with the honest verifier (with a uniformly chosen random-tape). Then, we have that

$$\mathrm{Exp}[t_{\mathrm{P}}(r)] = \frac{1}{2^n}\sum_{s\in\{0,1\}^n} t_{\mathrm{P}}(r,s)$$

Therefore, by Claim A.1 we have that the expected running-time of Stage 2 is

$$\mathrm{Exp}[T_r] = \frac{1}{p}\cdot \mathrm{Exp}[t_{\mathrm{P}}(r)]$$

We are now ready to conclude the analysis of the overall running-time of the extractor. Recall that the extractor first (honestly) verifies the proof from the prover. If it succeeds (and this occurs with probability p), then Stage 2 of the extraction procedure is run. Therefore, the expected running-time (when r is fixed) equals:

$$\mathrm{Exp}[t_{\mathrm{P}}(r)] + p\cdot \mathrm{Exp}[T_r] = \mathrm{Exp}[t_{\mathrm{P}}(r)] + p\cdot\frac{1}{p}\cdot \mathrm{Exp}[t_{\mathrm{P}}(r)] = 2\cdot \mathrm{Exp}[t_{\mathrm{P}}(r)]$$

Until this point, we have considered a fixed r. However, the extractor chooses P^*'s random-tape uniformly. Therefore, the expected running-time of the extractor (over all possible r's) equals exactly twice the expected running time of P^*. Recalling that P^* is expected polynomial-time, this completes the analysis.

A.1.2 Simulation for expected polynomial-time verifiers

Loosely speaking, the Feige-Shamir argument system works in two stages. In the first stage, the verifier proves that it knows some secret, using a (witness hiding) proof of knowledge. Then, in the second stage, the prover proves that it knows the witness to the input statement. However, this second proof is designed in such a way so that if the secret from the first stage is known, then the proof can be successfully proven without really knowing the witness to the input statement. Furthermore, this proof is indistinguishable from a real proof. The zero-knowledge simulator for this argument system therefore works by extracting the secret from the first stage and then using it to prove the proof of the second stage. Thus, in order to show that the simulator remains expected polynomial-time when the verifier may run in expected polynomial-time, we must first show that the extraction procedure does not take "too long". However, in Section A.1.1, we have already shown that the extraction procedure in this setting terminates in expected polynomial-time. It therefore remains to show that the second stage of the argument system also terminates in expected polynomial-time.

The basis for this claim is as follows. First, notice that the second stage of the simulation requires no rewinding and works by running a single proof with the verifier V^*. Furthermore, and this is the key point, there exists a (not necessarily efficient) interactive machine who interacts with V^* and generates exactly the same distribution of messages as the simulator (without being given the secret obtained by the extractor from the first stage). This machine simply extracts the secret by itself (it is not computationally bounded and can therefore do this) and uses it in the same way as the simulator. Since the verifier is expected polynomial-time when interacting with this machine, this also holds when interacting with the simulator of the second stage. We conclude that the entire simulation terminates in expected polynomial-time.

A.2 The Zero-Knowledge Proof System of [20]

In this section, we show that in contrast to the argument system of [17], the simulator provided for the zero-knowledge proof system of Goldreich and Kahan [20] does *not* necessarily remain expected polynomial-time when simulating for an expected polynomial-time verifier. We stress that we do not claim that it is impossible to construct a *different* simulator that will have this property. However, it seems from our analysis below that it would be difficult to construct such a simulator.

For this section, we assume familiarity with the proof system of [20]. Recall that in this proof system, the verifier begins by committing to its random query string (using a perfectly hiding commitment scheme). The parties then continue by running the zero-knowledge proof for 3-coloring of [24] in parallel, using the verifier's queries from the first step. That is, the prover sends (perfectly binding) commitments to randomly permuted colorings of the graph. Then, the verifier decommits, revealing its query string. Finally, the prover answers according to the revealed queries. The exact soundness of the system depends on the number of parallel executions and is negligible. We denote the soundness of the proof system by $\mu(n)$ (i.e., the probability that V accepts and the graph is not 3-colorable is less than $\mu(n)$). We stress that the exact value of $\mu(n)$ can be calculated and this does not depend on any computational assumptions.

Before proceeding, we note that the prover's commitments (to the colorings) are only computationally hiding. Therefore, given enough time, it is possible to break them and extract the committed values (which in this case equals the coloring itself). In particular, in time 2^n (where n is the security parameter), it is possible to break these commitments.

Loosely speaking, we will construct a verifier that with probability 2^{-n} runs for 2^n steps and breaks the prover's commitments. Then, the verifier checks if these commitments are "real" or "convincing garbage", where convincing garbage is a commitment that would convince the verifier, yet does not constitute a legal 3-coloring. Then, if it finds that it received convincing garbage, it enters a very long loop (and otherwise continues like the honest verifier). The key point is that although the simulator can generate convincing garbage, the probability that any (even all-powerful) machine can do the same is negligible. Therefore, when interacting in a real protocol execution, the verifier enters the loop with very small probability. On the other hand, the simulator *always* generates convincing garbage. By correctly choosing the number of steps run by the verifier in the loop, we can ensure that its overall expected-time during simulation is super-polynomial. Details follow.

The Verifier V^*:

1. Send the prover a perfectly-hiding commitment to a random query string q, exactly according to the protocol specification.

2. Upon receiving the prover's commitments (to many 3-colorings) do the following:

   - With probability 2^{-n}, break the prover's commitments and obtain the values. (This takes time at most 2^n.)
     If the commitments are such that *none* of them constitute a valid 3-coloring, yet they all answer the query string q perfectly,[19] then run for $2^n/\mu(n)$ steps.

3. Continue in the same way as the honest verifier.

We first claim that V^* is an expected polynomial-time machine. This can be seen as follows. V^* attempts to break the commitments with probability 2^{-n}. Therefore, the 2^n time it takes to do this contributes only a single step to its expected running-time. Furthermore, the probability that any machine sends a commitment of the form that causes V^* to run for $2^n/\mu(n)$ steps is at most $\mu(n)$ (by the soundness of the proof system). Therefore, V^* runs for $2^n/\mu(n)$ steps only with probability $2^{-n} \cdot \mu(n)$ and this also contributes only a single step to its expected running-time. That is, the expected running-time of V^* is at most:

$$\frac{1}{2^n} \cdot \left(2^n + \mu(n) \cdot \frac{2^n}{\mu(n)} + p(n)\right) + \left(1 - \frac{1}{2^n}\right) \cdot p(n) = \text{poly}(n)$$

where $p(n)$ equals the running-time of the honest verifier.

On the other hand, we claim that the [20] simulator runs for a super-polynomial number of steps, when simulating for this V^*. In particular, this simulator *always* sends a commitment that causes V^* to run in time $2^n/\mu(n)$. Therefore, the expected running time of the simulator of V^* is greater than:

$$\frac{1}{2^n} \cdot \left(2^n + 1 \cdot \frac{2^n}{\mu(n)} + p(n)\right) + \left(1 - \frac{1}{2^n}\right) \cdot p(n) > \frac{1}{\mu(n)}$$

Since $\mu(n)$ is a negligible function, we have that the expected running-time of the simulator is super-polynomial. Therefore, the simulator presented by [20] for demonstrating the zero-knowledge property of their proof system is only expected polynomial-time if the verifier is limited to strict polynomial-time.

[19] A commitment answers the query string perfectly if for every edge in the query string, it turns out that the committed colors of the vertices specified by the edge are different. Therefore, such a commitment would convince the honest verifier in the proof.

We conclude with an open question that is raised by the above observation regarding the [20] proof system. That is, is it possible to construct a constant-round zero-knowledge *proof* system for $\mathcal{NP}$ (with negligible soundness), that remains zero-knowledge for an expected polynomial-time verifier?